## Contents

### About This Manual
- Product Website and Support ................................................................. xix  
- Related Publications .............................................................................. xx 
- Conventions ............................................................................................ xxii 
  - Platform Labels ................................................................................ xxiv 

### Chapter 1 Overview of the Libraries
- Portability Routines .................................................................................. 1-2 
- National Language Support Routines (W*32, W*64) .......................... 1-8 
  - NLS Date and Time Format (W*32, W*64) ........................................ 1-10 
- POSIX* Routines .................................................................................... 1-11 
- QuickWin and Graphics Routines (W*32, W*64) ................................. 1-16 
- Dialog Routines (W*32) ........................................................................ 1-22 
- COM and AUTO Routines (W*32) ............................................................ 1-23 
- Miscellaneous Run-Time Routines ........................................................ 1-25 

### Chapter 2 Descriptions of the Library Routines
- ABORT ...................................................................................................... 2-2 
- ABOUTBOXQQ ....................................................................................... 2-2 
- ACCESS .................................................................................................. 2-3 
- ALARM ................................................................................................... 2-4 
- APPENDMENUQQ .................................................................................. 2-6 
- ARC, ARC_W ........................................................................................ 2-8 
- AUTOAddArg ........................................................................................ 2-10 
- AUTOAllocateInvokeArgs ........................................................................ 2-11
AUTODeallocateInvokeArgs ................................................................. 2-12
AUTOGetExceptInfo ................................................................................ 2-12
AUTOGetProperty .................................................................................. 2-13
AUTOGetPropertyByID ................................................................. 2-14
AUTOGetPropertyInvokeArgs ............................................................. 2-15
AUTOInvoke ....................................................................................... 2-15
AUTOSetProperty ............................................................................... 2-16
AUTOSetPropertyByID ....................................................................... 2-17
AUTOSetPropertyInvokeArgs ............................................................. 2-17
BEEPQQ ............................................................................................. 2-18
BESJ0, BESJ1, BESJN, BESY0, BESY1, BESYN .................................. 2-18
BIC, BIS .............................................................................................. 2-19
BIT ....................................................................................................... 2-20
BSEARCHQQ ..................................................................................... 2-21
CDFLOAT ............................................................................................. 2-23
CHANGEDIRQQ .................................................................................. 2-23
CHANGEDRIVEQQ .......................................................................... 2-24
CHDIR .................................................................................. 2-25
CHMOD .............................................................................................. 2-26
CLEARSCREEN ............................................................................... 2-28
CLEARSTATUSFPQQ ........................................................................ 2-29
CLICKMENUQQ ............................................................................... 2-30
CLOCK .......................................................................................... 2-31
CLOCKX ............................................................................................ 2-31
COMAddObjectReference ................................................................. 2-32
COMCLSIDFromProgID .................................................................... 2-32
COMCLSIDFromString ...................................................................... 2-33
COMCreateObjectByGUID ............................................................... 2-33
COMCreateObjectByProgID ............................................................. 2-34
COMGetActiveObjectByGUID .......................................................... 2-34
COMGetActiveObjectByProgID ......................................................... 2-35
COMGetFileObject ............................................................................... 2-35
COMInitialize ..................................................................................... 2-36
COMIsEqualGUID ........................................................................... 2-37
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMITQQ</td>
<td>2-38</td>
</tr>
<tr>
<td>COMPLINT, COMPLREAL, COMPLLOG</td>
<td>2-39</td>
</tr>
<tr>
<td>COMQueryInterface</td>
<td>2-39</td>
</tr>
<tr>
<td>COMReleaseObject</td>
<td>2-40</td>
</tr>
<tr>
<td>COMStringFromGUID</td>
<td>2-40</td>
</tr>
<tr>
<td>COMUninitialize</td>
<td>2-41</td>
</tr>
<tr>
<td>CSMG</td>
<td>2-41</td>
</tr>
<tr>
<td>CTIME</td>
<td>2-42</td>
</tr>
<tr>
<td>DATE</td>
<td>2-43</td>
</tr>
<tr>
<td>DATE4</td>
<td>2-44</td>
</tr>
<tr>
<td>DBESJ0, DBESJ1, DBESJN, DBESY0, DBESY1, DBESYN</td>
<td>2-44</td>
</tr>
<tr>
<td>DCLK</td>
<td>2-45</td>
</tr>
<tr>
<td>DELDIRQQ</td>
<td>2-46</td>
</tr>
<tr>
<td>DELETEMENUQQ</td>
<td>2-46</td>
</tr>
<tr>
<td>DELFILESQQ</td>
<td>2-47</td>
</tr>
<tr>
<td>DFLOATI, DFLOATJ, DFLOATK</td>
<td>2-48</td>
</tr>
<tr>
<td>DISPLAYCURSOR</td>
<td>2-49</td>
</tr>
<tr>
<td>DLGEXIT</td>
<td>2-49</td>
</tr>
<tr>
<td>DLGFLUSH</td>
<td>2-50</td>
</tr>
<tr>
<td>DLGGET, DLGGETINT, DLGGETLOG, DLGGETCHAR</td>
<td>2-52</td>
</tr>
<tr>
<td>DLGINIT, DLGINITWITHRESOURCEHANDLE</td>
<td>2-53</td>
</tr>
<tr>
<td>DLGISDLGMESSAGE, DLGISDLGMESSAGEWITHDLG</td>
<td>2-55</td>
</tr>
<tr>
<td>DLGMODAL, DLGMODALWITHPARENT</td>
<td>2-56</td>
</tr>
<tr>
<td>DLGMODELESS</td>
<td>2-58</td>
</tr>
<tr>
<td>DLGSENDCTRLMESSAGE</td>
<td>2-60</td>
</tr>
<tr>
<td>DLGSET, DLGSETINT, DLGSETLOG, DLGSETCHAR</td>
<td>2-62</td>
</tr>
<tr>
<td>DLGSETCTRLLEVENTHANDLER</td>
<td>2-63</td>
</tr>
<tr>
<td>DLGSETRETURN</td>
<td>2-65</td>
</tr>
<tr>
<td>DLGSETSUB</td>
<td>2-66</td>
</tr>
<tr>
<td>DLGSETTITLE</td>
<td>2-69</td>
</tr>
<tr>
<td>DLGUNINIT</td>
<td>2-70</td>
</tr>
<tr>
<td>DRAND, DRANDM</td>
<td>2-70</td>
</tr>
<tr>
<td>DRANSET</td>
<td>2-72</td>
</tr>
<tr>
<td>DTIME</td>
<td>2-72</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ELLIPSE, ELLIPSE_W</td>
<td>2-73</td>
</tr>
<tr>
<td>ETIME</td>
<td>2-75</td>
</tr>
<tr>
<td>FDATE</td>
<td>2-76</td>
</tr>
<tr>
<td>FGETC</td>
<td>2-76</td>
</tr>
<tr>
<td>FINDFILEQQ</td>
<td>2-77</td>
</tr>
<tr>
<td>FLOODFILL, FLOODFILL_W</td>
<td>2-78</td>
</tr>
<tr>
<td>FLOODFILLRGB, FLOODFILLRGB_W</td>
<td>2-80</td>
</tr>
<tr>
<td>FLUSH</td>
<td>2-81</td>
</tr>
<tr>
<td>FOCUSQQ</td>
<td>2-82</td>
</tr>
<tr>
<td>FOR_DESCRIPTOR_ASSUME</td>
<td>2-83</td>
</tr>
<tr>
<td>FOR_GET_FPE</td>
<td>2-86</td>
</tr>
<tr>
<td>FOR_SET_FPE</td>
<td>2-87</td>
</tr>
<tr>
<td>FOR_SET_REENTRANCY</td>
<td>2-88</td>
</tr>
<tr>
<td>FPUTC</td>
<td>2-89</td>
</tr>
<tr>
<td>FSEEK</td>
<td>2-90</td>
</tr>
<tr>
<td>FSTAT</td>
<td>2-91</td>
</tr>
<tr>
<td>FTLL, FTLLI8</td>
<td>2-94</td>
</tr>
<tr>
<td>FULLPATHQQ</td>
<td>2-95</td>
</tr>
<tr>
<td>GERROR</td>
<td>2-96</td>
</tr>
<tr>
<td>GETACTIVEQQ</td>
<td>2-97</td>
</tr>
<tr>
<td>GETARCINFO</td>
<td>2-98</td>
</tr>
<tr>
<td>GETBKCOLOR</td>
<td>2-99</td>
</tr>
<tr>
<td>GETBKCOLORRGB</td>
<td>2-100</td>
</tr>
<tr>
<td>GETC</td>
<td>2-101</td>
</tr>
<tr>
<td>GETCHARQQ</td>
<td>2-102</td>
</tr>
<tr>
<td>GETCOLOR</td>
<td>2-104</td>
</tr>
<tr>
<td>GETCOLORRRGB</td>
<td>2-105</td>
</tr>
<tr>
<td>GETCONTROLFPQQ</td>
<td>2-107</td>
</tr>
<tr>
<td>GETCURRENTPOSITION, GETCURRENTPOSITION_W</td>
<td>2-109</td>
</tr>
<tr>
<td>GETCWD</td>
<td>2-110</td>
</tr>
<tr>
<td>GETDAT</td>
<td>2-111</td>
</tr>
<tr>
<td>GETDRIVEDIRQQ</td>
<td>2-112</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>GETDRIVESIZEQQ</td>
<td>2-113</td>
</tr>
<tr>
<td>GETDRIVESQQ</td>
<td>2-115</td>
</tr>
<tr>
<td>GETENV</td>
<td>2-115</td>
</tr>
<tr>
<td>GETENVQQ</td>
<td>2-116</td>
</tr>
<tr>
<td>GETEXCEPTIONPTRSSQ</td>
<td>2-118</td>
</tr>
<tr>
<td>GETEXITQQ</td>
<td>2-119</td>
</tr>
<tr>
<td>GETFILEINFOQQ</td>
<td>2-120</td>
</tr>
<tr>
<td>GETFILLMASK</td>
<td>2-123</td>
</tr>
<tr>
<td>GETFONTINFO</td>
<td>2-124</td>
</tr>
<tr>
<td>GETGID</td>
<td>2-126</td>
</tr>
<tr>
<td>GETGTEXTEXTENT</td>
<td>2-126</td>
</tr>
<tr>
<td>GETGTEXTROTATION</td>
<td>2-127</td>
</tr>
<tr>
<td>GETHWNDQQ</td>
<td>2-128</td>
</tr>
<tr>
<td>GETIMAGE, GETIMAGE_W</td>
<td>2-129</td>
</tr>
<tr>
<td>GETLASTERROR</td>
<td>2-130</td>
</tr>
<tr>
<td>GETLASTERRORQQ</td>
<td>2-130</td>
</tr>
<tr>
<td>GETLINESTYLE</td>
<td>2-132</td>
</tr>
<tr>
<td>GETLOG</td>
<td>2-133</td>
</tr>
<tr>
<td>GETPHYSCOORD</td>
<td>2-133</td>
</tr>
<tr>
<td>GETPID</td>
<td>2-133</td>
</tr>
<tr>
<td>GETPIXEL, GETPIXEL_W</td>
<td>2-135</td>
</tr>
<tr>
<td>GETPIXELRGB, GETPIXELRGB_W</td>
<td>2-136</td>
</tr>
<tr>
<td>GETPIXELS</td>
<td>2-138</td>
</tr>
<tr>
<td>GETPIXELSRGB</td>
<td>2-139</td>
</tr>
<tr>
<td>GETPOS, GETPOSI8</td>
<td>2-141</td>
</tr>
<tr>
<td>GETSTATUSFPQQ</td>
<td>2-141</td>
</tr>
<tr>
<td>GETSTRQQ</td>
<td>2-143</td>
</tr>
<tr>
<td>GETTEXTCOLOR</td>
<td>2-144</td>
</tr>
<tr>
<td>GETTEXTCOLORRGB</td>
<td>2-145</td>
</tr>
<tr>
<td>GETTEXTPOSITION</td>
<td>2-147</td>
</tr>
<tr>
<td>GETTEXTWINDOW</td>
<td>2-148</td>
</tr>
<tr>
<td>GETTIM</td>
<td>2-149</td>
</tr>
<tr>
<td>GETTIMEOFDAY</td>
<td>2-149</td>
</tr>
<tr>
<td>GETUID</td>
<td>2-150</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>GETUNITQQ</td>
<td>2-150</td>
</tr>
<tr>
<td>GETVIEWCOORD, GETVIEWCOORD_W</td>
<td>2-151</td>
</tr>
<tr>
<td>GETWINDOWCONFIG</td>
<td>2-152</td>
</tr>
<tr>
<td>GETWINDOWCOORD</td>
<td>2-154</td>
</tr>
<tr>
<td>GETWRITEMODE</td>
<td>2-155</td>
</tr>
<tr>
<td>GETWSIZEQQ</td>
<td>2-156</td>
</tr>
<tr>
<td>GMTIME</td>
<td>2-157</td>
</tr>
<tr>
<td>GRSTATUS</td>
<td>2-159</td>
</tr>
<tr>
<td>HOSTNAM</td>
<td>2-162</td>
</tr>
<tr>
<td>IDATE</td>
<td>2-163</td>
</tr>
<tr>
<td>IDATE4</td>
<td>2-164</td>
</tr>
<tr>
<td>IDFLOAT</td>
<td>2-164</td>
</tr>
<tr>
<td>IEEE_FLAGS</td>
<td>2-165</td>
</tr>
<tr>
<td>IEEE_HANDLER</td>
<td>2-168</td>
</tr>
<tr>
<td>IERRNO</td>
<td>2-170</td>
</tr>
<tr>
<td>IFLOATI, IFLOATJ</td>
<td>2-171</td>
</tr>
<tr>
<td>IMAGESIZE, IMAGESIZE_W</td>
<td>2-172</td>
</tr>
<tr>
<td>INCHARQQ</td>
<td>2-173</td>
</tr>
<tr>
<td>INITIALIZEFONTS</td>
<td>2-174</td>
</tr>
<tr>
<td>INITIALSETTINGS</td>
<td>2-175</td>
</tr>
<tr>
<td>INMAX</td>
<td>2-176</td>
</tr>
<tr>
<td>INQFOCUSQQ</td>
<td>2-176</td>
</tr>
<tr>
<td>INSERTMENUQQ</td>
<td>2-177</td>
</tr>
<tr>
<td>INTC</td>
<td>2-179</td>
</tr>
<tr>
<td>INTEGRERTORGB</td>
<td>2-180</td>
</tr>
<tr>
<td>IPXFARGC</td>
<td>2-181</td>
</tr>
<tr>
<td>IPXFCONST</td>
<td>2-181</td>
</tr>
<tr>
<td>IPXFLENTRIM</td>
<td>2-182</td>
</tr>
<tr>
<td>IPXFWEXITSTATUS</td>
<td>2-182</td>
</tr>
<tr>
<td>IPXFWSTOPSIG</td>
<td>2-183</td>
</tr>
<tr>
<td>IPXFWTERMSIG</td>
<td>2-184</td>
</tr>
<tr>
<td>IRAND, IRANDM</td>
<td>2-184</td>
</tr>
<tr>
<td>IRANGE</td>
<td>2-185</td>
</tr>
<tr>
<td>IRANSET</td>
<td>2-186</td>
</tr>
</tbody>
</table>
ISATTY ................................................................................................... 2-186
ITIME .................................................................................................... 2-187
JABS ...................................................................................................... 2-187
JDATE .................................................................................................. 2-188
JDATE4 .................................................................................................. 2-188
KILL ...................................................................................................... 2-189
LCWRQQ ............................................................................................. 2-190
LINETO, LINETO_W ........................................................................... 2-191
LINETOAR ........................................................................................... 2-192
LINETOAREX ...................................................................................... 2-193
LNBLNK ............................................................................................... 2-195
LOADIMAGE, LOADIMAGE_W ........................................................... 2-196
LONG .................................................................................................... 2-197
LSTAT ................................................................................................... 2-197
LTIME .................................................................................................. 2-198
MAKEDIRQQ ...................................................................................... 2-200
MBCharLen .......................................................................................... 2-201
MBConvertMBToUnicode .................................................................. 2-201
MBConvertUnicodeToMB .................................................................. 2-202
MBCurMax .......................................................................................... 2-204
MBINCHARQQ .................................................................................... 2-204
MBINDEX ............................................................................................ 2-205
MBJISToJMS, MBJMSToJIS ................................................................. 2-205
MBLead ............................................................................................... 2-206
MBLen ................................................................................................. 2-207
MBLen_Trim ....................................................................................... 2-208
MBLGE, MBLGT, MBLLE, MBLLT, MBLEM, MBLNE ......................... 2-208
MBNext ............................................................................................... 2-210
MBPrev ............................................................................................... 2-211
MBSCAN ............................................................................................... 2-211
MBStrLead .......................................................................................... 2-212
MBVERIFY ........................................................................................... 2-213
MESSAGEBOXQQ ................................................................................ 2-213
MODIFYMENUFLAGSQQ ..................................................................... 2-215
MODIFYMENUROUTINEQQ ................................................................. 2-216
MODIFYMENUSTRINGQQ................................................................. 2-218
MOVETO, MOVETO_W .................................................................... 2-219
NLSEnumCodepages ...................................................................... 2-220
NLSEnumLocales .......................................................................... 2-221
NLSFormatCurrency .................................................................... 2-222
NLSFormatDate ............................................................................. 2-223
NLSFormatNumber ...................................................................... 2-224
NLSFormatTime ............................................................................. 2-225
NLSSetEnvironmentCodepage ...................................................... 2-227
NLSSetLocale ............................................................................... 2-227
NLSSetLocaleInfo ......................................................................... 2-228
NLSSetEnvironmentCodepage ...................................................... 2-234
NLSSetLocale ............................................................................... 2-235
OUTGTEXT ..................................................................................... 2-236
OUTTEXT ........................................................................................ 2-238
PACKTIMEQQ ................................................................................ 2-238
PASSEVENTSQQ ............................................................................. 2-240
PEEKCHARQQ ................................................................................ 2-243
PEROR ............................................................................................ 2-244
PIE, PIE_W ..................................................................................... 2-245
POLYBEZIER, POLYBEZIER_W .......................................................... 2-247
POLYBEZIERTO, POLYBEZIERTO_W ............................................... 2-252
POLYGON, POLYGON_W ................................................................. 2-253
POLYLINEXQQ ............................................................................... 2-256
PUTC ............................................................................................... 2-257
PUTIMAGE, PUTIMAGE_W ............................................................. 2-258
PXFTYPE>GET ................................................................................ 2-260
PXFTYPE>SET ................................................................................ 2-262
PXFA<TYPE>GET ............................................................................ 2-263
PXFA<TYPE>SET ............................................................................ 2-264
PXFACCESS .................................................................................... 2-265
PXFALARM ..................................................................................... 2-266
PXFCALLSUBHANDLE ................................................................. 2-267
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXFCFGETISPEED</td>
<td>2-268</td>
</tr>
<tr>
<td>PXFCFGETOSPEED</td>
<td>2-269</td>
</tr>
<tr>
<td>PXFCFSETISPEED</td>
<td>2-269</td>
</tr>
<tr>
<td>PXFCFSETOSPEED</td>
<td>2-270</td>
</tr>
<tr>
<td>PXFCHDIR</td>
<td>2-270</td>
</tr>
<tr>
<td>PXFCHMOD</td>
<td>2-271</td>
</tr>
<tr>
<td>PXFCHOWN</td>
<td>2-272</td>
</tr>
<tr>
<td>PXFCLEARENV</td>
<td>2-272</td>
</tr>
<tr>
<td>PXFCLOSE</td>
<td>2-273</td>
</tr>
<tr>
<td>PXFCLOSEDIR</td>
<td>2-273</td>
</tr>
<tr>
<td>PXFCNTL</td>
<td>2-274</td>
</tr>
<tr>
<td>PXFCONST</td>
<td>2-276</td>
</tr>
<tr>
<td>PXFCREAT</td>
<td>2-276</td>
</tr>
<tr>
<td>PXFCTERMID</td>
<td>2-277</td>
</tr>
<tr>
<td>PXFDUP, PXFDUP2</td>
<td>2-278</td>
</tr>
<tr>
<td>PXFE&lt;TYPE&gt;GET</td>
<td>2-278</td>
</tr>
<tr>
<td>PXFE&lt;TYPE&gt;SET</td>
<td>2-280</td>
</tr>
<tr>
<td>PXFEXECV</td>
<td>2-281</td>
</tr>
<tr>
<td>PXFEXECVE</td>
<td>2-282</td>
</tr>
<tr>
<td>PXFEXECVP</td>
<td>2-283</td>
</tr>
<tr>
<td>PXFEXIT, PXFFASTEXIT</td>
<td>2-284</td>
</tr>
<tr>
<td>PXFFDOPEN</td>
<td>2-285</td>
</tr>
<tr>
<td>PXFFFLUSH</td>
<td>2-286</td>
</tr>
<tr>
<td>PXFFGETC</td>
<td>2-286</td>
</tr>
<tr>
<td>PXFFILENO</td>
<td>2-287</td>
</tr>
<tr>
<td>PXFFORK</td>
<td>2-288</td>
</tr>
<tr>
<td>PXFFPATHCONF</td>
<td>2-289</td>
</tr>
<tr>
<td>PXFFPUTC</td>
<td>2-291</td>
</tr>
<tr>
<td>PXFFSEEK</td>
<td>2-291</td>
</tr>
<tr>
<td>PXFFSTAT</td>
<td>2-292</td>
</tr>
<tr>
<td>PXFFTELL</td>
<td>2-293</td>
</tr>
<tr>
<td>PXFGETARG</td>
<td>2-294</td>
</tr>
<tr>
<td>PXFGETATTY</td>
<td>2-294</td>
</tr>
<tr>
<td>PXFGETC</td>
<td>2-295</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>PXFGETCWD</td>
<td>2-295</td>
</tr>
<tr>
<td>PXFGETEGID</td>
<td>2-296</td>
</tr>
<tr>
<td>PXFGETENV</td>
<td>2-296</td>
</tr>
<tr>
<td>PXFGETEUID</td>
<td>2-297</td>
</tr>
<tr>
<td>PXFGETGID</td>
<td>2-297</td>
</tr>
<tr>
<td>PXFGETGRGID</td>
<td>2-298</td>
</tr>
<tr>
<td>PXFGETGRNAM</td>
<td>2-299</td>
</tr>
<tr>
<td>PXFGETGROUPS</td>
<td>2-299</td>
</tr>
<tr>
<td>PXFGETLOGIN</td>
<td>2-302</td>
</tr>
<tr>
<td>PXFGETPGRP</td>
<td>2-302</td>
</tr>
<tr>
<td>PXFGETPID</td>
<td>2-302</td>
</tr>
<tr>
<td>PXFGETPPID</td>
<td>2-303</td>
</tr>
<tr>
<td>PXFGETPWNAME</td>
<td>2-303</td>
</tr>
<tr>
<td>PXFGETPWUID</td>
<td>2-304</td>
</tr>
<tr>
<td>PXFGETSUBHANDLE</td>
<td>2-305</td>
</tr>
<tr>
<td>PXFGETUID</td>
<td>2-306</td>
</tr>
<tr>
<td>PXFISBLK</td>
<td>2-306</td>
</tr>
<tr>
<td>PXFISCHR</td>
<td>2-306</td>
</tr>
<tr>
<td>PXFISCONST</td>
<td>2-307</td>
</tr>
<tr>
<td>PXFISDIR</td>
<td>2-307</td>
</tr>
<tr>
<td>PXFISFIFO</td>
<td>2-308</td>
</tr>
<tr>
<td>PXFISREG</td>
<td>2-308</td>
</tr>
<tr>
<td>PXFKILL</td>
<td>2-309</td>
</tr>
<tr>
<td>PXFLINK</td>
<td>2-309</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>2-310</td>
</tr>
<tr>
<td>PXFLSEEK</td>
<td>2-311</td>
</tr>
<tr>
<td>PXFMKDIR</td>
<td>2-312</td>
</tr>
<tr>
<td>PXFMKFIFO</td>
<td>2-313</td>
</tr>
<tr>
<td>PXFOPEN</td>
<td>2-314</td>
</tr>
<tr>
<td>PXFOPENDIR</td>
<td>2-316</td>
</tr>
<tr>
<td>PXFPATHCONF</td>
<td>2-317</td>
</tr>
<tr>
<td>PXFPAUSE</td>
<td>2-318</td>
</tr>
<tr>
<td>PXFPIPE</td>
<td>2-319</td>
</tr>
<tr>
<td>PXFPOSIXIO</td>
<td>2-319</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
</tr>
<tr>
<td>PXFPUTC</td>
<td>2-320</td>
</tr>
<tr>
<td>PXFREAD</td>
<td>2-321</td>
</tr>
<tr>
<td>PXFREaddir</td>
<td>2-321</td>
</tr>
<tr>
<td>PXFRENAME</td>
<td>2-322</td>
</tr>
<tr>
<td>PXFREWINDIR</td>
<td>2-323</td>
</tr>
<tr>
<td>PXFRmdir</td>
<td>2-323</td>
</tr>
<tr>
<td>PXFSETENV</td>
<td>2-324</td>
</tr>
<tr>
<td>PXFSETGID</td>
<td>2-325</td>
</tr>
<tr>
<td>PXFSETPGID</td>
<td>2-326</td>
</tr>
<tr>
<td>PXFSETsid</td>
<td>2-327</td>
</tr>
<tr>
<td>PXFSETUID</td>
<td>2-327</td>
</tr>
<tr>
<td>PXFSIGACTION</td>
<td>2-328</td>
</tr>
<tr>
<td>PXFSIGADDSET</td>
<td>2-329</td>
</tr>
<tr>
<td>PXFSIGDELSET</td>
<td>2-330</td>
</tr>
<tr>
<td>PXFSIGEMPTYSET</td>
<td>2-330</td>
</tr>
<tr>
<td>PXFSIGFILLSET</td>
<td>2-331</td>
</tr>
<tr>
<td>PXFSIGISMEMBER</td>
<td>2-332</td>
</tr>
<tr>
<td>PXFSIGPENDING</td>
<td>2-333</td>
</tr>
<tr>
<td>PXFSIGPROCMASK</td>
<td>2-333</td>
</tr>
<tr>
<td>PXFSIGSUSPEND</td>
<td>2-334</td>
</tr>
<tr>
<td>PXFSLEEP</td>
<td>2-334</td>
</tr>
<tr>
<td>PXFSTAT</td>
<td>2-335</td>
</tr>
<tr>
<td>PXFSTRUCTCOPY</td>
<td>2-336</td>
</tr>
<tr>
<td>PXFSTRUCTCREATE</td>
<td>2-336</td>
</tr>
<tr>
<td>PXFSTRUCTFREE</td>
<td>2-341</td>
</tr>
<tr>
<td>PXFSYSCONF</td>
<td>2-341</td>
</tr>
<tr>
<td>PXFTCDRAIN</td>
<td>2-343</td>
</tr>
<tr>
<td>PXFTCFLOW</td>
<td>2-344</td>
</tr>
<tr>
<td>PXFTCFLUSH</td>
<td>2-345</td>
</tr>
<tr>
<td>PXFTCGETATTR</td>
<td>2-345</td>
</tr>
<tr>
<td>PXFTCGETPGRP</td>
<td>2-346</td>
</tr>
<tr>
<td>PXFTCSENDBREAK</td>
<td>2-346</td>
</tr>
<tr>
<td>PXFTCSEATTR</td>
<td>2-347</td>
</tr>
<tr>
<td>PXFTCSETPGRP</td>
<td>2-348</td>
</tr>
</tbody>
</table>
Intel Fortran Libraries Reference

SETVIEWPORT ................................................................. 2-428
SETWINDOW ................................................................. 2-429
SETWINDOWCONFIG ................................................... 2-430
SETWINDOWMENUQQ .................................................. 2-434
SETWRITEMODE ......................................................... 2-435
SETWSIZEQQ ................................................................. 2-437
SHORT ................................................................. 2-439
SIGNALQQ ................................................................. 2-439
SIGNAL ................................................................. 2-442
SLEEPQQ ................................................................. 2-444
SLEEP ................................................................. 2-444
SORTQQ ................................................................. 2-445
SPLITPATHQQ ............................................................... 2-447
SPORT_CANCEL_IO .................................................. 2-448
SPORT_CONNECT ....................................................... 2-449
SPORT_CONNECT_EX .................................................. 2-450
SPORT_GET_HANDLE .................................................. 2-452
SPORT_GET_STATE ..................................................... 2-453
SPORT_GET_STATE_EX .................................................. 2-454
SPORT_GET_TIMEOUTS ................................................. 2-456
SPORT_PEEK_DATA ..................................................... 2-457
SPORT_PEEK_LINE ...................................................... 2-458
SPORT_PURGE ............................................................ 2-459
SPORT_READ_DATA ..................................................... 2-460
SPORT_READ_LINE ...................................................... 2-461
SPORT_RELEASE .......................................................... 2-462
SPORT_SET_STATE ...................................................... 2-463
SPORT_SET_STATE_EX .................................................. 2-464
SPORT_SET_TIMEOUTS .................................................. 2-467
SPORT_SHOW_STATE ..................................................... 2-468
SPORT_SPECIAL_FUNC .................................................. 2-469
SPORT_WRITE_DATA ..................................................... 2-470
SPORT_WRITE_LINE ..................................................... 2-471
SRAND ................................................................. 2-472
## Contents

SSWRQQ ................................................................. 2-473  
STAT ............................................................... 2-474  
SYSTEM ........................................................... 2-476  
SYSTEMQQ ....................................................... 2-478  
TIME ............................................................... 2-479  
TIMEF .............................................................. 2-480  
TRACEBACKQQ .................................................. 2-480  
TTYNAM ............................................................ 2-483  
UNLINK ............................................................. 2-483  
UNPACKTIMEQQ ................................................ 2-484  
UNREGISTERMOUSEEVENT ............................. 2-485  
WAITONMOUSEEVENT ..................................... 2-486  
WRAPON ........................................................... 2-488

## Index
About This Manual

This manual describes the Intel® Fortran library routines. The routines and their descriptions apply to all platforms unless otherwise noted. Architectural differences, if any, are also noted.

For details on managing and linking libraries with the Intel Fortran compiler, see Building Applications.

This manual is intended for experienced applications programmers who have a basic understanding of Fortran concepts and the Fortran 95/90 language, and are using Intel Fortran in either a single-platform or multiplatform environment.

Some familiarity with programming concepts and your operating system is helpful. This manual is not a Fortran or programming tutorial.

This manual is organized as follows:

- Chapter 1, “Overview of the Libraries”
- Chapter 2, “Descriptions of the Library Routines”

Product Website and Support

For the latest product information, visit the Intel web site:

http://developer.intel.com/software/products/

At this site, you will find comprehensive product information, including:

- Links to each product, where you will find technical information such as white papers and articles
- Links to user forums
- Links to news and events

To find technical support information, to register your product, or to contact Intel, please visit:

http://www.intel.com/software/products/support
Related Publications

Tutorial information on Fortran 95 and Fortran 90

The following commercially published documents provide reference or tutorial information on Fortran 95 and Fortran 90:


Intel® does not endorse these books or recommend them over other books on the same subjects.

Standard and Specification Documents

The following copyrighted standard and specification documents provide descriptions of many of the features found in Intel® Fortran:

- American National Standard Programming Language FORTRAN, ANSI X3.9-1978
  This Standard is equivalent to: International Standards Organization Programming Language Fortran, ISO/IEC 1539:1991 (E).
- American National Standard Programming Language Fortran 95, ANSI X3J3/96-007
  This Standard is equivalent to: International Standards Organization Programming Language Fortran, ISO/IEC 1539-1:1997 (E).
Associated Intel Documents

The following Intel documents provide additional information about the Intel® Fortran Compiler, Intel® architecture, Intel® processors, or tools:

- Intel® Fortran Language Reference
- Intel® Fortran Libraries Reference
- Intel® Visual Fortran Compiler for Windows® Systems Installing and Getting Started
- Intel® Array Visualizer online help reference
- Intel® Array Viewer online help reference
- Using the Intel® License Manager for FLEXlm®
- Intel® C++ Compiler User's Guide
- VTune™ Performance Analyzer online help
- Enhanced Debugger online help
- Pentium® Processor Family Developer's Manual
- Intel® Processor Identification with the CPUID Instruction, Intel Corporation, doc. number 241618
- Intel® Itanium® Architecture Manuals
- Intel® Itanium® Architecture Software Conventions & Runtime Architecture Guide
- Intel® Itanium® Assembler User's Guide
- Intel® Itanium® Architecture Assembly Language Reference Guide

Most Intel documents can be found at the Intel web site:
http://developer.intel.com/software/products/

Optimization and Vectorization Terminology and Technology

The following documents provide details on basic optimization and vectorization terminology and technology:


• *An Auto-vectorizing Compiler for the Intel® Architecture*, Aart Bik, Paul Grey, Milind Girkar, and Xinmin Tian. Submitted for publication

• *Efficient Exploitation of Parallelism on Pentium® III and Pentium® 4 Processor-Based Systems*, Aart Bik, Milind Girkar, Paul Grey, and Xinmin Tian.


**Tutorial information on the Intel Fortran Compiler**

For additional training on the Intel Fortran Compiler, choose a course in the Intel® Software College - Course Catalog at [https://shale.intel.com/SoftwareCollege/CourseCatalog.asp](https://shale.intel.com/SoftwareCollege/CourseCatalog.asp).

For additional technical product information including white papers about Intel compilers, open the page associated with your product at [http://developer.intel.com/software/products/](http://developer.intel.com/software/products/).

**Conventions**

The following table describes the typographic and terminology conventions used in this manual:

<table>
<thead>
<tr>
<th>Typographic Conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensions to Fortran 95</td>
<td>This color indicates extensions to the Fortran 95 Standard. These extensions may or may not be implemented by other compilers that conform to the language standard.</td>
</tr>
<tr>
<td>AUTOMATIC, INTRINSIC, WRITE</td>
<td>Uppercase letters indicate Fortran95/90 statements, data types, directives, and other syntax keywords. Examples of statement keywords are WRITE, INTEGER, DO, and OPEN.</td>
</tr>
<tr>
<td>option, option</td>
<td>This italic type indicates an keyword arguments in syntax, new terms, emphasized text, or a book title. Most new terms are defined in the Glossary of the <em>Language Reference</em>.</td>
</tr>
<tr>
<td>USE IFQWIN</td>
<td>This courier type indicates a code example, a program name, a derived type name, or a pathname.</td>
</tr>
</tbody>
</table>
About This Manual

CTRL. Small capital letters indicate the names of keys and key sequences, such as CTRL+C.

A plus indicates a combination of keys. For example, CTRL+E means to hold down the CTRL key while pressing the E key.

(choice1 | choice2) Braces and vertical bars indicate a choice of items. You can usually only choose one of the items in the braces.

[optional item] In syntax, single square brackets indicate items that are optional. In code examples, they are used to show arrays.

s[, s]… A horizontal ellipsis (three dots in a row) following an item indicates that the item preceding the ellipsis can be repeated. In code examples, a horizontal ellipsis means that not all of the statements are shown.

Adobe Acrobat* An asterisk at the end of a word or name indicates it is a third-party product trademark.

Terminology Conventions

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiler option</td>
<td>This term refers to Linux* options and Windows* options that can be used on the compiler command line.</td>
</tr>
<tr>
<td>cat(1)</td>
<td>This format refers to an online reference page; the section number of the page is shown in parentheses. For example, a reference to cat(1) indicates that you can find the material on the cat command in Section 1 of the reference pages. To read online reference pages, use the man command. Your operating system documentation also includes reference page descriptions.</td>
</tr>
<tr>
<td>Intel Fortran</td>
<td>This term refers to the name of the common compiler language supported by the Intel® Visual Fortran Compiler for Windows* and Intel® Fortran Compiler for Linux* products. For more information on these compilers, see <a href="http://developer.intel.com/software/products/">http://developer.intel.com/software/products/</a> .</td>
</tr>
<tr>
<td>Fortran</td>
<td>This term refers to language information that is common to ANSI FORTRAN 77, ANSI/ISO Fortran 95/90, and Intel Fortran.</td>
</tr>
<tr>
<td>Fortran 95/90</td>
<td>This term refers to language information that is common to ANSI/ISO Fortran 95 and ANSI/ISO Fortran 90.</td>
</tr>
<tr>
<td>Fortran 95</td>
<td>This term refers to language features of ANSI/ISO Fortran 95.</td>
</tr>
<tr>
<td>Fortran 90</td>
<td>This term refers to language features of ANSI/ISO Fortran 90.</td>
</tr>
<tr>
<td>Windows systems</td>
<td>This term refers to all supported Microsoft* Windows operating systems. (See also &quot;Platform Labels&quot;.)</td>
</tr>
<tr>
<td>Linux systems</td>
<td>This term refers to all supported Linux operating systems. (See also &quot;Platform Labels&quot;.)</td>
</tr>
<tr>
<td>integer</td>
<td>This term refers to the INTEGER(KIND=1), INTEGER(KIND=2), INTEGER (INTEGER(KIND=4)), and INTEGER(KIND=8) data types as a group.</td>
</tr>
</tbody>
</table>
The following example shows how this manual's typographic conventions are used to indicate the syntax of the CHMOD portability function:

```
result = CHMOD (name, mode)
```

This syntax shows that when you use this routine, you must specify the following:
- The keyword CHMOD.
- A left parenthesis.
- The arguments name and mode, where name is the name of a file and mode is the file permission.
- A terminating right parenthesis.

The syntax is shown in teal color, which indicates the the function is a language extension to Fortran 95.

### Platform Labels

A *platform* is a combination of operating system and central processing unit (CPU) that provides a distinct environment in which to use a product (in this case, a language). This manual contains information for the following language platforms:

<table>
<thead>
<tr>
<th>Language</th>
<th>Platform</th>
<th>Operating System</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Fortran</td>
<td></td>
<td>Linux</td>
<td>IA-32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux</td>
<td>Intel® Itanium®</td>
</tr>
</tbody>
</table>
In this manual, information applies to all supported platforms unless it is otherwise labeled for a specific platform, as follows:

<table>
<thead>
<tr>
<th>Language</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operating System</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows® 2000</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows NT® 4.0</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows XP®</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows XP</td>
</tr>
</tbody>
</table>

L*X Applies to Linux® on Intel® IA-32 processors and Intel® Itanium® processors.
L*X32 Applies to Linux on Intel IA-32 processors.
L*X64 Applies to Linux on Intel Itanium processors.
W*64 Applies to Microsoft Windows XP operating systems on Intel Itanium processors.
i32 Applies to 32-bit operating systems on Intel IA-32 processors.
i64 Applies to 64-bit operating systems on Intel Itanium processors.

For example, the IOFOCUS specifier (for an OPEN statement) is labeled "(W*32, W*64)", so this specifier is valid only on Windows operating systems.
Overview of the Libraries

This chapter provides an overview of the various Intel® Fortran library routines:

- “Portability Routines”
- “National Language Support Routines (W*32, W*64)”
- “POSIX* Routines”
- “QuickWin and Graphics Routines (W*32, W*64)”
- “Dialog Routines (W*32)”
- “COM and AUTO Routines (W*32)”
- “Miscellaneous Run-Time Routines”

When you include the statement USE module-name in your program, these library routines are automatically linked to your program if called.

You can restrict what is accessed from a USE module by adding ONLY clauses to the USE statement. For more information on the USE statement, see the Language Reference.

All the library routines are language extensions to Fortran 95.

In Chapter 2 of this book, all the library routines are listed alphabetically and described in detail.

---

NOTE. Intrinsic procedures are described in the Language Reference.

---

See Also

- The USE statement in the Language Reference
- The section on portability routines in Building Applications
Portability Routines

The portability routines help you port your programs to or from other systems, or help you perform basic I/O to serial ports on Windows® systems.

To use these routines, add the following statement to the program unit containing the routine:

USE IFPORT

Table 1-1 summarizes portability routines.

Table 1-1 Summary of Portability Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Information Retrieval:</strong></td>
<td></td>
</tr>
<tr>
<td>FSTAT</td>
<td>Returns information about a logical file unit.</td>
</tr>
<tr>
<td>GETENV</td>
<td>Searches the environment for a given string and returns its value if found.</td>
</tr>
<tr>
<td>GETGID</td>
<td>Returns the group ID of the user.</td>
</tr>
<tr>
<td>GETLOG</td>
<td>Returns the user's login name.</td>
</tr>
<tr>
<td>GETPID</td>
<td>Returns the process ID of the process.</td>
</tr>
<tr>
<td>GETUID</td>
<td>Returns the user ID of the user of the process.</td>
</tr>
<tr>
<td>HOSTNAM1</td>
<td>Returns the name of the user's host.</td>
</tr>
<tr>
<td>ISATTY</td>
<td>Checks whether a logical unit number is a terminal.</td>
</tr>
<tr>
<td>RENAME</td>
<td>Renames a file.</td>
</tr>
<tr>
<td>STAT, LSTAT</td>
<td>Returns information about a named file.</td>
</tr>
<tr>
<td>UNLINK</td>
<td>Deletes the file given by path.</td>
</tr>
<tr>
<td><strong>Process Control:</strong></td>
<td></td>
</tr>
<tr>
<td>ABORT</td>
<td>Stops execution of the current process, clears I/O buffers, and writes a string to external unit 0.</td>
</tr>
<tr>
<td>ALARM</td>
<td>Executes an external subroutine after waiting a specified number of seconds.</td>
</tr>
<tr>
<td>KILL</td>
<td>Sends a signal code to the process given by ID.</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>Changes the action for signal.</td>
</tr>
<tr>
<td>SLEEP</td>
<td>Suspends program execution for a specified number of seconds.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Executes a command in a separate shell.</td>
</tr>
<tr>
<td><strong>Numeric Values and Conversion:</strong></td>
<td></td>
</tr>
<tr>
<td>BESJ0, BESJ1, BESJN, BESY0, BESY1, BESYN</td>
<td>Return single-precision values of Bessel functions of the first and second kind of orders 1, 2, and n, respectively.</td>
</tr>
<tr>
<td>BIC, BIS, BIT</td>
<td>Perform bit level clear, set, and test for integers.</td>
</tr>
</tbody>
</table>
### Table 1-1 Summary of Portability Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFLOAT</td>
<td>Converts a COMPLEX(4) argument to DOUBLE PRECISION type.</td>
</tr>
<tr>
<td>COMPLINT, COMPLREAL, COMPLLOG</td>
<td>Return a BIT-WISE complement or logical .NOT. of the argument.</td>
</tr>
<tr>
<td>CSMG</td>
<td>Performs an effective BIT-WISE store under mask.</td>
</tr>
<tr>
<td>DBESJ0, DBESJ1, DBESJN, DBESY0, DBESY1, DBESYN</td>
<td>Return double-precision values of Bessel functions of the first and second kind of orders 1, 2, and n, respectively.</td>
</tr>
<tr>
<td>DFLOATI, DFLOATJ, DFLOATK</td>
<td>Convert an integer to double-precision real type.</td>
</tr>
<tr>
<td>DRAND, DRANDM</td>
<td>Return double-precision random values in the range 0 through 1.0.</td>
</tr>
<tr>
<td>DRANSET</td>
<td>Sets the seed for the random number generator</td>
</tr>
<tr>
<td>IDFLOAT</td>
<td>Converts an INTEGER(4) argument to double-precision real type.</td>
</tr>
<tr>
<td>IFLOATI, IFLOATJ</td>
<td>Convert an integer to single-precision real type.</td>
</tr>
<tr>
<td>INMAX</td>
<td>Returns the maximum positive value for an integer.</td>
</tr>
<tr>
<td>INTC</td>
<td>Converts an INTEGER(4) argument to INTEGER(2) type.</td>
</tr>
<tr>
<td>IRAND, IRANDM</td>
<td>Return a positive integer in the range 0 through 2<strong>31-1 or 2</strong>15-1 if called without an argument.</td>
</tr>
<tr>
<td>IRANGET</td>
<td>Returns the current seed.</td>
</tr>
<tr>
<td>IRANSET</td>
<td>Sets the seed for the random number generator</td>
</tr>
<tr>
<td>JABS</td>
<td>Computes an absolute value.</td>
</tr>
<tr>
<td>LONG</td>
<td>Converts an INTEGER(2) argument to INTEGER(4) type.</td>
</tr>
<tr>
<td>QRANSET</td>
<td>Sets the seed for a sequence of pseudo-random numbers.</td>
</tr>
<tr>
<td>RAND, RANDOM2</td>
<td>Return random values in the range 0 through 1.0.</td>
</tr>
<tr>
<td>RANDF</td>
<td>Generates a random number between 0.0 and RAND_MAX.</td>
</tr>
<tr>
<td>RANGET</td>
<td>Returns the current seed.</td>
</tr>
<tr>
<td>RANSET</td>
<td>Sets the seed for the random number generator</td>
</tr>
<tr>
<td>SEED</td>
<td>Changes the starting point of the random number generator.</td>
</tr>
<tr>
<td>SHORT</td>
<td>Converts an INTEGER(4) argument to INTEGER(2) type.</td>
</tr>
<tr>
<td>SRAND</td>
<td>Seeds the random number generator used with IRAND and RAND.</td>
</tr>
<tr>
<td>ACCESS</td>
<td>Checks a file for accessibility according to mode.</td>
</tr>
<tr>
<td>CHMOD</td>
<td>Changes file attributes.</td>
</tr>
<tr>
<td>FGETC</td>
<td>Reads a character from an external unit.</td>
</tr>
<tr>
<td>FLUSH</td>
<td>Flushes the buffer for an external unit to its associated file.</td>
</tr>
</tbody>
</table>

### Input and Output:

- ACCESS
- CHMOD
- FGETC
- FLUSH
### Intel Fortran Libraries Reference

#### Table 1-1 Summary of Portability Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPUTC</td>
<td>Writes a character to an external unit.</td>
</tr>
<tr>
<td>FSEEK</td>
<td>Repositions a file on an external unit.</td>
</tr>
<tr>
<td>FTELL, FTELL8</td>
<td>Return the offset, in bytes, from the beginning of the file.</td>
</tr>
<tr>
<td>GETC</td>
<td>Reads a character from unit 5.</td>
</tr>
<tr>
<td>GETPOS, GETPOSI8</td>
<td>Return the offset, in bytes, from the beginning of the file.</td>
</tr>
<tr>
<td>PUTC</td>
<td>Writes a character to unit 6.</td>
</tr>
</tbody>
</table>

### Date and Time:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOCK</td>
<td>Returns current time in “hh:mm:ss” format using a 24-hour clock.</td>
</tr>
<tr>
<td>CLOCKX</td>
<td>Returns the processor clock to the nearest microsecond.</td>
</tr>
<tr>
<td>CTIME</td>
<td>Converts a system time to a 24-character ASCII string.</td>
</tr>
<tr>
<td>DATE</td>
<td>Returns the current system date.</td>
</tr>
<tr>
<td>DATE4</td>
<td>Returns the current system date.</td>
</tr>
<tr>
<td>DCLOCK</td>
<td>Returns the elapsed time in seconds since the start of the current process.</td>
</tr>
<tr>
<td>DTIME</td>
<td>Returns CPU time since later of (1) start of program, or (2) most recent call to DTIME.</td>
</tr>
<tr>
<td>ETIME</td>
<td>Returns elapsed CPU time since the start of program execution.</td>
</tr>
<tr>
<td>FDATE</td>
<td>Returns the current date and time as an ASCII string.</td>
</tr>
<tr>
<td>GETDATE</td>
<td>Returns the date.</td>
</tr>
<tr>
<td>GETTIM</td>
<td>Returns the time.</td>
</tr>
<tr>
<td>GMTIME</td>
<td>Returns Greenwich Mean Time as a 9-element integer array.</td>
</tr>
<tr>
<td>IDATE</td>
<td>Returns the date either as one 3-element array or three scalar parameters (month, day, year).</td>
</tr>
<tr>
<td>IDATE4</td>
<td>Returns the date either as one 3-element array or three scalar parameters (month, day, year).</td>
</tr>
<tr>
<td>ITIME</td>
<td>Returns current time as a 3-element array (hour, minute, second).</td>
</tr>
<tr>
<td>JDATE</td>
<td>Returns current date as an 8-character string with the Julian date.</td>
</tr>
<tr>
<td>JDATE4</td>
<td>Returns current date as a 10-character string with the Julian date.</td>
</tr>
<tr>
<td>LTIME</td>
<td>Returns local time as a 9-element integer array.</td>
</tr>
<tr>
<td>RTC</td>
<td>Returns number of seconds since 00:00:00 GMT, Jan 1, 1970.</td>
</tr>
<tr>
<td>SECNDS</td>
<td>Returns number of seconds since midnight, less the value of its argument.</td>
</tr>
<tr>
<td>SETDAT</td>
<td>Sets the date.</td>
</tr>
<tr>
<td>SETTIM</td>
<td>Sets the time.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIME</td>
<td>As a subroutine, returns time formatted as HH:MM:SS; as a function, returns time in seconds since 00:00:00 GMT, Jan 1, 1970.</td>
</tr>
<tr>
<td>TIMEF</td>
<td>Returns the number of seconds since the first time this function was called (or zero).</td>
</tr>
</tbody>
</table>

**Error Handling:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETLASTERROR</td>
<td>Returns the last error set.</td>
</tr>
<tr>
<td>GETLASTERRORQQ</td>
<td>Returns the last error set by a run-time function or subroutine.</td>
</tr>
<tr>
<td>IERRNO</td>
<td>Returns the last code error.</td>
</tr>
<tr>
<td>SETERRORMODEQQ</td>
<td>Sets the mode for handling critical errors.</td>
</tr>
</tbody>
</table>

**Program Call and Control:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAISEQQ</td>
<td>Sends an interrupt to the executing program, simulating an interrupt from the operating system.</td>
</tr>
<tr>
<td>RUNQQ</td>
<td>Calls another program and waits for it to execute.</td>
</tr>
<tr>
<td>SIGNALQQ</td>
<td>Controls signal handling.</td>
</tr>
<tr>
<td>SLEEPQQ</td>
<td>Delays execution of the program for a specified time.</td>
</tr>
</tbody>
</table>

**System, Drive, or Directory Control and Inquiry:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHDIR</td>
<td>Changes the current working directory.</td>
</tr>
<tr>
<td>CHANGEDIRQQ</td>
<td>Makes the specified directory the current (default) directory.</td>
</tr>
<tr>
<td>CHANGEDRIVEQQ</td>
<td>Makes the specified drive the current drive.</td>
</tr>
<tr>
<td>DELDIRQQ</td>
<td>Deletes a specified directory.</td>
</tr>
<tr>
<td>GETDRIVEDIRQQ</td>
<td>Returns the current drive and directory path.</td>
</tr>
<tr>
<td>GETDRIVESIZEQQ</td>
<td>Returns the size of the specified drive.</td>
</tr>
<tr>
<td>GETDRIVESQQ</td>
<td>Returns the drives available to the system.</td>
</tr>
<tr>
<td>GETENVQQ</td>
<td>Returns a value from the current environment.</td>
</tr>
<tr>
<td>MAKEDIRQQ</td>
<td>Creates a directory with the specified directory name.</td>
</tr>
<tr>
<td>SETENVQQ</td>
<td>Adds a new environment variable or sets the value of an existing one.</td>
</tr>
<tr>
<td>SYSTEMQQ</td>
<td>Executes a command by passing a command string to the operating system's command interpreter.</td>
</tr>
</tbody>
</table>

**Speakers:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEEPQQ</td>
<td>Sounds the speaker for a specified duration in milliseconds at a specified frequency in Hertz.</td>
</tr>
</tbody>
</table>
## Table 1-1 Summary of Portability Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Management:</strong></td>
<td></td>
</tr>
<tr>
<td>DELFILESQQ</td>
<td>Deletes the specified files in a specified directory.</td>
</tr>
<tr>
<td>FINDFILEQQ</td>
<td>Searches for a file in the directories specified in the PATH environment variable.</td>
</tr>
<tr>
<td>FULLPATHQQ</td>
<td>Returns the full path for a specified file or directory.</td>
</tr>
<tr>
<td>GETFILEINFOQQ</td>
<td>Returns information about files with names that match a request string.</td>
</tr>
<tr>
<td>PACKTIMEQQ</td>
<td>Packs time values for use by SETFILETIMEQQ.</td>
</tr>
<tr>
<td>RENAMEFILEQQ</td>
<td>Renames a file.</td>
</tr>
<tr>
<td>SETFILEACCESSQQ</td>
<td>Sets file-access mode for the specified file.</td>
</tr>
<tr>
<td>SETFILETIMEQQ</td>
<td>Sets modification time for the specified file.</td>
</tr>
<tr>
<td>SPLITPATHQQ</td>
<td>Breaks a full path into four components.</td>
</tr>
<tr>
<td>UNPACKTIMEQQ</td>
<td>Unpacks a file's packed time and date value into its component parts.</td>
</tr>
<tr>
<td><strong>Arrays:</strong></td>
<td></td>
</tr>
<tr>
<td>BSEARCHQQ</td>
<td>Performs a binary search for a specified element on a sorted one-dimensional array of intrinsic type.</td>
</tr>
<tr>
<td>SORTQQ</td>
<td>Sorts a one-dimensional array of intrinsic type.</td>
</tr>
<tr>
<td><strong>Floating-Point Inquiry and Control:</strong></td>
<td></td>
</tr>
<tr>
<td>CLEARSTATUSFPQQ</td>
<td>Clears the exception flags in the floating-point processor status word.</td>
</tr>
<tr>
<td>GETCONTROLFPQQ</td>
<td>Returns the value of the floating-point processor control word.</td>
</tr>
<tr>
<td>GETSTATUSFPQQ</td>
<td>Returns the value of the floating-point processor status word.</td>
</tr>
<tr>
<td>LCWRQQ</td>
<td>Same as SETCONTROLFPQQ.</td>
</tr>
<tr>
<td>SCWRQQ</td>
<td>Same as GETCONTROLFPQQ.</td>
</tr>
<tr>
<td>SETCONTROLFPQQ</td>
<td>Sets the value of the floating-point processor control word.</td>
</tr>
<tr>
<td>SSWRQQ</td>
<td>Same as GETSTATUSFPQQ.</td>
</tr>
<tr>
<td><em><em>IEEE</em> Functionality:</em>*</td>
<td></td>
</tr>
<tr>
<td>IEEE_FLAGS</td>
<td>Sets, gets, or clears IEEE flags.</td>
</tr>
<tr>
<td>IEEE_HANDLER</td>
<td>Establishes a handler for IEEE exceptions.</td>
</tr>
<tr>
<td><strong>Serial Port I/O:</strong></td>
<td></td>
</tr>
<tr>
<td>SPORT_CANCEL_IO</td>
<td>Cancels any I/O in progress to the specified port.</td>
</tr>
<tr>
<td>SPORT_CONNECT</td>
<td>Establishes the connection to a serial port and defines certain usage parameters.</td>
</tr>
</tbody>
</table>
### Table 1-1 Summary of Portability Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPORT_CONNECT_EX</td>
<td>Establishes the connection to a serial port, defines certain usage parameters, and defines the size of the internal buffer for data reception.</td>
</tr>
<tr>
<td>SPORT_GET_HANDLE</td>
<td>Returns the Windows® handle associated with the communications port.</td>
</tr>
<tr>
<td>SPORT_GET_STATE</td>
<td>Returns the baud rate, parity, data bits setting, and stop bits setting of the communications port.</td>
</tr>
<tr>
<td>SPORT_GET_STATE_EX</td>
<td>Returns the baud rate, parity, data bits setting, stop bits, and other settings of the communications port.</td>
</tr>
<tr>
<td>SPORT_GET_TIMESTOUTS</td>
<td>Returns the user selectable timeouts for the serial port.</td>
</tr>
<tr>
<td>SPORT_PEEK_DATA</td>
<td>Returns information about the availability of input data.</td>
</tr>
<tr>
<td>SPORT_PEEK_LINE</td>
<td>Returns information about the availability of input records.</td>
</tr>
<tr>
<td>SPORT_PURGE</td>
<td>Executes a purge function on the specified port.</td>
</tr>
<tr>
<td>SPORT_READ_DATA</td>
<td>Reads available data from the port specified.</td>
</tr>
<tr>
<td>SPORT_READ_LINE</td>
<td>Reads a record from the port specified.</td>
</tr>
<tr>
<td>SPORT_RELEASE</td>
<td>Releases a serial port that has previously been connected.</td>
</tr>
<tr>
<td>SPORT_SET_STATE</td>
<td>Sets the baud rate, parity, data bits setting, and stop bits setting of the communications port.</td>
</tr>
<tr>
<td>SPORT_SET_STATE_EX</td>
<td>Sets the baud rate, parity, data bits setting, stop bits, and other settings of the communications port.</td>
</tr>
<tr>
<td>SPORT_SET_TIMESTOUTS</td>
<td>Sets the user selectable timeouts for the serial port.</td>
</tr>
<tr>
<td>SPORT_SHOW_STATE</td>
<td>Displays the state of a port.</td>
</tr>
<tr>
<td>SPORT_SPECIAL_FUNC</td>
<td>Executes a communications function on a specified port.</td>
</tr>
<tr>
<td>SPORT_WRITE_DATA</td>
<td>Outputs data to a specified port.</td>
</tr>
<tr>
<td>SPORT_WRITE_LINE</td>
<td>Outputs data to a specified port and follows it with a record terminator.</td>
</tr>
</tbody>
</table>

**Miscellaneous:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNBLNK</td>
<td>Returns the index of the last non-blank character in a string.</td>
</tr>
<tr>
<td>QSORT</td>
<td>Returns a sorted version of a one-dimensional array of a specified number of elements of a named size.</td>
</tr>
<tr>
<td>RINDEX</td>
<td>Returns the index of the last occurrence of a substring in a string.</td>
</tr>
<tr>
<td>SCANENV</td>
<td>Scans the environment for the value of an environment variable.</td>
</tr>
<tr>
<td>TTYNAM</td>
<td>Checks whether a logical unit is a terminal.</td>
</tr>
</tbody>
</table>

1. This routine can also be specified as HOSTNM.
2. There is a RANDOM function and a RANDOM subroutine in the portability library.
3. W*32, W*64
For more information, see the section on portability routines in *Building Applications*.

**National Language Support Routines (W*32, W*64)**

The National Language Support (NLS) routines provide language localization and a multibyte character set (MBCS) to let you write applications in different languages.

To use an NLS routine, add the following statement to the program unit containing the routine:

```fortran
USE IFNLS
```

Table 1-2 summarizes the NLS routines. Routine names are shown in mixed case to make the names easier to understand. When writing your applications, you can use any case.

<table>
<thead>
<tr>
<th>Table 1-2</th>
<th>Summary of NLS Routines (W<em>32, W</em>64)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Locale Setting and Inquiry:</strong></td>
<td></td>
</tr>
<tr>
<td>NLSEnumCodepages</td>
<td>Returns all the supported codepages on the system.</td>
</tr>
<tr>
<td>NLSEnumLocales</td>
<td>Returns all the languages and country combinations supported by the system.</td>
</tr>
<tr>
<td>NLSGetEnvironmentCodepage</td>
<td>Returns the codepage number for the system codepage or the console codepage.</td>
</tr>
<tr>
<td>NLSGetLocale</td>
<td>Returns the current language, country, and codepage.</td>
</tr>
<tr>
<td>NLSGetLocaleInfo</td>
<td>Returns requested information about the current local code set.</td>
</tr>
<tr>
<td>NLSSetEnvironmentCodepage</td>
<td>Changes the codepage for the current console.</td>
</tr>
<tr>
<td>NLSSetLocale</td>
<td>Sets the language, country, and codepage.</td>
</tr>
<tr>
<td><strong>Formatting:</strong></td>
<td></td>
</tr>
<tr>
<td>NLSFormatCurrency</td>
<td>Formats a number string and returns the correct currency string for the current locale.</td>
</tr>
<tr>
<td>NLSFormatDate</td>
<td>Returns a correctly formatted string containing the date for the current locale.</td>
</tr>
<tr>
<td>NLSFormatNumber</td>
<td>Formats a number string and returns the correct number string for the current locale.</td>
</tr>
<tr>
<td>NLSFormatTime</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>
</tr>
<tr>
<td>MBCharLen</td>
<td>Returns the length of the first multibyte character in a string.</td>
</tr>
<tr>
<td>MBCurMax</td>
<td>Returns the longest possible multibyte character for the current codepage.</td>
</tr>
</tbody>
</table>
For more information, see the section on National Language Support routines in *Building Applications.*
NLS Date and Time Format (W*32, W*64)

When NLSGetLocaleInfo (type, outstr) returns information about the date and time formats of the current locale, the value returned in outstr can be interpreted according to the following tables. Any text returned within a date and time string that is enclosed within single quotes should be left in the string in its exact form; that is, do not change the text or the location within the string.

Day

The day can be displayed in one of four formats using the letter "d". The following table shows the four variations:

- **d**: Day of the month as digits without leading zeros for single-digit days
- **dd**: Day of the month as digits with leading zeros for single-digit days
- **ddd**: Day of the week as a three-letter abbreviation (SABBREVDAYNAME)
- **dddd**: Day of the week as its full name (SDAYNAME)

Month

The month can be displayed in one of four formats using the letter "M". The uppercase "M" distinguishes months from minutes. The following table shows the four variations:

- **M**: Month as digits without leading zeros for single-digit months
- **MM**: Month as digits with leading zeros for single-digit months
- **MMM**: Month as a three-letter abbreviation (SABBREVMONTHNAME)
- **MMMM**: Month as its full name (SMONTHNAME)

Year

The year can be displayed in one of three formats using the letter "y". The following table shows the three variations:

- **y**: Year represented by only the last digit
- **yy**: Year represented by only the last two digits
- ** yyyy**: Year represented by the full 4 digits

Period/Era

The period/era string is displayed in a single format using the letters "gg".

- **gg**: Period/Era string
Time
The time can be displayed in one of many formats using the letter "h" or "H" to denote hours, the
letter "m" to denote minutes, the letter "s" to denote seconds and the letter "t" to denote the time
marker. The following table shows the numerous variations of the time format. Lowercase "h"
denotes the 12 hour clock and uppercase "H" denotes the 24 hour clock. The lowercase "m"
distinguishes minutes from months.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Hours without leading zeros for single-digit hours (12 hour clock)</td>
</tr>
<tr>
<td>hh</td>
<td>Hours with leading zeros for single-digit hours (12 hour clock)</td>
</tr>
<tr>
<td>H</td>
<td>Hours without leading zeros for single-digit hours (24 hour clock)</td>
</tr>
<tr>
<td>HH</td>
<td>Hours with leading zeros for single-digit hours (24 hour clock)</td>
</tr>
<tr>
<td>m</td>
<td>Minutes without leading zeros for single-digit minutes</td>
</tr>
<tr>
<td>mm</td>
<td>Minutes with leading zeros for single-digit minutes</td>
</tr>
<tr>
<td>s</td>
<td>Seconds without leading zeros for single-digit seconds</td>
</tr>
<tr>
<td>ss</td>
<td>Seconds with leading zeros for single-digit seconds</td>
</tr>
<tr>
<td>t</td>
<td>One-character time marker string</td>
</tr>
<tr>
<td>tt</td>
<td>Multicharacter time marker string</td>
</tr>
</tbody>
</table>

See Also: "NLSGetLocaleInfo"

Example

```fortran
USE IFNLS
INTEGER(4) strlen
CHARACTER(40) str
strlen = NLSGetLocaleInfo(NLS$LI_SDAYNAME1, str)
print *, str       ! prints Monday if language is English
strlen = NLSGetLocaleInfo(NLS$LI_SDAYNAME2, str)
print *, str       ! prints Tuesday if language is English
```

POSIX* Routines
The POSIX routines help you write Fortran programs that comply with the POSIX Standard. They implement the IEEE POSIX FORTRAN-77 language bindings.

To use a POSIX routine, add the following statement to the program unit containing the routine:

```fortran
USE IFPOSIX
```
Table 1-3 summarizes the Intel Fortran POSIX library routines.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPXFARGC</td>
<td>Returns the index of the last command-line argument.</td>
</tr>
<tr>
<td>IPXFCONST</td>
<td>Returns the value associated with a constant defined in the C POSIX standard.</td>
</tr>
<tr>
<td>IPXFLENTRIM</td>
<td>Returns the index of the last non-blank character in an input string.</td>
</tr>
<tr>
<td>IPXFEXITSTATUS¹</td>
<td>Returns the exit code of a child process.</td>
</tr>
<tr>
<td>IPXFSTOPSIG¹</td>
<td>Returns the number of the signal that caused a child process to stop.</td>
</tr>
<tr>
<td>IPXFTERMSIG¹</td>
<td>Returns the number of the signal that caused a child process to terminate.</td>
</tr>
<tr>
<td>PXF&lt;TYPE&gt;GET</td>
<td>Gets the value stored in a component (or field) of a structure.</td>
</tr>
<tr>
<td>PXF&lt;TYPE&gt;SET</td>
<td>Sets the value of a component (or field) of a structure.</td>
</tr>
<tr>
<td>PXFA&lt;TYPE&gt;GET</td>
<td>Gets the array values stored in a component (or field) of a structure.</td>
</tr>
<tr>
<td>PXFA&lt;TYPE&gt;SET</td>
<td>Sets the value of an array component (or field) of a structure.</td>
</tr>
<tr>
<td>PXFAccess</td>
<td>Determines the accessibility of a file.</td>
</tr>
<tr>
<td>PXFAAlarm</td>
<td>Schedules an alarm.</td>
</tr>
<tr>
<td>PXFcallsubhandle</td>
<td>Calls the associated subroutine.</td>
</tr>
<tr>
<td>PXFCFGETISPEED¹</td>
<td>Returns the input baud rate from a termios structure.</td>
</tr>
<tr>
<td>PXFCFGETOSPEED¹</td>
<td>Returns the output baud rate from a termios structure.</td>
</tr>
<tr>
<td>PXFCFSETISPEED¹</td>
<td>Sets the input baud rate in a termios structure.</td>
</tr>
<tr>
<td>PXFCFSETOSPEED¹</td>
<td>Sets the output baud rate in a termios structure.</td>
</tr>
<tr>
<td>PXFChdir</td>
<td>Changes the current working directory.</td>
</tr>
<tr>
<td>PXFChmod</td>
<td>Changes the ownership mode of the file.</td>
</tr>
<tr>
<td>PXFChown¹</td>
<td>Changes the owner and group of a file.</td>
</tr>
<tr>
<td>PXFClearenv</td>
<td>Clears the process environment.</td>
</tr>
<tr>
<td>PXFClose</td>
<td>Closes the file associated with the descriptor.</td>
</tr>
<tr>
<td>PXFClosedir</td>
<td>Closes the directory stream.</td>
</tr>
<tr>
<td>PXFConst</td>
<td>Returns the value associated with a constant.</td>
</tr>
<tr>
<td>PXFCntl¹</td>
<td>Manipulates an open file descriptor.</td>
</tr>
<tr>
<td>PXFCreat</td>
<td>Creates a new file or rewrites an existing file.</td>
</tr>
<tr>
<td>PXFCtermid¹</td>
<td>Generates a terminal pathname.</td>
</tr>
<tr>
<td>PXFDup, PXFDup2</td>
<td>Duplicates an existing file descriptor.</td>
</tr>
<tr>
<td>PXFE&lt;TYPE&gt;GET</td>
<td>Gets the value stored in an array element component (or field) of a structure.</td>
</tr>
<tr>
<td>PXFE&lt;TYPE&gt;SET</td>
<td>Sets the value of an array element component (or field) of a structure.</td>
</tr>
</tbody>
</table>
Table 1-3 Summary of POSIX Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXFEXECV, PXFEXECVE, PXFEXECVP</td>
<td>Execute a new process by passing command-line arguments.</td>
</tr>
<tr>
<td>PXFEXIT, PXFFASTEXIT</td>
<td>Exits from a process.</td>
</tr>
<tr>
<td>PXFFDOPEN</td>
<td>Opens an external unit.</td>
</tr>
<tr>
<td>PXFFFLUSH</td>
<td>Flushes a file directly to disk.</td>
</tr>
<tr>
<td>PXFFGETC</td>
<td>Reads a character from a file.</td>
</tr>
<tr>
<td>PXFFILENO</td>
<td>Returns the file descriptor associated with a specified unit.</td>
</tr>
<tr>
<td>PXFFORK¹</td>
<td>Creates a child process that differs from the parent process only in its PID.</td>
</tr>
<tr>
<td>PXFFPATHCONF</td>
<td>Gets the value for a configuration option of an opened file.</td>
</tr>
<tr>
<td>PXFFPUTC</td>
<td>Writes a character to a file.</td>
</tr>
<tr>
<td>PXFFSEEK</td>
<td>Modifies a file position.</td>
</tr>
<tr>
<td>PXFFSTAT</td>
<td>Gets a file's status information.</td>
</tr>
<tr>
<td>PXFTELL</td>
<td>Returns the relative position in bytes from the beginning of the file.</td>
</tr>
<tr>
<td>PXFGETARG</td>
<td>Tests whether a file descriptor is connected to a terminal.</td>
</tr>
<tr>
<td>PXFGETC</td>
<td>Reads a character from standard input unit 5.</td>
</tr>
<tr>
<td>PXFGETCWD</td>
<td>Returns the path of the current working directory.</td>
</tr>
<tr>
<td>PXFGETEgid¹</td>
<td>Gets the effective group ID of the current process.</td>
</tr>
<tr>
<td>PXFGETENV</td>
<td>Gets the setting of an environment variable.</td>
</tr>
<tr>
<td>PXFGETEuid¹</td>
<td>Gets the effective user ID of the current process.</td>
</tr>
<tr>
<td>PXFGETGid¹</td>
<td>Gets the real group ID of the current process.</td>
</tr>
<tr>
<td>PXFGETGrgid¹</td>
<td>Gets group information for the specified GID.</td>
</tr>
<tr>
<td>PXFGETGrnam¹</td>
<td>Gets group information for the named group.</td>
</tr>
<tr>
<td>PXFGETGroups¹</td>
<td>Gets supplementary group IDs.</td>
</tr>
<tr>
<td>PXFGETLOGIN</td>
<td>Gets the name of the user.</td>
</tr>
<tr>
<td>PXFGETPgrp¹</td>
<td>Gets the process group ID of the calling process.</td>
</tr>
<tr>
<td>PXFGETPID</td>
<td>Gets the process ID of the calling process.</td>
</tr>
<tr>
<td>PXFGETPpid</td>
<td>Gets the process ID of the parent of the calling process.</td>
</tr>
<tr>
<td>PXFGETPwnam¹</td>
<td>Gets password information for a specified name.</td>
</tr>
<tr>
<td>PXFGETPuid¹</td>
<td>Gets password information for a specified UID.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>PXFGETSUBHANDLE</td>
<td>Returns a handle for a subroutine.</td>
</tr>
<tr>
<td>PXFGETUID(^1)</td>
<td>Gets the real user ID of the current process.</td>
</tr>
<tr>
<td>PXFISBLK</td>
<td>Tests for a block special file.</td>
</tr>
<tr>
<td>PXFISCHR</td>
<td>Tests for a character file.</td>
</tr>
<tr>
<td>PXFISCONST</td>
<td>Tests whether a string is a valid constant name.</td>
</tr>
<tr>
<td>PXFISDIR</td>
<td>Tests whether a file is a directory.</td>
</tr>
<tr>
<td>PXFISFIFO</td>
<td>Tests whether a file is a special FIFO file.</td>
</tr>
<tr>
<td>PXFISREG</td>
<td>Tests whether a file is a regular file.</td>
</tr>
<tr>
<td>PXFKILL</td>
<td>Sends a signal to a specified process.</td>
</tr>
<tr>
<td>PXFLINK</td>
<td>Creates a link to a file or directory.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Converts a given elapsed time in seconds to local time.</td>
</tr>
<tr>
<td>PXFLOCALEXPORT</td>
<td>xlim</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Positions a file a specified distance in bytes.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Creates a new directory.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Creates a new FIFO.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Opens or creates a file.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Opens a directory and associates a stream with it.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Gets the value for a configuration option of an opened file.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Suspends process execution.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Creates a communications pipe between two processes.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Sets the current value of the POSIX I/O flag.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Outputs a character to logical unit 6 (stdout).</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Reads from a file.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Reads the current directory entry.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Changes the name of a file.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Resets the position of the stream to the beginning of the directory.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Removes a directory.</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>Adds a new environment variable or sets the value of an environment variable.</td>
</tr>
<tr>
<td>PXFSETGID(^1)</td>
<td>Sets the effective group ID of the current process.</td>
</tr>
<tr>
<td>PXFSETPGID(^1)</td>
<td>Sets the process group ID.</td>
</tr>
<tr>
<td>PXFSETSID(^1)</td>
<td>Creates a session and sets the process group ID.</td>
</tr>
<tr>
<td>PXFSETUID(^1)</td>
<td>Sets the effective user ID of the current process.</td>
</tr>
</tbody>
</table>
### Table 1-3 Summary of POSIX Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXFSIGACTION</td>
<td>Changes the action associated with a specific signal.</td>
</tr>
<tr>
<td>PXFSIGADDSET(^1)</td>
<td>Adds a signal to a signal set.</td>
</tr>
<tr>
<td>PXFSIGDELSSET(^1)</td>
<td>Deletes a signal from a signal set.</td>
</tr>
<tr>
<td>PXFSIGEMPTYSET(^1)</td>
<td>Empties a signal set.</td>
</tr>
<tr>
<td>PXFSIGFILLSET(^1)</td>
<td>Fills a signal set.</td>
</tr>
<tr>
<td>PXFSIGISMEMBER(^1)</td>
<td>Tests whether a signal is a member of a signal set.</td>
</tr>
<tr>
<td>PXFSIGPENDING(^1)</td>
<td>Examines pending signals.</td>
</tr>
<tr>
<td>PXFSIGPROCmask(^1)</td>
<td>Changes the list of currently blocked signals.</td>
</tr>
<tr>
<td>PXFSIGSUSPEND(^1)</td>
<td>Suspends the process until a signal is received.</td>
</tr>
<tr>
<td>PXFSLEEP</td>
<td>Forces the process to sleep.</td>
</tr>
<tr>
<td>PXFSSTAT</td>
<td>Gets the status of a file.</td>
</tr>
<tr>
<td>PXFSTRUCTCOPY</td>
<td>Copies the contents of one structure to another.</td>
</tr>
<tr>
<td>PXFSTRUCTCREATE</td>
<td>Creates an instance of the specified structure.</td>
</tr>
<tr>
<td>PXFSTRUCTFREE</td>
<td>Deletes the instance of a structure.</td>
</tr>
<tr>
<td>PXFSYSConf</td>
<td>Gets values for system limits or options.</td>
</tr>
<tr>
<td>PXFTCDRAIN(^1)</td>
<td>Waits until all output written has been transmitted.</td>
</tr>
<tr>
<td>PXFTCFLOW(^1)</td>
<td>Suspends the transmission or reception of data.</td>
</tr>
<tr>
<td>PXFTCFLUSH(^1)</td>
<td>Discards terminal input data, output data, or both.</td>
</tr>
<tr>
<td>PXFTCGETATTR(^1)</td>
<td>Reads current terminal settings.</td>
</tr>
<tr>
<td>PXFTCGETPGRP(^1)</td>
<td>Gets the foreground process group ID associated with the terminal.</td>
</tr>
<tr>
<td>PXFTCSENDBREAK(^1)</td>
<td>Sends a break to the terminal.</td>
</tr>
<tr>
<td>PXFTCSETATTR(^1)</td>
<td>Writes new terminal settings.</td>
</tr>
<tr>
<td>PXFTCSETPGRP(^1)</td>
<td>Sets the foreground process group associated with the terminal.</td>
</tr>
<tr>
<td>PXFTIME</td>
<td>Gets the system time.</td>
</tr>
<tr>
<td>PXFTIMES</td>
<td>Gets process times.</td>
</tr>
<tr>
<td>PXFTTYNAM(^1)</td>
<td>Gets the terminal pathname.</td>
</tr>
<tr>
<td>PXFUCOMPARE</td>
<td>Compares two unsigned integers.</td>
</tr>
<tr>
<td>PXFUMASK</td>
<td>Sets a new file creation mask and gets the previous one.</td>
</tr>
<tr>
<td>PXFUNAME</td>
<td>Gets the operation system name.</td>
</tr>
<tr>
<td>PXFUNLINK</td>
<td>Removes a directory entry.</td>
</tr>
<tr>
<td>PXFUTIME</td>
<td>Sets file access and modification times.</td>
</tr>
</tbody>
</table>
QuickWin and Graphics Routines (W*32, W*64)

QuickWin routines help you turn graphics programs into simple Windows* applications. The graphics routines can be used in Standard Graphics applications and in Quickwin applications. They can also be used in QuickWin applications.

To use a Quickwin or graphics routine, add the following statement to the program unit containing the routine:

```
USE IFQWIN
```

For graphics routines, you must also choose the QuickWin Graphics or Standard Graphics program type.

Table 1-4 summarizes the QuickWin routines.

<table>
<thead>
<tr>
<th>Table 1-4 Summary of QuickWin Routines (W<em>32, W</em>64)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>PXFWAIT$^1$</td>
</tr>
<tr>
<td>PXFWAITPID$^1$</td>
</tr>
<tr>
<td>PXFWIFEXITED$^1$</td>
</tr>
<tr>
<td>PXFWIFSIGNALED$^1$</td>
</tr>
<tr>
<td>PXFWIFSTOPPED$^1$</td>
</tr>
<tr>
<td>PXWRITE</td>
</tr>
</tbody>
</table>

1. L*X only
For more information, see the section on using QuickWin in *Building Applications*.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETWSIZEQQ</td>
<td>Sets the size and position of a window.</td>
</tr>
<tr>
<td>ABOUTBOXQQ</td>
<td>Adds an About Box with customized text.</td>
</tr>
<tr>
<td>APPENDMENUQQ</td>
<td>Appends a menu item.</td>
</tr>
<tr>
<td>CLICKMENUQQ</td>
<td>Simulates the effect of clicking or selecting a menu item.</td>
</tr>
<tr>
<td>DELETEMENUQQ</td>
<td>Deletes a menu item.</td>
</tr>
<tr>
<td>GETEXITQQ</td>
<td>Returns the setting for a QuickWin application's exit behavior.</td>
</tr>
<tr>
<td>INCHARQQ</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>INITIALSETTINGS</td>
<td>Controls initial menu settings and initial frame window.</td>
</tr>
<tr>
<td>INSERTMENUQQ</td>
<td>Inserts a menu item.</td>
</tr>
<tr>
<td>MESSAGEBOXQQ</td>
<td>Displays a message box.</td>
</tr>
<tr>
<td>MODIFYMENUFLAGSQQ</td>
<td>Modifies a menu item's state.</td>
</tr>
<tr>
<td>MODIFYMENUROUTINEQQ</td>
<td>Modifies a menu item's callback routine.</td>
</tr>
<tr>
<td>MODIFYMENUSTRINGQQ</td>
<td>Modifies a menu item's text string.</td>
</tr>
<tr>
<td>PASSDIRKEYSQQ</td>
<td>Determines the behavior of direction and page keys.</td>
</tr>
<tr>
<td>REGISTERMOUSEEVENT</td>
<td>Registers the application defined routines to be called on mouse events.</td>
</tr>
<tr>
<td>SETEXITQQ</td>
<td>Sets a QuickWin application's exit behavior.</td>
</tr>
<tr>
<td>SETMESSAGEQQ</td>
<td>Changes any QuickWin message, including status bar messages, state messages, and dialog box messages.</td>
</tr>
<tr>
<td>SETMOUSECURSOR</td>
<td>Sets the mouse cursor for the window in focus.</td>
</tr>
<tr>
<td>SETWINDOWMENUUQ</td>
<td>Sets the menu to which a list of current child window names are appended.</td>
</tr>
<tr>
<td>UNREGISTERMOUSEEVENT</td>
<td>Removes the routine registered by REGISTERMOUSEEVENT.</td>
</tr>
<tr>
<td>WAITONMOUSEEVENT</td>
<td>Blocks a return until a mouse event occurs.</td>
</tr>
<tr>
<td>INTEGERTORGB</td>
<td>Converts an RGB color value to its red, green, and blue components.</td>
</tr>
<tr>
<td>RGBTOINTEGER</td>
<td>Converts integers specifying red, green, and blue color into an RGB integer (for use in RGB routines).</td>
</tr>
</tbody>
</table>
Table 1-5 summarizes the graphics routines.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOODFILL</td>
<td>Fills an area using the current index and fill mask; fill starting point uses viewport coordinates.</td>
</tr>
<tr>
<td>FLOODFILL_W</td>
<td>Fills an area using the current index and fill mask; fill starting point uses window coordinates.</td>
</tr>
<tr>
<td>FLOODFILLRGB</td>
<td>Fills an area using the current RGB color and fill mask; fill starting point uses viewport coordinates.</td>
</tr>
<tr>
<td>FLOODFILLRGB_W</td>
<td>Fills an area using the current RGB color and fill mask; fill starting point uses viewport coordinates.</td>
</tr>
<tr>
<td>GETBKCOLOR</td>
<td>Returns current background color index for both text and graphics.</td>
</tr>
<tr>
<td>GETBKCOLORRGB</td>
<td>Returns current background RGB color value for both text and graphics.</td>
</tr>
<tr>
<td>GETCOLOR</td>
<td>Returns the current graphics color index.</td>
</tr>
<tr>
<td>GETCOLORRGB</td>
<td>Returns the current graphics color RGB value.</td>
</tr>
<tr>
<td>GETPIXEL</td>
<td>Returns the color index of a pixel; pixel is located using viewport coordinates.</td>
</tr>
<tr>
<td>GETPIXEL_W</td>
<td>Returns the color index of a pixel; pixel is located using window coordinates.</td>
</tr>
<tr>
<td>GETPIXELRGB</td>
<td>Returns the RGB color value of a pixel; pixel is located using viewport coordinates.</td>
</tr>
<tr>
<td>GETPIXELRGB_W</td>
<td>Returns the RGB color value of a pixel; pixel is located using window coordinates.</td>
</tr>
<tr>
<td>GETPIXELS</td>
<td>Returns the color indexes of multiple pixels.</td>
</tr>
<tr>
<td>GETPIXELSRGB</td>
<td>Returns the RGB color values of multiple pixels.</td>
</tr>
<tr>
<td>GETTEXTCOLOR</td>
<td>Returns the current text color index.</td>
</tr>
<tr>
<td>GETTEXTCOLORRGB</td>
<td>Returns the RGB color value of the current text.</td>
</tr>
<tr>
<td>REMAPALLPALETTERGB</td>
<td>Remaps an entire palette to an RGB color.</td>
</tr>
<tr>
<td>REMAPPALETTERGB</td>
<td>Remaps one color index to an RGB color.</td>
</tr>
<tr>
<td>SETBKCOLOR</td>
<td>Sets current background color index for both text and graphics.</td>
</tr>
<tr>
<td>SETBKCOLORRGB</td>
<td>Sets current background RGB color value for both text and graphics.</td>
</tr>
<tr>
<td>SETCOLOR</td>
<td>Sets the current graphics color index.</td>
</tr>
<tr>
<td>SETCOLORRGB</td>
<td>Sets the current graphics color to an RGB value.</td>
</tr>
</tbody>
</table>
### Table 1-5 Summary of Graphics Routines (W*32, W*64)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETPIXEL</td>
<td>Sets a pixel to the current graphics color index; pixel is located using viewport coordinates.</td>
</tr>
<tr>
<td>SETPIXEL_W</td>
<td>Sets a pixel to the current graphics color index; pixel is located using window coordinates.</td>
</tr>
<tr>
<td>SETPIXELRGB</td>
<td>Sets a pixel to an RGB color value; pixel is located using viewport coordinates.</td>
</tr>
<tr>
<td>SETPIXELRGB_W</td>
<td>Sets a pixel to an RGB color value; pixel is located using window coordinates.</td>
</tr>
<tr>
<td>SETPIXELS</td>
<td>Sets the color indexes of multiple pixels.</td>
</tr>
<tr>
<td>SETPIXELSRGB</td>
<td>Sets multiple pixels to an RGB color.</td>
</tr>
<tr>
<td>SETTEXTCOLOR</td>
<td>Sets the current text color index.</td>
</tr>
<tr>
<td>SETTEXTCOLORMRGB</td>
<td>Sets the current text color to an RGB value.</td>
</tr>
</tbody>
</table>

**Figure Characteristics:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETFILLMASK</td>
<td>Returns the current fill mask.</td>
</tr>
<tr>
<td>GETLINESTYLE</td>
<td>Returns the current line style.</td>
</tr>
<tr>
<td>GETWRITEMODE</td>
<td>Returns the logical write mode used when drawing lines.</td>
</tr>
<tr>
<td>SETCLIPRGN</td>
<td>Masks part of the screen; it does not change the viewport coordinates.</td>
</tr>
<tr>
<td>SETFILLMASK</td>
<td>Sets the current fill mask.</td>
</tr>
<tr>
<td>SETLINESTYLE</td>
<td>Sets the current line style.</td>
</tr>
<tr>
<td>SETWRITEMODE</td>
<td>Sets the logical write mode used when drawing lines.</td>
</tr>
</tbody>
</table>

**Coordinate Conversion and Settings:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPHYSCOORD</td>
<td>Converts viewpoint coordinates to physical coordinates.</td>
</tr>
<tr>
<td>GETVIEWCOORD</td>
<td>Converts physical coordinates to viewport coordinates.</td>
</tr>
<tr>
<td>GETVIEWCOORD_W</td>
<td>Converts window coordinates to viewport coordinates.</td>
</tr>
<tr>
<td>GETWINDOWCOORD</td>
<td>Converts viewport coordinates to window coordinates.</td>
</tr>
<tr>
<td>SETVIEWORG</td>
<td>Moves the viewport coordinate origin (0, 0) to a specified physical point.</td>
</tr>
<tr>
<td>SETVIEWPORT</td>
<td>Redefines viewport bounds to the specified limits and sets the viewport coordinate origin to the upper-left corner of this region.</td>
</tr>
<tr>
<td>SETWINDOW</td>
<td>Defines a window bound by specified window coordinates.</td>
</tr>
</tbody>
</table>

**Graphics Drawing:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC</td>
<td>Draws an arc using viewport coordinates.</td>
</tr>
<tr>
<td>ARC_W</td>
<td>Draws an arc using window coordinates.</td>
</tr>
<tr>
<td>CLEARSCREEN</td>
<td>Clears the screen, viewport, or text window.</td>
</tr>
</tbody>
</table>
### Table 1-5 Summary of Graphics Routines (W*32, W*64)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELLIPSE</td>
<td>Draws an ellipse or circle using viewport coordinates.</td>
</tr>
<tr>
<td>ELLIPSE_W</td>
<td>Draws an ellipse or circle using window coordinates.</td>
</tr>
<tr>
<td>GETARCINFO</td>
<td>Returns the endpoints of the most recently drawn arc or pie.</td>
</tr>
<tr>
<td>GETCURRENTPOSITION</td>
<td>Returns the viewport coordinates of the current graphics-output position.</td>
</tr>
<tr>
<td>GETCURRENTPOSITION_W</td>
<td>Returns the window coordinates of the current graphics-output position.</td>
</tr>
<tr>
<td>GRSTATUS</td>
<td>Returns the status (success or failure) of the most recently called graphics routine.</td>
</tr>
<tr>
<td>LINETO</td>
<td>Draws a line from the current graphics-output position to a specified point using viewport coordinates.</td>
</tr>
<tr>
<td>LINETO_W</td>
<td>Draws a line from the current graphics-output position to a specified point using window coordinates.</td>
</tr>
<tr>
<td>LINETOAR</td>
<td>Draws a line between points in one array and corresponding points in another array.</td>
</tr>
<tr>
<td>LINETOAREX</td>
<td>Similar to LINETOAR, but also lets you specify color and line style.</td>
</tr>
<tr>
<td>MOVETO</td>
<td>Moves the current graphics-output position to a specified point using viewport coordinates.</td>
</tr>
<tr>
<td>MOVETO_W</td>
<td>Moves the current graphics-output position to a specified point using window coordinates.</td>
</tr>
<tr>
<td>PIE</td>
<td>Draws a pie-slice-shaped figure using viewport coordinates.</td>
</tr>
<tr>
<td>PIE_W</td>
<td>Draws a pie-slice-shaped figure using window coordinates.</td>
</tr>
<tr>
<td>POLYBEZIER</td>
<td>Draws a Bezier curve using viewport coordinates.</td>
</tr>
<tr>
<td>POLYBEZIER_W</td>
<td>Draws a Bezier curve using window coordinates.</td>
</tr>
<tr>
<td>POLYBEZIERTO</td>
<td>Draws a Bezier curve using viewport coordinates.</td>
</tr>
<tr>
<td>POLYBEZIERTO_W</td>
<td>Draws a Bezier curve using window coordinates.</td>
</tr>
<tr>
<td>POLYGON</td>
<td>Draws a polygon using viewport coordinates.</td>
</tr>
<tr>
<td>POLYGON_W</td>
<td>Draws a polygon using window coordinates.</td>
</tr>
<tr>
<td>POLYLINEQQ</td>
<td>Draws a line between successive points in an array.</td>
</tr>
<tr>
<td>RECTANGLE</td>
<td>Draws a rectangle using viewport coordinates.</td>
</tr>
<tr>
<td>RECTANGLE_W</td>
<td>Draws a rectangle using window coordinates.</td>
</tr>
</tbody>
</table>

### Character-Based Text Display:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAYCURSOR</td>
<td>Sets the cursor on or off.</td>
</tr>
<tr>
<td>GETTEXTPOSITION</td>
<td>Returns the current text-output position.</td>
</tr>
</tbody>
</table>
Table 1-5  Summary of Graphics Routines (W*32, W*64)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETTEXTWINDOW</td>
<td>Returns the boundaries of the current text window.</td>
</tr>
<tr>
<td>OUTTEXT</td>
<td>Sends text to the screen at the current position.</td>
</tr>
<tr>
<td>SCROLLTEXTWINDOW</td>
<td>Scrolls the contents of a text window.</td>
</tr>
<tr>
<td>SETTEXTCURSOR</td>
<td>Sets the height and width of the text cursor for the window in focus.</td>
</tr>
<tr>
<td>SETTEXTPOSITION</td>
<td>Sets the current text-output position.</td>
</tr>
<tr>
<td>SETTEXTWINDOW</td>
<td>Sets the boundaries of the current text window.</td>
</tr>
<tr>
<td>WRAPON</td>
<td>Turns line wrapping on or off.</td>
</tr>
</tbody>
</table>

Font-Based Character Display:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETFONTINFO</td>
<td>Returns the current font characteristics.</td>
</tr>
<tr>
<td>GETGTEXTTEXTENT</td>
<td>Returns the width of specified text in the current font.</td>
</tr>
<tr>
<td>GETGTEXTROTATION</td>
<td>Returns the current orientation of the font text output by OUTGTEXT.</td>
</tr>
<tr>
<td>INITIALIZEFONTS</td>
<td>Initializes the font library.</td>
</tr>
<tr>
<td>OUTGTEXT</td>
<td>Sends text in the current font to the screen at the current position.</td>
</tr>
<tr>
<td>SETFONT</td>
<td>Finds one font that matches a specified set of characteristics and makes it the current font used by OUTGTEXT.</td>
</tr>
<tr>
<td>SETGTEXTROTATION</td>
<td>Sets the orientation angle of font text output in degrees.</td>
</tr>
</tbody>
</table>

Image Transfers in Memory:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETIMAGE</td>
<td>Stores a screen image using viewport coordinates.</td>
</tr>
<tr>
<td>GETIMAGE_W</td>
<td>Stores a screen image using window coordinates.</td>
</tr>
<tr>
<td>IMAGESIZE</td>
<td>Returns a viewport-coordinate image size in bytes.</td>
</tr>
<tr>
<td>IMAGESIZE_W</td>
<td>Returns a window-coordinate image size in bytes.</td>
</tr>
<tr>
<td>PUTIMAGE</td>
<td>Retrieves a viewport-coordinate image from memory and displays it.</td>
</tr>
<tr>
<td>PUTIMAGE_W</td>
<td>Retrieves a window-coordinate image from memory and displays it.</td>
</tr>
</tbody>
</table>

Image Loading and Saving:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOADIMAGE</td>
<td>Reads a Windows bitmap file (.BMP) from disk and displays it as specified viewport coordinates.</td>
</tr>
<tr>
<td>LOADIMAGE_W</td>
<td>Reads a Windows bitmap file (.BMP) from disk and displays it as specified window coordinates.</td>
</tr>
<tr>
<td>SAVEIMAGE</td>
<td>Saves an image from a specified part of the screen and saves it as a Windows bitmap file; screen location is specified using viewport coordinates.</td>
</tr>
</tbody>
</table>
For more information, see the sections on using QuickWin and drawing graphics in *Building Applications*.

## Dialog Routines (W*32)

The dialog routines let you add dialog boxes to Windows®, QuickWin, and console applications. To activate a dialog box, add the following statement to the application’s relevant program unit:

```fortran
USE IFLOGM
```

Table 1-6 summarizes the dialog routines.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLGEXIT</td>
<td>Closes an open dialog box.</td>
</tr>
<tr>
<td>DLGFLUSH</td>
<td>Updates the display of a dialog box.</td>
</tr>
<tr>
<td>DLGGET</td>
<td>Returns the value of a control variable.</td>
</tr>
<tr>
<td>DLGGETCHAR</td>
<td>Returns the value of a character control variable.</td>
</tr>
<tr>
<td>DLGGETINT</td>
<td>Returns the value of an integer control variable.</td>
</tr>
<tr>
<td>DLGGETLOG</td>
<td>Returns the value of a logical control variable.</td>
</tr>
<tr>
<td>DLIGINIT</td>
<td>Initializes a dialog box.</td>
</tr>
<tr>
<td>DLIGINITWITHRESOURCEHANDLE</td>
<td>Initializes a dialog box.</td>
</tr>
<tr>
<td>DLGISDLGMESSAGE</td>
<td>Determines whether a message is intended for a modeless dialog box.</td>
</tr>
<tr>
<td>DLGISDLGMESSAGEWITHDLG</td>
<td>Determines whether a message is intended for a specific modeless dialog box.</td>
</tr>
<tr>
<td>DLGMODAL</td>
<td>Displays a dialog box.</td>
</tr>
<tr>
<td>DLGMODALWITHPARENT</td>
<td>Displays a dialog box and indicates the parent window.</td>
</tr>
<tr>
<td>DLGMODELESS</td>
<td>Displays a modeless dialog box.</td>
</tr>
<tr>
<td>DLGSENDCTRLMESSAGE</td>
<td>Sends a message to a dialog box control.</td>
</tr>
</tbody>
</table>

1. RGB is Red-Green-Blue
2. OUTGTEXT allows use of special fonts; OUTTEXT does not
For more information, see the section on using dialogs in *Building Applications*.

### COM and AUTO Routines (W*32)

The COM and Auto routines help you write programs that use Component Object Model (COM) and Automation servers.

To use a COM routine, add the following statement to the program unit containing the routine:

```
USE IFCOM
```

To use an AUTO routine, add the following statement to the program unit containing the routine:

```
USE IFAUTO
```

Some of the routines may also require the statement USE IFWINTY.

*Table 1-7* summarizes the COM routines. Routine names are shown in mixed case to make the names easier to understand. When writing your applications, you can use any case.

### Table 1-6 Summary of Dialog Routines (W*32)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLGSET</td>
<td>Assigns a value to a control variable.</td>
</tr>
<tr>
<td>DLGSETCHAR</td>
<td>Assigns a value to a character control variable.</td>
</tr>
<tr>
<td>DLGSETCTRLEVENTHANDLER</td>
<td>Assigns user-written event handlers to ActiveX* controls in a dialog box.</td>
</tr>
<tr>
<td>DLGSETINT</td>
<td>Assigns a value to an integer control variable.</td>
</tr>
<tr>
<td>DLGSETLOG</td>
<td>Assigns a value to a logical control variable.</td>
</tr>
<tr>
<td>DLGSETRETURN</td>
<td>Sets the return value for DLGMODAL.</td>
</tr>
<tr>
<td>DLGSETSUB</td>
<td>Assigns a defined callback routine to a control.</td>
</tr>
<tr>
<td>DLGSETTITLE</td>
<td>Sets the title of a dialog box.</td>
</tr>
<tr>
<td>DLGUNINIT</td>
<td>Deallocates memory for an initialized dialog box.</td>
</tr>
</tbody>
</table>

For more information, see the section on using dialogs in *Building Applications*.

### Table 1-7 Summary of COM Routines (W*32)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMAddObjectReference</td>
<td>Adds a reference to an object’s interface.</td>
</tr>
<tr>
<td>COMCLSIDFromProgID</td>
<td>Passes a programmatic identifier and returns the corresponding class identifier.</td>
</tr>
<tr>
<td>COMCLSIDFromString</td>
<td>Passes a class identifier string and returns the corresponding class identifier.</td>
</tr>
</tbody>
</table>
Table 1-7 Summary of COM Routines (W*32)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMCreateObjectByGUID</td>
<td>Passes a class identifier, creates an instance of an object, and returns a pointer to the object's interface.</td>
</tr>
<tr>
<td>COMCreateObjectByProgID</td>
<td>Passes a programmatic identifier, creates an instance of an object, and returns a pointer to the object's IDispatch interface.</td>
</tr>
<tr>
<td>COMGetActiveObjectByGUID</td>
<td>Passes a class identifier and returns a pointer to the interface of a currently active object.</td>
</tr>
<tr>
<td>COMGetActiveObjectByProgID</td>
<td>Passes a programmatic identifier and returns a pointer to the IDispatch interface of a currently active object.</td>
</tr>
<tr>
<td>COMGetFileObject</td>
<td>Passes a file name and returns a pointer to the IDispatch interface of an automation object that can manipulate the file.</td>
</tr>
<tr>
<td>COMInitialize</td>
<td>Initializes the COM library.</td>
</tr>
<tr>
<td>COMIsEqualGUID</td>
<td>Determines whether two globally unique identifiers (GUIDs) are the same.</td>
</tr>
<tr>
<td>COMQueryInterface</td>
<td>Passes an interface identifier and returns a pointer to an object's interface.</td>
</tr>
<tr>
<td>COMReleaseObject</td>
<td>Indicates that the program is done with a reference to an object's interface.</td>
</tr>
<tr>
<td>COMStringFromGUID</td>
<td>Passes a globally unique identifier (GUID) and returns a string of printable characters.</td>
</tr>
<tr>
<td>COMUninitialize</td>
<td>Uninitializes the COM library.</td>
</tr>
</tbody>
</table>

Table 1-8 summarizes the AUTO routines. Routine names are shown in mixed case to make the names easier to understand. When writing your applications, you can use any case.

Table 1-8 Summary of AUTO Routines (W*32)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOAddArg</td>
<td>Passes an argument name and value and adds the argument to the argument list data structure.</td>
</tr>
<tr>
<td>AUTOAllocateInvokeArgs</td>
<td>Allocates an argument list data structure that holds the arguments to be passed to AUTOInvoke.</td>
</tr>
<tr>
<td>AUTODeallocateInvokeArgs</td>
<td>Deallocates an argument list data structure.</td>
</tr>
<tr>
<td>AUTOGetExceptInfo</td>
<td>Retrieves the exception information when a method has returned an exception status.</td>
</tr>
<tr>
<td>AUTOGetProperty</td>
<td>Passes the name or identifier of the property and gets the value of the automation object's property.</td>
</tr>
</tbody>
</table>
Overview of the Libraries

1-25

Miscellaneous Run-Time Routines

These routines help you write programs for applications. To use for_rtl_init_ and for_rtl_finish_, you must call them from a main program written in C. To use the other routines, add the following statement to the program unit containing the routine:

USE IFCORE

Table 1-9 summarizes these run-time routines:

Table 1-9 Summary of Miscellaneous Run-Time Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboards and Speakers:</td>
<td></td>
</tr>
<tr>
<td>GETCHARQQ</td>
<td>Returns the next keyboard keystroke.</td>
</tr>
<tr>
<td>GETSTRQQ</td>
<td>Reads a character string from the keyboard using buffered input.</td>
</tr>
<tr>
<td>PEEKCHARQQ</td>
<td>Checks the buffer to see if a keystroke is waiting.</td>
</tr>
<tr>
<td>File Management:</td>
<td></td>
</tr>
<tr>
<td>COMMITQQ</td>
<td>Executes any pending write operations for the file associated with the specified unit to the file's physical device.</td>
</tr>
</tbody>
</table>
For more information on traceback, see the section on using traceback information in *Building Applications.*
This chapter contains the descriptions of Intel® Fortran library routines listed in alphabetical order. They are all language extensions to Fortran 95.

Modules must be included in programs that contain the following routines:

- **Portability routines**
  These routines require a USE IFPORT statement to access the portability library.

- **POSIX* routines**
  These routines require a USE IFPOSIX statement to access the POSIX library.

- **NLS routines**
  These routines require a USE IFNLS statement to access the NLS library. These routines are only available on Windows* systems.

- **QuickWin and graphics routines**
  These routines require a USE IFQWIN statement to access the Visual Fortran library and graphics modules. These routines are only available on Windows* systems.

- **Serial port I/O routines**
  These routines require a USE IFPORT statement to access the portability library. These routines are only available on Windows* systems on IA-32 processors.

- **Dialog routines**
  These routines require a USE IFLOGM statement to access the dialog library. These routines are only available on Windows* systems on IA-32 processors.

- **Component Object Module (COM) server routines**
  These routines require a USE IFCOM statement to access the COM library. These routines are only available on Windows* systems on IA-32 processors.

- **Automation server routines**
  These routines require a USE IFAUTO statement to access the AUTO library. These routines are only available on Windows* systems on IA-32 processors.
• **“Miscellaneous Run-Time Routines”**
  Most of these routines require a USE IFCORE statement to obtain the proper interfaces.
  Required USE statements are prominent in the routine descriptions.
  In addition to the appropriate USE statement, for some routines you must specify the types of
  libraries to be used when linking. For more information, see "Specifying Path, Library, and
  Include Directories" in *Building Applications*.
  Also see "Understanding Errors During the Build Process" in *Building Applications*.
  In the description of routines, pointers and handles are INTEGER(4) on IA-32 processors;
  INTEGER(8) on Intel® Itanium® processors.

**ABORT**

**Portability Subroutine:** Flushes and closes I/O buffers, and terminates program execution.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL ABORT [string]
```

*string*

(Input; optional) Character*(*)*. Allows you to specify an abort message at program termination.
When ABORT is called, "abort:" is written to external unit 0, followed by *string*. If omitted, the
default message written to external unit 0 is "abort: Fortran Abort Called."

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS LIB

**See Also:** the EXIT and STOP statements in the *Language Reference*

**Example**

```fortran
USE IFPORT
! The following prints "abort: Fortran Abort Called"
CALL ABORT
! The following prints "abort: Out of here!"
Call ABORT ('"Out of here!"
```

**ABOUTBOXQQ**

**QuickWin Function:** Specifies the information displayed in the message box that appears when
the user selects the About command from a QuickWin application’s Help menu. This function is
only available on Windows* systems.
Module: USE IFQWIN

Syntax
result = ABOUTBOXQQ (cstring)

cstring
(Input; output) Character*(*) . Null-terminated C string.

Results:
The value of the result is INTEGER(4). It is zero if successful; otherwise, nonzero.
If your program does not call ABOUTBOXQQ, the QuickWin run-time library supplies a default string. For further discussion, see "Using QuickWin" in Building Applications.

Compatibility
QUICKWIN GRAPHICS LIB

Example
Consider the following:
USE IFQWIN
INTEGER(4) dummy
! Set the About box message
dummy = ABOUTBOXQQ ('Matrix Multiplier\r      Version 1.0'C)

ACCESS

Portability Function: Determines if a file exists and how it can be accessed.
Module: USE IFPORT

Syntax
result = ACCESS (name, mode)

name
(Input) Character*(*) . Name of the file whose accessibility is to be determined.

mode
(Input) Character*(*) . Modes of accessibility to check for. Must be a character string of length one or greater containing only the characters "r", "w", "x", or "" (a blank). These characters are interpreted as follows.

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Tests for read permission</td>
</tr>
<tr>
<td>w</td>
<td>Tests for write permission</td>
</tr>
</tbody>
</table>
The characters within \textit{mode} can appear in any order or combination. For example, \texttt{wrx} and \texttt{r} are legal forms of \textit{mode} and represent the same set of inquiries.

\textbf{Results:}

The value of the result is INTEGER(4). It is zero if all inquiries specified by \textit{mode} are true. If either argument is invalid, or if the file cannot be accessed in all of the modes specified, one of the following error codes is returned:

- \texttt{EACCES}: Access denied; the file’s permission setting does not allow the specified access.
- \texttt{EINVAL}: The mode argument is invalid.
- \texttt{ENOENT}: File or path not found.

For a list of error codes, see \texttt{IERRNO}.

The \textit{name} argument can contain either forward or backward slashes for path separators. On Windows\textsuperscript{*} systems, all files are readable. A test for read permission always returns 0.

\textbf{Compatibility}

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

\textbf{See Also:} \texttt{GETFILEINFOQQ}, the \texttt{INQUIRE} statement in the \textit{Language Reference}

\textbf{Example}

\begin{verbatim}
USE IFPORT
! checks for read and write permission on the file "DATAFILE.TXT"
J = ACCESS ("DATAFILE.TXT", "rw")
PRINT *, J
! checks whether "DATAFILE.TXT" is executable. It is not, since
! it does not end in .COM, .EXE, .BAT, or .CMD
J = ACCESS ("DATAFILE.TXT","x")
PRINT *, J
\end{verbatim}

\textbf{ALARM}

\textbf{Portability Function:} Causes a subroutine to begin execution after a specified amount of time has elapsed.

\textbf{Module:} USE IFPORT
Syntax

\[
\text{result} = \text{ALARM} \left( \text{time}, \text{proc} \right)
\]

**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 arguments and must be declared **EXTERNAL**.

**Results:**
The return value is INTEGER(4). It is zero if no alarm is pending. If an alarm is pending (has already been set by a previous call to ALARM), it 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 more information on multithreading, see "Creating Multithread Applications" in *Building Applications*.

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, than any pending alarm is deactivated before it has a chance to run.

**Compatibility**

| CONSOLE | STANDARD | GRAPHICS | QUICKWIN | GRAPHICS | WINDOWS | LIB |

**See Also:** [“RUNQQ”](#)

**Example**

```fortran
USE IFPORT
INTEGER(4) numsec, istat
EXTERNAL subprog
numsec = 4
write *, "subprog will begin in ", numsec, " seconds"
ISTAT = ALARM (numsec, subprog)
```
APPENDMENUQQ

QuickWin Function: Appends a menu item to the end of a menu and registers its callback subroutine. This function is only available on Windows® systems.

Module: USE IFQWIN

Syntax

result = APPENDMENUQQ (menuID, flags, text, routine)

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 (see Results 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*(*)). 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 take a single LOGICAL parameter that 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.
- WINSELECTTEXT – Selects text from the current window.
- WINSELECTGRAPHICS – Selects graphics from the current window.
- WINSELECTALL – Selects the entire contents of the current window.
- WININPUT – Brings to the top the child window requesting input and makes it 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.

Results:
The result type is logical. It is .TRUE. if successful; otherwise, .FALSE.. 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. $MENUDISABLED. Some combinations do not make sense, such as $MENUDISABLED and $MENUNOTDISABLED, 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.

For more information about customizing QuickWin menus, see "Using QuickWin" in Building Applications.
Compatibility

QUICKWIN GRAPHICS LIB

See Also: "INSERTMENUQQ", "DELETEN[]MENUQQ", "MODIFYMEN[]FLAGSQQ", "MODIFYMENUROUTINEQQ", "MODIFYMENUSSTRINGQQ"

Example

USE IFQWIN
LOGICAL(4) result
CHARACTER(25) str
...
! Append two items to the bottom of the first (FILE) menu
str    = '@Add to File Menu'C ! 'A' is a quick-access key
result = APPENDMEN[]UQQ(1, $MENUENABLED, str, WINSTATUS)
str    = 'Menu Item &2b'C ! '2' is a quick-access key
result = APPENDMEN[]UQQ(1, $MENUENABLED, str, WINCASCADE)
! Append an item to the bottom of the second (EDIT) menu
str    = 'Add to Second &Menu'C ! 'M' is a quick-access key
result = APPENDMEN[]UQQ(2, $MENUENABLED, str, WINTILE)

ARC, ARC_W

Graphics Functions: Draw elliptical arcs using the current graphics color. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = ARC (x1, y1, x2, y2, x3, y3, x4, y4)
result = ARC_W (wx1, wy1, wx2, wy2, wx3, wy3, wx4, wy4)

x1, y1
(Input) INTEGER(2). Viewport coordinates for upper-left corner of bounding rectangle.
wx1, 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.
 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.

Results:
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.
The center of the arc is the center of the bounding rectangle defined by the points (x1, y1) and (x2,
y2) for ARC and (wx1, wy1) and (wx2, wy2) for ARC_W.
The arc starts where it intersects an imaginary line extending from the center of the arc through
(x3, y3) for ARC and (wx3, wy3) for ARC_W. 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) for ARC and (wx4, wy4) for ARC_W.
ARC uses the view-coordinate system. ARC_W uses the window-coordinate system. In each case,
the arc is drawn using the current color.

NOTE. The ARC routine described here is a QuickWin graphics routine. If
you are trying to use the Microsoft* Platform SDK version of the Arc routine by
including the IFWIN module, you need to specify the routine name as
MSFWINS$Arc. For more information, see "Special Naming Convention for

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

Example
This program draws the arc shown below.
USE IFQWIN
INTEGER(2) status, x1, y1, x2, y2, x3, y3, x4, y4
x1 = 80; y1 = 50
x2 = 240; y2 = 150
x3 = 120; y3 = 75
x4 = 90; y4 = 180
status = ARC( x1, y1, x2, y2, x3, y3, x4, y4 )
END

AUTOAddArg

**AUTO Subroutine:** Passes an argument name and value and adds the argument to the argument list data structure. This subroutine is only available on Windows® systems on IA-32 processors.

**Modules:** USE IFAUTO, USE IFWINTY

**Syntax**

```
CALL AUTOAddArg (invoke_args, name, value [, intent_arg [, type]])
```

*invoke_args*

The argument list data structure. Must be of type INTEGER(4).

*name*

The argument’s name of type CHARACTER*(*)

*value*

The argument’s value. Must be of type INTEGER(2), INTEGER(4), REAL(4), REAL(8), LOGICAL(2), LOGICAL(4), CHARACTER*(*) , or a single dimension array of one of these types. Can also be of type VARIANT, which is defined in the IFWINTY module.

*intent_arg*

Indicates the intended use of the argument by the called method. Must be one of the following constants defined in the IFAUTO module:
• AUTO_ARG_IN: The argument’s value is read by the called method, but not written. This is the default value if `intent_arg` is not specified.
• AUTO_ARG_OUT: The argument’s value is written by the called method, but not read.
• AUTO_ARG_INOUT: The argument’s value is read and written by the called method.

When the value of `intent_arg` is AUTO_ARG_OUT or AUTO_ARG_INOUT, the variable used in the value parameter should be declared using the VOLATILE attribute. This is because the value of the variable will be changed by the subsequent call to AUTOInvoke. The compiler’s global optimizations need to know that the value can change unexpectedly.

type
The variant type of the argument. Must be one of the following constants defined in the IFWINTY module:

<table>
<thead>
<tr>
<th>VARIANT Type</th>
<th>Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_I2</td>
<td>INTEGER(2)</td>
</tr>
<tr>
<td>VT_I4</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_R4</td>
<td>REAL(4)</td>
</tr>
<tr>
<td>VT_R8</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_CY</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_DATE</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>CHARACTER*(*)</td>
</tr>
<tr>
<td>VT_DISPATCH</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_ERROR</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>LOGICAL(2)</td>
</tr>
<tr>
<td>VT VARIANT</td>
<td>TYPE(VARIANT)</td>
</tr>
<tr>
<td>VT UNKNOWN</td>
<td>INTEGER(4)</td>
</tr>
</tbody>
</table>

See Also: “AUTOInvoke”, the VOLATILE attribute in the Language Reference

Example
See the example in “COMInitialize”.

**AUTOAllocateInvokeArgs**

**AUTO Function:** Allocates an argument list data structure that holds the arguments to be passed to AUTOInvoke. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFAUTO
Syntax

    result = AUTOAllocateInvokeArgs( )

Results:
The value returned is an argument list data structure of type INTEGER(4).

See Also: “AUTOInvoke”

Example
See the example in “COMInitialize”.

AUTODeallocateInvokeArgs

AUTO Subroutine: Deallocates an argument list data structure. This subroutine is only available on Windows* systems on IA-32 processors.
Module: USE IFAUTO
Syntax

    CALL AUTODeallocateInvokeArgs (invoke_args)

invoke_args
The argument list data structure. Must be of type INTEGER(4).

Example
See the example in “COMInitialize”.

AUTOGetExceptInfo

AUTO Subroutine: Retrieves the exception information when a method has returned an exception status. This subroutine is only available on Windows* systems on IA-32 processors.
Module: USE IFAUTO
Syntax

    CALL AUTOGetExceptInfo (invoke_args, code, source, description, h_file, h_context, scode)

invoke_args
The argument list data structure. Must be of type INTEGER(4).

code
An output argument that returns the error code. Must be of type INTEGER(2).
**source**
An output argument that returns a human-readable name of the source of the exception. Must be of type CHARACTER(*)

**description**
An output argument that returns a human-readable description of the error. Must be of type CHARACTER(*)

**h_file**
An output argument that returns the fully qualified path of a Help file with more information about the error. Must be of type CHARACTER(*).

**h_context**
An output argument that returns the Help context of the topic within the Help file. Must be of type INTEGER(4).

**scode**
An output argument that returns an SCODE describing the error. Must be of type INTEGER(4).

**AUTOGetProperty**

**AUTO Function:** Passes the name or identifier of the property and gets the value of the automation object’s property. This function is only available on Windows* systems on IA-32 processors.

**Modules:** USE IFAUTO, USE IFWINTY

**Syntax**

```
result = AUTOGetProperty (idispatch, id, value [, type])
```

**idispatch**
The object’s IDispatch interface pointer. Must be of type INTEGER(4).

**id**
The argument’s name of type CHARACTER(*), or its member ID of type INTEGER(4).

**value**
An output argument that returns the argument’s value. Must be of type INTEGER(2), INTEGER(4), REAL(4), REAL(8), LOGICAL(2), LOGICAL(4), CHARACTER(*), or a single dimension array of one of these types.
type
The variant type of the requested argument. Must be one of the following constants defined in the IFWINTY module:

<table>
<thead>
<tr>
<th>VARIANT Type</th>
<th>Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_I2</td>
<td>INTEGER(2)</td>
</tr>
<tr>
<td>VT_I4</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_R4</td>
<td>REAL(4)</td>
</tr>
<tr>
<td>VT_R8</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_CY</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_DATE</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>CHARACTER*(*)</td>
</tr>
<tr>
<td>VT_DISPATCH</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_ERROR</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>LOGICAL(2)</td>
</tr>
<tr>
<td>VT_UNKNOWN</td>
<td>INTEGER(4)</td>
</tr>
</tbody>
</table>

Results:
Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).

**AUTOGetPropertyByID**

**AUTO Function:** Passes the member ID of the property and gets the value of the automation object's property into the argument list's first argument. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFAUTO

**Syntax**

```
result = AUTOGetPropertyByID (idispatch, memid, invoke_args)
```

idispatch
The object’s IDispatch interface pointer. Must be of type INTEGER(4).

memid
Member ID of the property. Must be of type INTEGER(4).

invoke_args
The argument list data structure. Must be of type INTEGER(4).
Results:
Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).

**AUTOGetPropertyInvokeArgs**

**AUTO Function:** Passes an argument list data structure and gets the value of the automation object’s property specified in the argument list’s first argument. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFAUTO

**Syntax**

```
result = AUTOGetPropertyInvokeArgs (idispatch, invoke_args)
```

- **idispatch**
The object’s IDispatch interface pointer. Must be of type INTEGER(4).

- **invoke_args**
The argument list data structure. Must be of type INTEGER(4).

**Results:**
Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).

**AUTOInvoke**

**AUTO Function:** Passes the name or identifier of an object’s method and an argument list data structure and invokes the method with the passed arguments. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFAUTO

**Syntax**

```
result = AUTOInvoke (idispatch, id, invoke_args)
```

- **idispatch**
The object’s IDispatch interface pointer. Must be of type INTEGER(4).

- **id**
The argument’s name of type CHARACTER*(*) or its member ID of type INTEGER(4).

- **invoke_args**
The argument list data structure. Must be of type INTEGER(4).
Results:
Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).

Example
See the example in “COMInitialize”.

AUTOSetProperty

AUTO Function: Passes the name or identifier of the property and a value, and sets the value of the automation object’s property. This function is only available on Windows® systems on IA-32 processors.

Module: USE IFAUTO, USE IFWINTY

Syntax

\[
\text{result} = \text{AUTOSetProperty} \left( \text{idispatch, id, value }[, \text{type}] \right)
\]

idispatch
The object’s IDispatch interface pointer. Must be of type INTEGER(4).

id
The argument’s name of type CHARACTER*(*), or its member ID of type INTEGER(4).

value
The argument’s value. Must be of type INTEGER(2), INTEGER(4), REAL(4), REAL(8), LOGICAL(2), LOGICAL(4), CHARACTER*(*), or a single dimension array of one of these types.

type
The variant type of the argument. Must be one of the following constants defined in the IFWINTY module:

<table>
<thead>
<tr>
<th>VARIANT Type</th>
<th>Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_I2</td>
<td>INTEGER(2)</td>
</tr>
<tr>
<td>VT_I4</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>VT_R4</td>
<td>REAL(4)</td>
</tr>
<tr>
<td>VT_R8</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_CY</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_DATE</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>CHARACTER*(*)</td>
</tr>
<tr>
<td>VT_DISPATCH</td>
<td>INTEGER(4)</td>
</tr>
</tbody>
</table>
**AUTOSetPropertyByID**

**AUTO Function:** Passes the member ID of the property and sets the value of the automation object’s property into the argument list’s first argument. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFAUTO

**Syntax**

```
result = AUTOSetPropertyByID (idispatch, memid, invoke_args)
```

- `idispatch` The object’s IDispatch interface pointer. Must be of type INTEGER(4).
- `memid` Member ID of the property. Must be of type INTEGER(4).
- `invoke_args` The argument list data structure. Must be of type INTEGER(4).

**Results:**

Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).

---

**AUTOSetPropertyInvokeArgs**

**AUTO Function:** Passes an argument list data structure and sets the value of the automation object’s property specified in the argument list’s first argument. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFAUTO

**Syntax**

```
result = AUTOSetPropertyInvokeArgs (idispatch, invoke_args)
```

**Results:**

Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).
idispatch
The object’s IDispatch interface pointer. Must be of type INTEGER(4).

invoke_args
The argument list data structure. Must be of type INTEGER(4).

Results:
Returns an HRESULT describing the status of the operation. Must be of type INTEGER(4).

BEEPQQ

Portability Subroutine: Sounds the speaker at the specified frequency for the specified duration in milliseconds.

Module: USE IFPORT

Syntax
CALL BEEPQQ (frequency, duration)

frequency
(Input) INTEGER(4). Frequency of the tone in Hz.

duration
(Input) INTEGER(4). Length of the beep in milliseconds.

BEEPQQ does not return until the sound terminates.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “SLEEPQQ”

Example
USE IFPORT
INTEGER(4) frequency, duration
frequency = 4000
duration = 1000
CALL BEEPQQ(frequency, duration)

BESJ0, BESJ1, BESJN, BESY0, BESY1, BESYN

Portability Functions: Compute the single-precision values of Bessel functions of the first and second kinds.

Module: USE IFPORT
Syntax

result = BESJ0 (posvalue)
result = BESJ1 (posvalue)
result = BESJN (n, posvalue)
result = BESY0 (posvalue)
result = BESY1 (posvalue)
result = BESYN (n, posvalue)

posvalue
(Input) REAL(4). Independent variable for a Bessel function. Must be greater than or equal to zero.

n
(Input) INTEGER(4). 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 \( \text{posvalue} \).

BESY0, BESY1, and BESYN return Bessel functions of the second kind, orders 0, 1, and \( n \), respectively, with the independent variable \( \text{posvalue} \).

Negative arguments cause BESY0, BESY1, and BESYN to return QNAN.

Bessel functions are explained more fully in most mathematics reference books, such as the Handbook of Mathematical Functions (Abramowitz and Stegun. Washington: U.S. Government Printing Office, 1964). These functions are commonly used in the mathematics of electromagnetic wave theory.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: "DBESJ0, DBESJ1, DBESJN, DBESY0, DBESY1, DBESYN"

BIC, BIS

Portability Subroutines: Perform a bit-level set and clear for integers.

Module: USE IFPORT

Syntax

CALL BIC (bitnum, target)
CALL BIS (bitnum, target)
bitnum
(Input) INTEGER(4). Bit number to set. Must be in the range 0 (least significant bit) to 31 (most significant bit) if target is INTEGER(4). If target is INTEGER(8), bitnum must be in range 0 to 63.

target
(Input; output) INTEGER(4) or INTEGER(8). Variable whose bit is to be set.
BIC sets bit bitnum of target to 0; BIS sets bit bitnum to 1.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “BIT”

Example
Consider the following:

USE IFPORT
integer(4) bitnum, target_i4
integer(8) target_i8
target_i4 = Z'AAAA'
bitnum = 1
call BIC(bitnum, target_i4)
target_i8 = Z'FFFFFFFF00000000'
bitnum = 40
call BIC(bitnum, target_i8)
bitnum = 0
call BIS(bitnum, target_i4)
bitnum = 1
call BIS(bitnum, target_i8)
print '(" integer*4 result ",Z)', target_i4
print '(" integer*8 result ",Z)', target_i8
end

BIT

Portability Function: Performs a bit-level test for integers.

Module: USE IFPORT

Syntax

result = BIT (bitnum, source)
**bitnum**
(Input) INTEGER(4). Bit number to test. Must be in the range 0 (least significant bit) to 31 (most significant bit).

**source**
(Input) INTEGER(4) or INTEGER(8). Variable being tested.

**Results:**
The result type is logical. It is .TRUE. if bit \textit{bitnum} of \textit{source} is 1; otherwise, .FALSE..

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** “BIC, BIS”

---

**BSEARCHQQ**

**Portability Function:** Performs a binary search of a sorted one-dimensional array for a specified element. The array elements cannot be derived types or structures.

**Module:** USE IFPORT

**Syntax**
```
result = BSEARCHQQ (adrkey, adarray, length, size)
```

**adrkey**
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. Address of the variable containing the element to be found (returned by LOC).

**adarray**
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. Address of the array (returned by LOC).

**length**
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. 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 constants, defined in IFPORT.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>
</tbody>
</table>
If the value provided in \textit{size} is not a symbolic constant and is less than 32,767, the array is assumed to be a character array with \textit{size} characters per element.

\textbf{Results:}

The result type is \texttt{INTEGER(4)}. It is an 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.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Constant} & \textbf{Type of array} \\
\hline
\texttt{SRT$\text{INTEGER2}$} & \texttt{INTEGER(2)} or equivalent \\
\texttt{SRT$\text{INTEGER4}$} & \texttt{INTEGER(4)} or equivalent \\
\texttt{SRT$\text{INTEGER8}$} & \texttt{INTEGER(8)} or equivalent \\
\texttt{SRT$\text{REAL4}$} & \texttt{REAL(4)} or equivalent \\
\texttt{SRT$\text{REAL8}$} & \texttt{REAL(8)} or equivalent \\
\texttt{SRT$\text{REAL16}$} & \texttt{REAL(16)} or equivalent \\
\hline
\end{tabular}
\end{center}

\textbf{CAUTION.} The location of the array and the element to be found must both be passed by address using the \texttt{LOC} function. This defeats Fortran type checking, so you must make certain that the length and size arguments are correct, and that size is the same for the element to be found and the array searched.

If you pass invalid arguments, \texttt{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 a General Protection Violation message.

\textbf{Compatibility}

\texttt{CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB}

\textbf{See Also:} \textit{``SORTQQ'', the \texttt{LOC} intrinsic function in the Language Reference}

\textbf{Example}

USE IFPORT
INTEGER(4) array(10), length
INTEGER(4) result, target
length = SIZE(array)
...
result = BSEARCHQQ(LOC(target),LOC(array),length,SRT$INTEGER4)

CDFLOAT

Portability Function: Converts a COMPLEX(4) argument to double-precision real type.
Module: USE IFPORT
Syntax
    result = CDFLOAT (input)
input
  (Input) COMPLEX(4). The value to be converted.
Results:
  The result type is REAL(8).
Compatibility
  CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

CHANGEDIRQQ

Portability Function: Makes the specified directory the current, default directory.
Module: USE IFPORT
Syntax
    result = CHANGEDIRQQ (dir)
dir
  (Input) Character*(*). Directory to be made the current directory.
Results:
  The result type is LOGICAL(4). It is .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.
Compatibility
  CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
See Also: "GETDRIVEDIRQQ", "MAKEDIRQQ", "DELDIRQQ", "CHANGEDRIVEQQ"
Example
    USE IFPORT
CHANGEDRIVEQQ

Portability Function: Makes the specified drive the current, default drive.

Module: USE IFPORT

Syntax

\[
\text{result} = \text{CHANGEDRIVEQQ}(\text{drive})
\]

\text{drive}

(Input) Character*(*). String beginning with the drive letter.

Results:

The result type is LOGICAL(4). On Windows* systems, the result is .TRUE. if successful; otherwise, .FALSE. On Linux* systems, the result is always .FALSE..

Because drives are identified by a single alphabetic character, CHANGEDRIVEQQ examines only the first character of \text{drive}. 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.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: "GETDRIVESQQ", "GETDRIVESIZEQQ", "GETDRIVEDIRQQ", "CHANGEDIRQQ"

Examples

USE IFPORT
LOGICAL(4) status
status = CHANGEDRIVEQQ('d')

Consider the following:

USE IFPORT
LOGICAL(4) CHANGEDIT
CHANGEDIT = CHANGEDRIVEQQ('d')
IF (CHANGEDIT) THEN
    PRINT *, 'CHANGEDRIVEQQ SUCCESSFUL'
ELSE
    PRINT *, 'Drive could not be changed'
**CHDIR**

**Portability Function:** Changes the default directory.

**Module:** USE IFPORT

**Syntax**

```fortran
result = CHDIR (dir_name)
```

- **dir_name**
  - (Input) Character*. Name of a directory to become the default directory.

**Results:**

The result type is INTEGER(4). It 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 `dir_name` parameter is not a directory.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** “CHANGEDIRQQ”

**Examples**

```fortran
use ifport
integer(4) istatus, enoent, enotdir
character(255) newdir
character(300) prompt, errmsg

prompt = 'Please enter directory name: '
10 write(*,*) TRIM(prompt)
read *, newdir
ISTATUS = CHDIR(newdir)
select case (istatus)
  case (2) ! ENOENT
    errmsg = 'The directory ' // TRIM(newdir) // ' does not exist'
  case (20) ! ENOTDIR
    errmsg = TRIM(newdir) // ' is not a directory'
  case (0) ! NO error
    goto 40
```
case default  
    write (errmsg,*) 'Error with code ', istatus  
end select  
write(*,*) TRIM(errmsg)  
goto 10  
40    write(*,*) 'Default directory successfully changed.'  
end

The following shows another example:
USE IFPORT  
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

Portability Function: Changes the access mode of a file.  
Module: USE IFPORT  
Syntax  
    result = CHMOD (name, mode)  
name  
(Input) Character*(*). Name of the file whose access mode is to be changed. Must have a single path.  
mode  
(Input) Character*(*). 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 following permission bits ORed together:
The following regular expression represents a symbolic mode:

\[[ugoa]*[+-=][rwxXst]*\]

"[ugoa]*" is ignored on Windows* systems. On Linux* systems, a combination of the letters "ugo" control which users’ access to the file will be changed:

- u: The user who owns the file
- g: Other users in the group that owns the file
- o: Other users not in the group that owns the file
- a: All users

"[+-=]" 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. On Windows systems, only "w" is significant and affects write permission; all other letters are ignored. On Linux systems, all letters are significant.
Results:
The result type is INTEGER(4). The result is zero if the mode was changed successfully; otherwise, an error code. Possible error codes are:

- ENOENT: The specified file was not found.
- EINVAL: The mode argument is invalid.
- EPERM: Permission denied; the file’s mode cannot be changed.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “SETFILEACCESSQQ”

Example

USE IFPORT
integer(4) I, Istatus
I = ACCESS ("DATAFILE.TXT", "w")
if (i) then
    ISTATUS = CHMOD ("datafile.txt", "[+w]")
end if
I = ACCESS ("DATAFILE.TXT", "w")
print *, i

CLEARSCREEN

Graphics Subroutine: Erases the target area and fills it with the current background color. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL CLEARSCREEN (area)

area

(Input) INTEGER(4). Identifies the target area. Must be one of the following symbolic constants (defined in IFQWIN.F90):

- $GCLEARSCREEN - Clears the entire screen.
- $GVIEWPORT - Clears only the current viewport.
- $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.
Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB
See Also: “GETBKCOLORRGB”, “SETBKCOLORRGB”, “SETTEXTWINDOW”, “SETVIEWPORT”

Example
USE IFQWIN
CALL CLEARSCREEN ($GCLEARSCREEN)

CLEARSTATUSFPQQ

Portability Subroutine: Clears the exception flags in the floating-point processor status word.
Module: USE IFPORT
Syntax
CALL CLEARSTATUSFPQQ ( )
The floating-point status word indicates which floating-point exception conditions have occurred. Intel® Visual Fortran initially clears (sets to 0) all floating-point status flags, but as exceptions occur, the status flags accumulate until the program clears the flags again. CLEARSTATUSFPQQ will clear the flags.
CLEARSTATUSFPQQ is appropriate for use in applications that poll the floating-point status register as the method for detecting a floating-point exception has occurred.
For a full description of the floating-point status word, exceptions, and error handling, see “The Floating-Point Environment” in Building Applications.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS LIB
See Also: “GETSTATUSFPQQ”, “SETCONTROLFPQQ”, “GETCONTROLFPQQ”, “SIGNALFPQQ”

Example
! Program to demonstrate CLEARSTATUSFPQQ.
! This program uses polling to detect that a floating-point exception has occurred.
! So, build this console application with the default floating-point exception behavior, fpe3.
! PROGRAM CLEARFP
USE IFPORT
REAL*4 A, B, C
INTEGER*2 STS

A = 2.0E0
B = 0.0E0

! Poll and display initial floating point status
CALL GETSTATUSFPQQ(STS)
WRITE(*,'(1X,A,Z4.4)') 'Initial fp status = ',STS

! Cause a divide-by-zero exception
! Poll and display the new floating point status
C = A/B
CALL GETSTATUSFPQQ(STS)
WRITE(*,'(1X,A,Z4.4)') 'After div-by-zero fp status = ',STS

! If a divide by zero error occurred, clear the floating point
! status register so future exceptions can be detected.
IF ((STS .AND. FPSW$ZERODIVIDE) > 0) THEN
  CALL CLEARSTATUSFPQQ()
  CALL GETSTATUSFPQQ(STS)
  WRITE(*,'(1X,A,Z4.4)') 'After CLEARSTATUSFPQQ fp status = ',STS
ENDIF
END

**CLICKMENUQQ**

**QuickWin Function:** Simulates the effect of clicking or selecting a menu command. The QuickWin application responds as though the user had clicked or selected the command. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

result = CLICKMENUQQ (item)

**item**

(Input) INTEGER(4). Constant that represents the command selected from the Window menu. Must be one of the following symbolic constants (defined in IFQWIN.F90):

- QWIN$STATUS - Status command
- QWIN$TILE - Tile command
Descriptions of the Library Routines

- QWIN$CASCADE - Cascade command
- QWIN$ARRANGE - Arrange Icons command

**Results:**
The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero.

**Compatibility**
QUICKWIN GRAPHICS LIB

**See Also:** "REGISTERMOUSEEVENT", "UNREGISTERMOUSEEVENT", "WAITONMOUSEEVENT", "Using QuickWin" in *Building Applications*

---

**CLOCK**

**Portability Function:** Converts a system time into an 8-character ASCII string.

**Module:** USE IFPORT

**Syntax**
```
result = CLOCK ( )
```

**Results:**
The result type is character with a length of 8. The result is the current time in the form hh:mm:ss, using a 24-hour clock.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** the DATE_AND_TIME intrinsic procedure in the *Language Reference*

**Example**
```
USE IFPORT
character(8) whatimeisit
whatimeisit = CLOCK ( )
print *, 'The current time is ',whatimeisit
```

---

**CLOCKX**

**Portability Subroutine:** Returns the processor clock to the nearest microsecond.

**Module:** USE IFPORT

**Syntax**
```
CALL CLOCKX (clock)
```
**clock**

(Input) REAL(8). The current time.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

### COMAddObjectReference

**COM Function:** Adds a reference to an object’s interface. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFCOM

**Syntax**

```fortran
result = COMAddObjectReference (iunknown)
```

*iunknown*

An IUnKnown interface pointer. Must be of type INTEGER(4).

**Results:**

The result type is INTEGER(4). It is the object’s current reference count.

### COMCLSIDFromProgID

**COM Subroutine:** Passes a programmatic identifier and returns the corresponding class identifier. This subroutine is only available on Windows* systems on IA-32 processors.

**Modules:** USE IFCOM, USE IFWINTY

**Syntax**

```fortran
CALL COMCLSIDFromProgID (prog_id, clsid, status)
```

*prog_id*

The programmatic identifier of type CHARACTER*(*).

*clsid*

The class identifier corresponding to the programmatic identifier. Must be of type GUID, which is defined in the IFWINTY module.

*status*

The status of the operation. It can be any status returned by CLSIDFromProgID (see the Microsoft* Platform SDK). Must be of type INTEGER(4).
COMCLSIDFromString

**COM Subroutine:** Passes a class identifier string and returns the corresponding class identifier. This subroutine is only available on Windows* systems on systems on IA-32 processors.

**Modules:** USE IFCOM, USE IFWINTY

**Syntax**

```call comclsidfromstring (string, clsid, status)
```

**string**
The class identifier string of type CHARACTER*(*)

**clsid**
The class identifier corresponding to the identifier string. Must be of type GUID, which is defined in the IFWINTY module.

**status**
The status of the operation. It can be any status returned by CLSIDFromString (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

COMCreateObjectByGUID

**COM Subroutine:** Passes a class identifier, creates an instance of an object, and returns a pointer to the object’s interface. This subroutine is only available on Windows* systems on systems on IA-32 processors.

**Modules:** USE IFCOM, USE IFWINTY

**Syntax**

```call comcreateobjectbyguid (clsid, clsctx, iid, interface, status)
```

**clsid**
The class identifier of the class of object to be created. Must be of type GUID, which is defined in the IFWINTY module.

**clsctx**
Lets you restrict the types of servers used for the object. Must be of type INTEGER(4). Must be one of the CLSCTX_* constants defined in the IFWINTY module.

**iid**
The interface identifier of the interface being requested. Must be of type GUID, which is defined in the IFWINTY module.
**interface**
An output argument that returns the object’s interface pointer. Must be of type INTEGER(4).

**status**
The status of the operation. It can be any status returned by CoCreateInstance (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

### COMCreateObjectByProgID

**COM Subroutine:** Passes a programmatic identifier, creates an instance of an object, and returns a pointer to the object’s IDispatch interface. This subroutine is only available on Windows* systems on IA-32 processors.

**Module:** USE IFCOM

**Syntax**
```
CALL COMCreateObjectByProgID (prog_id, idispatch, status)
```

**prog_id**
The programmatic identifier of type CHARACTER*(*)

**idispatch**
An output argument that returns the object’s IDispatch interface pointer. Must be of type INTEGER(4).

**status**
The status of the operation. It can be any status returned by CLSIDFromProgID or CoCreateInstance (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

### COMGetActiveObjectByGUID

**COM Subroutine:** Passes a class identifier and returns a pointer to the interface of a currently active object. This subroutine is only available on Windows* systems on IA-32 processors.

**Module:** USE IFCOM, USE IFWINTY

**Syntax**
```
CALL COMGetActiveObjectByGUID (clsid, iid, interface, status)
```

**clsid**
The class identifier of the class of object to be found. Must be of type GUID, which is defined in the IFWINTY module.
iid
The interface identifier of the interface being requested. Must be of type GUID, which is defined in the IFWINTY module.

interface
An output argument that returns the object’s interface pointer. Must be of type INTEGER(4).

status
The status of the operation. It can be any status returned by GetActiveObject (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

**COMGetActiveObjectByProgID**

**COM Subroutine:** Passes a programmatic identifier and returns a pointer to the IDispatch interface of a currently active object. This subroutine is only available on Windows* systems on IA-32 processors.

**Module:** USE IFCOM

**Syntax**

CALL COMGetActiveObjectByProgID (prog_id, idispatch, status)

**prog_id**
The programmatic identifier of type CHARACTER*(*).

**idispatch**
An output argument that returns the object’s IDispatch interface pointer. Must be of type INTEGER(4).

**status**
The status of the operation. It can be any status returned by CLSIDFromProgID or GetActiveObject (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

**Example**

See the example in “COMInitialize”.

**COMGetFileObject**

**COM Subroutine:** Passes a file name and returns a pointer to the IDispatch interface of an automation object that can manipulate the file. This subroutine is only available on Windows* systems on IA-32 processors.

**Module:** USE IFCOM
Syntax

CALL COMGetFileObject (filename, idispatch, status)

filename
The path of the file of type CHARACTER*(*)

idispatch
An output argument that returns the object’s IDispatch interface pointer. Must be of type INTEGER(4).

status
The status of the operation. It can be any status returned by the CreateBindCtx or MkParseDisplayName routines, or the IMoniker::BindToObject method (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

COMInitialize

COM Subroutine: Initializes the COM library. This subroutine is only available on Windows* systems on IA-32 processors.

Module: USE IFCOM

Syntax

CALL COMInitialize (status)

status
The status of the operation. It can be any status returned by OleInitialize (see the Microsoft* Platform SDK). Must be of type INTEGER(4).

You must use this routine to initialize the COM library before calling any other COM or AUTO routine.

Example

program COMExample

use ifwin
use ifcom
use ifauto

   implicit none

   ! Variables
   integer(4) word_app
   integer(4) status
integer(4) invoke_args

call COMInitialize(status)

! Call GetActiveObject to get a reference to a running MS WORD
! application

call COMGetActiveObjectByProgID("Word.Application", word_app, status)

if (status >= 0) then

! Print the active document

   invoke_args = AutoAllocateInvokeArgs()
   call AutoAddArg(invoke_args, "Copies", 2)
   status = AutoInvoke(word_app, "PrintOut", invoke_args)
   call AutoDeallocateInvokeArgs(invoke_args)

! Release the reference

   status = COMReleaseObject(word_app)
end if

call COMUninitialize()

end program

---

**COMIsEqualGUID**

**COM Function:** Determines whether two globally unique identifiers (GUIDs) are the same. This function is only available on Windows® systems on IA-32 processors.

**Modules:** USE IFCOM, USE IFWINTY

**Syntax**

    result = COMIsEqualGUID (guid1, guid2)

**guid1**

The first GUID. Must be of type GUID, which is defined in the IFWINTY module. It can be any type of GUID, including a class identifier (CLSID), or an interface identifier (IID).

**guid2**

The second GUID, which will be compared to guid1. It must be the same type of GUID as guid1. For example, if guid1 is a CLSID, guid2 must also be a CLSID.

**Results:**

The result type is LOGICAL(4). The result is .TRUE. if the two GUIDs are the same; otherwise, .FALSE.
COMMITQQ

**Run-time Function:** Forces the operating system to execute any pending write operations for the file associated with a specified unit to the file’s physical device.

**Module:** USE IFCORE

**Syntax**

```fortran
result = COMMITQQ (unit)
```

*unit* (Input) INTEGER(4). A Fortran logical unit attached to a file to be flushed from cache memory to a physical device.

**Results:**

The result type is LOGICAL(4). If an open unit number is supplied, .TRUE. is returned and uncommitted records (if any) are written. If an unopened unit number is supplied, .FALSE. is returned.

Data written to files on physical devices is often initially written into operating-system buffers and then written to the device when the operating system is ready. 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 tells the operating system to write any cached data intended for a file on a physical device to that device immediately. This is called flushing the file.

COMMITQQ is most useful when you want to be certain that no loss of data occurs at a critical point in your program; for example, after a long calculation has concluded and you have written the results to a file, or after the user has entered a group of data items, or if you are on a network with more than one program sharing the same file. Flushing a file to disk provides the benefits of closing and reopening the file without the delay.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** the PRINT and WRITE statements in the Language Reference

**Example**

```fortran
USE IFCORE
INTEGER unit / 10 /
INTEGER len
CHARACTER(80) stuff
OPEN(unit, 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 (unit, *) stuff
IF (.NOT. COMMITQQ(unit)) WRITE (*,*) 'Failed'
END DO
CLOSE (unit)
END

COMPLINT, COMPLREAL, COMPLLOG

**Portability Functions:** Return a BIT-WISE complement or logical .NOT. of the argument.

**Module:** USE IFPORT

**Syntax**

- result = COMPLINT (intval)
- result = COMPLREAL (realval)
- result = COMPLLOG (logval)

- intval
  (Input) INTEGER(4).
- realval
  (Input) REAL(4).
- logval
  (Input) LOGICAL(4).

**Results:**

If the argument 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 COMPLLOG, just toggle 1<-->0.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

COMQueryInterface

**COM Subroutine:** Passes an interface identifier and returns a pointer to an object’s interface.

This subroutine is only available on Windows* systems on IA-32 processors.
**COMQueryInterface**

**Syntax**

```fortran
CALL COMQueryInterface (iunknown, iid, interface, status)
```

- `iunknown` - An IUnknown interface pointer. Must be of type INTEGER(4).
- `iid` - The interface identifier of the interface being requested. Must be of type GUID, which is defined in the IFWINTY module.
- `interface` - An output argument that returns the object’s interface pointer. Must be of type INTEGER(4).
- `status` - The status of the operation. It can be any status returned by the IUnknown method QueryInterface (see the Microsoft Platform SDK). Must be of type INTEGER(4).

**COMReleaseObject**

**COM Function**: Indicates that the program is done with a reference to an object’s interface. This function is only available on Windows systems on IA-32 processors.

**Module**: USE IFCOM

**Syntax**

```fortran
result = COMReleaseObject (iunknown)
```

- `iunknown` - An IUnknown interface pointer. Must be of type INTEGER(4).

**Results**: The result type is INTEGER(4). It is the object’s current reference count.

**Example**

See the example in “COMInitialize”.

**COMStringFromGUID**

**COM Subroutine**: Passes a globally unique identifier (GUID) and returns a string of printable characters. This subroutine is only available on Windows systems on IA-32 processors.

**Modules**: USE IFCOM, USE IFWINTY
**Descriptions of the Library Routines**

**Syntax**

CALL COMStringFromGUID (guid, string, status)

**guid**
The GUID to be converted. Must be of type GUID, which is defined in the IFWINTY module. It can be any type of GUID, including a class identifier (CLSID), or an interface identifier (IID).

**string**
A character variable of type CHARACTER*(*)(*) that receives the string representation of the GUID. The length of the character variable should be at least 38.

**status**
The status of the operation. If the string is too small to contain the string representation of the GUID, the value is zero. Otherwise, the value is the number of characters in the string representation of the GUID. Must be of type INTEGER(4).

The string representation of a GUID has a format like that of the following:

```
[c200e360-38c5-11ce-a6e2-08002b2b79ef]
```

where the successive fields break the GUID into the form DWORD-WORD-WORD-WORD-WORD.DWORD covering the 128-bit GUID. The string includes enclosing braces, which are an OLE convention.

**COMUninitialize**

**COM Subroutine:** Uninitializes the COM library. This subroutine is only available on Windows* systems on IA-32 processors.

**Module:** USE IFCOM

**Syntax**

CALL COMUninitialize ( )

When using COM routines, this must be the last routine called.

**Example**

See the example in “COMInitialize”.

**CSMG**

**Portability Function:** Performs an effective BIT-WISE store under mask.

**Module:** USE IFPORT
Syntax

result = CSMG (x, y, z)

x, y, z
(Input) INTEGER(4).

Results:
The result type is INTEGER(4). The result is equal to the following expression:

\[(x \& z) \mid (y \& \sim z)\]

where "\&" is a bitwise AND operation, \mid - bitwise OR, \sim - bitwise NOT.
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 \(y\).

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

CTIME

Portability Function: Converts a system time into a 24-character ASCII string.

Module: USE IFPORT

Syntax

result = CTIME (stime)

stime
(Input) INTEGER(4). An elapsed time in seconds since 00:00:00 Greenwich mean time, January 1, 1970.

Results:
The result is a value in the form Mon Jan 31 04:37:23 1994. Hours are expressed using a 24-hour clock.
The value of \(stime\) can be determined by calling the TIME function. CTIME(TIME( )) returns the current time and date.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: the DATE_AND_TIME intrinsic procedure in the Language Reference

Example

USE IFPORT

character (24) systime
systime = CTIME (TIME( ))
print *, 'Current date and time is ', systime

DATE

Portability Function or Subroutine: Returns the current system date.
Module: USE IFPORT
Function Syntax:
result = DATE ( )
Subroutine Syntax:
    CALL DATE (string)
string
(Output) CHARACTER. Variable or array containing at least nine bytes of storage.
DATE in its function form returns a CHARACTER string of length 8 in the form mm/dd/yy, where
mm, dd, and yy are two-digit representations of the month, day, and year, respectively.
DATE in its subroutine form returns string in the form dd-mmm-yy, where dd is a two-digit
representation of the current day of the month, mmm is a three-character abbreviation for the
current month (for example, Jan) and yy are the last two digits of the current year.

NOTE. DATE is an intrinsic procedure unless you specify USE IFPORT.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
See Also: the DATE subroutine in the Language Reference
Example
USE IFPORT
!If today's date is March 02, 2000, the following
!code prints "02-Mar-00"
CHARACTER(9) TODAY
CALL DATE(TODAY)
PRINT *, TODAY
!The next line prints "03/02/00"
PRINT *, DATE( )

2-43
DATE4

**Portability Subroutine:** Returns the current system date.

**Module:** USE IFPORT

**Syntax**

```
CALL DATE4 (datestr)
```

`datestr`

(Output) CHARACTER.

This subroutine returns `datestr` in the form dd-mmm-yyyy, where dd is a two-digit representation of the current day of the month, mmm is a three-character abbreviation for the current month (for example, Jan) and yyyy are the four digits of the current year.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

DBESJ0, DBESJ1, DBESJN, DBESY0, DBESY1, DBESYN

**Portability Functions:** Compute the double-precision values of Bessel functions of the first and second kinds.

**Module:** USE IFPORT

**Syntax**

```
result = DBESJ0 (posvalue)
result = DBESJ1 (posvalue)
result = DBESJN (n, posvalue)
result = DBESY0 (posvalue)
result = DBESY1 (posvalue)
result = DBESYN (n, posvalue)
```

`posvalue`

(Input) REAL(8). Independent variable for a Bessel function. Must be greater than or equal to zero.

`n`

(Input) 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 `posvalue`. 
DBESY0, DBESY1, and DBESYN return Bessel functions of the second kind, orders 0, 1, and \( n \), respectively, with the independent variable \( \text{posvalue} \).
Negative arguments cause DBESY0, DBESY1, and DBESYN to return a huge negative value.
Bessel functions are explained more fully in most mathematics reference books, such as the *Handbook of Mathematical Functions* (Abramowitz and Stegun. Washington: U.S. Government Printing Office, 1964). These functions are commonly used in the mathematics of electromagnetic wave theory.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** "BESJ0, BESJ1, BESJN, BESY0, BESY1, BESYN"

**Example**

```fortran
USE IFPORT
real(8) besnum, besout
10 read *, besnum
    besout = dbesj0(besnum)
    print *, 'result is ',besout
    goto 10
end
```

**DCLOCK**

**Portability Function:** Returns the elapsed time in seconds since the start of the current process.

**Module:** USE IFPORT

**Syntax**

```fortran
result = DCLOCK ( )
```

**Results:**
The result type is REAL(8). 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 subroutine.
Note that the first call to DCLOCK performs calibration.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** the DATE_AND_TIME and CPU_TIME subroutines in the *Language Reference*

**Example**

USE IFPORT
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

Portability Function: Deletes a specified directory.
Module: USE IFPORT
Syntax
result = DELDIRQQ (dir)
dir
(Input) Character*(*) String containing the path of the directory to be deleted.
Results:
The result type 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.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
See Also: “GETDRIVEDIRQQ”, “MAKEDIRQQ”, “CHANGEDIRQQ”,
“CHANGEDRIVEQQ”, “UNLINK”

DELETEMENUQQ

QuickWin Function: Deletes a menu item from a QuickWin menu. This function is only
available on Windows® systems.
Module: USE IFQWIN
Syntax
result = DELETEMENUQQ (menuID, itemID)
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.

**Results:**
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

**Compatibility**
QUICKWIN GRAPHICS LIB

**See Also:** “APPENDMENUQQ”, “INSERTMENUQQ”, “MODIFYMENUFAGSQQ”, “MODIFYMENUROUTINEQQ”, “MODIFYMENUSTRINGQQ”, "Using QuickWin" in Building Applications

**Example**
USE IFQWIN
LOGICAL(4) result
CHARACTER(25) str
str = 'Add to EDIT Menu'C   ! Append to 2nd menu
result = APPENDMENUQQ(2, $MENUENABLED, str, WINSTATUS)
! Delete third item (EXIT) from menu 1 (FILE)
result = DELETEMENUQQ(1, 3)
! Delete entire fifth menu (WINDOW)
result = DELETEMENUQQ(5,0)
END

**DELEFILESQQ**

**Portability Function:** Deletes all files matching the name specification, which can contain wildcards (* and ?).

**Module:** USE IFPORT

**Syntax**
```
result = DELEFILESQQ (files)
```

**files**
(Input) Character*(*) Files to be deleted. Can contain wildcards (* and ?).

**Results:**
The result type is INTEGER(2). The result 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** "FINDFILEQQ"

**Example**

USE IFPORT
USE IFCORE
INTEGER(4) len, count
CHARACTER(80) file
CHARACTER(1) ch
WRITE(*,*) "Enter names of files to delete: ", file
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')) STOP
END IF
count = DELFILESQQ(file)
WRITE(*,*) "Deleted ", count, " files."
END

**DFLOATI, DFLOATJ, DFLOATK**

**Portability Functions:** Convert an integer to double-precision real type.

**Module:** USE IFPORT

**Syntax**

\[
\text{result} = \text{DFLOATI} \ (i) \\
\text{result} = \text{DFLOATJ} \ (j) \\
\text{result} = \text{DFLOATK} \ (k)
\]

\*\*\* (Input) Must be of type INTEGER(2).\*

---

2-48
**DISPLAYCURSOR**

**Graphics Function:** Controls cursor visibility. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```plaintext
result = DISPLAYCURSOR (toggle)
```

**toggle**

(Input) INTEGER(2). Constant that defines the cursor state. Has two possible values:
- $GCURSOROFF - Makes the cursor invisible regardless of its current shape and mode.
- $GCURSORON - Makes the cursor always visible in graphics mode.

**Results:**
The result type is INTEGER(2). The result is the previous value of `toggle`.

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.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** "SETTEXTCURSOR", "SETWINDOWCONFIG"

**DLGEXIT**

**Dialog Subroutine:** Closes an open dialog box. This subroutine is only available on Windows* systems on IA-32 processors.
Module: USE IFLOGM

Syntax

CALL DLGEXIT (dlg)

dlg

(Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.

If you want to exit a dialog box on a condition other than the user selecting the OK or Cancel button, you need to include a call to DLGEXIT from within your callback routine. DLGEXIT saves the data associated with the dialog box controls and then closes the dialog box. The dialog box is exited after DLGEXIT has returned control back to the dialog manager, not immediately after the call to DLGEXIT.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “DLGSETRETURN”, “DLGINIT, DLGINITWITHRESOURCEHANDLE”, “DLGMODAL, DLGMODALWITHPARENT”, “DLGMODELESS”

Example

SUBROUTINE EXITSUB (dlg, exit_button_id, callbacktype)
USE IFLOGM
TYPE (DIALOG) dlg
INTEGER exit_button_id, callbacktype
...
CALL DLGEXIT (dlg)

DLGFLUSH

Dialog Subroutine: Updates the display of a dialog box. This subroutine is only available on Windows* systems on IA-32 processors.

Module: USE IFLOGM

Syntax

CALL DLGFLUSH (dlg [, flushall])

dlg

(Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.
flushall

(Input; optional) Logical. If .FALSE. (the default), then only the controls that the dialog routines have marked as changed are updated. If .TRUE.. all controls are updated with the state of the controls as known by the dialog routines. Normally, you would not set flushall to .TRUE..

When your application calls DLGSET to change a property of a control in a dialog box, the change is not immediately reflected in the displayed dialog box. Changes are applied when the dialog box is first displayed, and then after every dialog callback to the user’s code.

This design expects that, after a call to DLGMODAL or DLGMODELESS, every call to DLGSET will be made from within a callback routine, and that the callback routine finishes quickly. This is true most of the time.

However, there may be cases where you want to change a control outside of a dialog callback, or from within a loop in a dialog callback.

In these cases, DLGFLUSH is required, but is not always sufficient, to update the dialog display. DLGFLUSH sends pending Windows system messages to the dialog box and the controls that it contains. However, many display changes do not appear until after the program reads and processes these messages. A loop that processes the pending messages may be required; for example:

```fortran
use IFWINTY
use IFLOGM
use USER32
logical lNotQuit, lret
integer iret

TYPE (T_MSG) mesg
lNotQuit = .TRUE.
do while (lNotQuit .AND. (PeekMessage(mesg, 0, 0, 0, PM_NOREMOVE) <> 0))
   lNotQuit = GetMessage(mesg, NULL, 0, 0)
   if (lNotQuit) then
      if (DLGISDLGMESSAGE(mesg) .EQV. .FALSE) then
         lret = TranslateMessage(mesg)
         iret = DispatchMessage(mesg)
      end if
   end if
end do
```

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
DLGGET, DLGGETINT, DLGGETLOG, DLGGETCHAR

**Dialog Functions:** Return the state of the dialog control variable. These functions are only available on Windows systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

\[
\begin{align*}
\text{result} &= \text{DLGGET} (\text{dlg}, \text{controlid}, \text{value} [, \text{index}]) \\
\text{result} &= \text{DLGGETINT} (\text{dlg}, \text{controlid}, \text{value} [, \text{index}]) \\
\text{result} &= \text{DLGGETLOG} (\text{dlg}, \text{controlid}, \text{value} [, \text{index}]) \\
\text{result} &= \text{DLGGETCHAR} (\text{dlg}, \text{controlid}, \text{value} [, \text{index}])
\end{align*}
\]

- **dlg**
  - (Input) Derived type `dialog`. Contains dialog box parameters. The components of the type `dialog` are defined with the `PRIVATE` attribute, and cannot be changed or individually accessed by the user.

- **controlid**
  - (Input) Integer. Specifies the identifier of a control within the dialog box. Can be either the symbolic name for the control or the identifier number, both listed in the Include file (with extension .FD).

- **value**
  - (Output) Integer, logical, or character. The value of the control’s variable.

- **index**
  - (Input; optional) Integer. Specifies the control variable whose value is returned. Necessary if the control has more than one variable of the same data type and you do not want to get the value of the default for that type.

**Results:**

The result type is `LOGICAL(4)`. The result is `.TRUE.` if successful; otherwise, the result is `.FALSE.`.
Use the DLGGET functions to return the values of variables associated with your dialog box controls. Each control has at least one of the integer, logical, or character variable associated with it, but not necessarily all. The control variables are listed in the table in "Control Indexes" in Building Applications. The types of controls they are associated with are listed in the table in "Available Indexes for Each Dialog Control" in Building Applications.

You can use DLGGET to return the value of any variable. You can also use DLGGETINT to return an integer value, or DLGGETLOG and DLGGETCHAR to return logical and character values, respectively. If you use DLGGET, you do not have to worry about matching the function to the variable type. If you use the wrong function type for a variable or try to return a variable type that is not available, the DLGGET functions return .FALSE..

If two or more controls have the same controlid, you cannot use these controls in a DLGGET operation. In this case the function returns .FALSE..

The dialog box does not need to be open to access its control variables.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: "DLGSET, DLGSETINT, DLGSETLOG, DLGSETCHAR", "DLGSETSUB", "DLGINIT, DLGINITWITHRESOURCEHANDLE", "DLGMODAL, DLGMODALWITHPARENT", "DLGMODELESS", "Using Dialog Controls" and "Dialog Controls" in Building Applications

Example

USE IFLOGM
INCLUDE "THISDLG.FD"
TYPE (DIALOG)  dlg
INTEGER        val
LOGICAL        retlog, is_checked
CHARACTER(256) text
...
retlog = DLGGET (dlg, IDC_CHECKBOX1, is_checked, dlg_status)
retlog = DLGGET (dlg, IDC_SCROLLBAR2, val, dlg_range)
retlog = DLGGET (dlg, IDC_STATIC_TEXT1, text, dlg_title)
...

DLGINIT, DLGINITWITHRESOURCEHANDLE

Dialog Functions: Initialize a dialog box. These functions are only available on Windows* systems on IA-32 processors.

Module: USE IFLOGM
Syntax

\[
\text{result} = \text{DLGINIT} (id, \text{dlg}) \\
\text{result} = \text{DLGINITWITHRESOURCEHANDLE} (id, hinst, \text{dlg})
\]

`id`

(Input) INTEGER(4). Dialog identifier. Can be either the symbolic name for the dialog or the identifier number, both listed in the Include file (with extension .FD).

`dlg`

(Output) Derived type `dialog`. Contains dialog box parameters.

`hinst`

(Input) INTEGER(4). Module instance handle in which the dialog resource can be found.

**Results:**

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, the result is .FALSE..

DLGINIT must be called to initialize a dialog box before it can be used with DLGMODAL, DLGMODELESS, or any other dialog function.

DLGINIT will only search for the dialog box resource in the main application. For example, it will not find a dialog box resource that has been built into a dynamic link library.

DLGINITWITHRESOURCEHANDLE can be used when the dialog resource is not in the main application. If the dialog resource is in a dynamic link library (DLL), `hinst` must be the value passed as the first argument to the DLLMAIN procedure.

Dialogs can be used from any application, including console, QuickWin, and Windows applications.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “DLGEXIT”, “DLGMODAL, DLGMODALWITHPARENT”, “DLGMODELESS”, “DLGUNINIT”

**Example**

```
USE IFLOGM
INCLUDE 'DLG1.FD'
LOGICAL retlog
TYPE (DIALOG) thisdlg
... 
retlog = DLGINIT (IDD_DLG3, thisdlg)
IF (.not. retlog) THEN
```
WRITE (*,*) 'ERROR: dialog not found'
ELSE
...

**DLGISDLGMESSAGE, DLGISDLGMESSAGEWITHDLG**

**Dialog Functions:** Determine whether the specified message is intended for one of the currently displayed modeless dialog boxes, or a specific dialog box. These functions are only available on Windows* systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

\[
\text{result} = \text{DLGISDLGMESSAGE} (\text{mesg})
\]
\[
\text{result} = \text{DLGISDLGMESSAGEWITHDLG} (\text{mesg}, \text{dlg})
\]

*mesg*

*(Input)* Derived type `T_MSG`. Contains a Windows message.

*dnl*

*(Input)* Derived type `dialog`. Contains dialog box parameters. The components of the type `dialog` are defined with the `PRIVATE` attribute, and cannot be changed or individually accessed by the user.

**Results:**
The result type is `LOGICAL(4)`. The result is `.TRUE.` if the message is processed by the dialog box. Otherwise, the result is `.FALSE.` and the message should be further processed.

DLGISDLGMESSAGE must be called in the message loop of Windows applications that display a modeless dialog box using DLGMODELESS. DLGISDGMEMESSAGE determines whether the message is intended for one of the currently displayed modeless dialog boxes. If it is, it passes the message to the dialog box to be processed.

DLGISDLGMESSAGEWITHDLG specifies a particular dialog box to check. Use DLGISDLGMESSAGEWITHDLG when the message loop is in a main application and the currently active modeless dialog box was created by a DLL.

**Compatibility**

WINDOWS

**See Also:** ["DLGMODELESS"], "Using a Modeless Dialog Box" in Building Applications

**Example**

use IFLOGM
include 'resource.fd'
type (DIALOG) dlg
type (T_MSG) mesg
integer*4 ret
logical*4 ret

... Create the main dialog box and set up the controls and callbacks
lret = DlgInit(IDD_THERM_DIALOG, dlg)
lret = DlgSetSub(dlg, IDD_THERM_DIALOG, ThermSub)
...
lret = DlgModeless(dlg, nCmdShow)
...

! Read and process messages
do while( GetMessage (mesg, NULL, 0, 0) )
  ! Note that DlgIsDlgMessage must be called in order to give
  ! the dialog box first chance at the message.
  if ( DlgIsDlgMessage(mesg) .EQV. .FALSE. ) then
    lret = TranslateMessage( mesg )
    ret  = DispatchMessage( mesg )
  end if
end do

! Cleanup dialog box memory and exit the application
call DlgUninit(dlg)
WinMain = mesg%wParam
return

**DLGMODAL, DLGMODALWITHPARENT**

**Dialog Functions:** Display a dialog box and process user control selections made within the box. These functions are only available on Windows* systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

```
result = DLGMODAL (dlg)
result = DLGMODALWITHPARENT (dlg, hwndParent)
```

`dlg`
(Input) Derived type `dialog`. Contains dialog box parameters. The components of the type `dialog` are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.
hwndParent

(Input) Integer. Specifies the parent window for the dialog box. If omitted, the value is determined in this order:

1. If DLGMODAL is called from the callback of a modal or modeless dialog box, then that dialog box is the parent window.
2. If it is a QuickWin or Standard Graphics application, then the frame window is the parent window.
3. The Windows desktop window is the parent window.

Results:
The result type is INTEGER(4). By default, if successful, it returns the identifier of the control that caused the dialog to exit; otherwise, it returns −1. The return value can be changed with the DLGSETRETURN subroutine.

During execution, DLGMODAL displays a dialog box and then waits for user control selections. When a control selection is made, the callback routine, if any, of the selected control (set with DLGSETSUB) is called.

The dialog remains active until an exit control is executed: either the default exit associated with the OK and Cancel buttons, or DLGEXIT within your own control callbacks. DLGMODAL does not return a value until the dialog box is exited.

The default return value for DLGMODAL is the identifier of the control that caused it to exit (for example, IDOK for the OK button and IDCANCEL for the Cancel button). You can specify your own return value with DLGSETRETURN from within one of your dialog control callback routines. You should not specify −1 as your return value, because this is the error value DLGMODAL returns if it cannot open the dialog.

Use DLGMODALWITHPARENT when you want the parent window to be other than the default value (see the definition of hwndParent above). In particular, in an SDI or MDI Windows application, you may want the parent window to be the main application window. The parent window is disabled for user input while the modal dialog box is displayed.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “DLGSETRETURN”, “DLGSETSUB”, “DLGEXIT”, “DLGINIT, DLGINITWITHRESOURCEHANDLE”

Example
USE IFLOGM
INCLUDE "MYDLG.FD"
INTEGER return
TYPE (DIALOG) mydialog
return = DLGMODAL (mydialog)

**DLGMODELESS**

**Dialog Function:** Displays a modeless dialog box. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

```fortran
result = DLGMODELESS (dlg [, nCmdShow, hwndParent])
```

*dlg*

(Input) Derived type *dialog*. Contains dialog box parameters. The components of the type *dialog* are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user. The variable passed to this function must remain in memory for the duration of the dialog box, that is from the DLGINIT call through the DLGUNINIT call.

The variable can be declared as global data in a module, as a variable with the STATIC attribute, or in a calling procedure that is active for the duration of the dialog box. It must not be an AUTOMATIC variable in the procedure that calls DLGMODELESS.

*nCmdShow*

(Input) Integer. Specifies how the dialog box is to be shown. It must be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_HIDE</td>
<td>Hides the dialog box.</td>
</tr>
<tr>
<td>SW_MINIMIZE</td>
<td>Minimizes the dialog box.</td>
</tr>
<tr>
<td>SW_RESTORE</td>
<td>Activates and displays the dialog box. If the dialog box is minimized or maximized, the Windows system restores it to its original size and position.</td>
</tr>
<tr>
<td>SW_SHOW</td>
<td>Activates the dialog box and displays it in its current size and position.</td>
</tr>
<tr>
<td>SW_SHOWMAXIMIZED</td>
<td>Activates the dialog box and displays it as a maximized window.</td>
</tr>
<tr>
<td>SW_SHOWMINIMIZED</td>
<td>Activates the dialog box and displays it as an icon.</td>
</tr>
<tr>
<td>SW_SHOWMINNOACTIVE</td>
<td>Displays the dialog box as an icon. The window that is currently active remains active.</td>
</tr>
</tbody>
</table>
The default value is SW_SHOWNORMAL.

hwndParent

(Input) Integer. Specifies the parent window for the dialog box. The default value is determined in this order:
1. If DLGMODELESS is called from a callback of a modeless dialog box, then that dialog box is the parent window.
2. The Windows desktop window is the parent window.

Results:
The result type is LOGICAL(4). The value is .TRUE. if the function successfully displays the dialog box. Otherwise the result is .FALSE..

During execution, DLGMODELESS displays a modeless dialog box and returns control to the calling application. The dialog box remains active until DLGEXIT is called, either explicitly or as the result of the invocation of a default button callback.

DLGMODELESS is typically used in a Windows application. The application must contain a message loop that processes Windows messages. The message loop must call DLGISDLGMESSAGE for each message (see the example below). Multiple modeless dialog boxes can be displayed at the same time. A modal dialog box can be displayed from a modeless dialog box by calling DLGMODAL from a modeless dialog callback. However, DLGMODELESS cannot be called from a modal dialog box callback.

DLGMODELESS also can be used in a Console, DLL, or LIB project. However, the requirements remain that the application must contain a message loop and must call DLGISDLGMESSAGE for each message. For an example of calling DLGMODELESS in a DLL project, see the Dilprgrs sample in the ...\SAMPLES\DIALOG folder.

Use the DLG_INIT callback with DLGSETSUB to perform processing immediately after the dialog box is created and before it is displayed, and to perform processing immediately before the dialog box is destroyed.

Compatibility

WINDOWS  CONSOLE  DLL  LIB
See Also: “DLGSETSUB”, “DLGINIT, DLGINITWITHRESOURCEHANDLE”, “DLGEXIT”, “DLGISDLGMESSAGE, DLGISDLGMESSAGewithDLG”, “Using a Modeless Dialog Box” in Building Applications

Example

use IFLOGM
include 'resource.fd'
type (DIALOG)   dlg
type (T_MSG)    mesg
integer*4   ret
logical*4   lret
...
! Create the main dialog box and set up the controls and callbacks
lret = DlgInit(IDD_THERM_DIALOG, dlg)
lret = DlgSetSub(dlg, IDD_THERM_DIALOG, ThermSub)
...
lret = DlgModeless(dlg, nCmdShow)
...
! Read and process messsages
do while( GetMessage (mesg, NULL, 0, 0) )
    ! Note that DlgIsDlgMessage must be called in order to give
    ! the dialog box first chance at the message.
    if ( DlgIsDlgMessage(mesg) .EQV. .FALSE. ) then
        lret = TranslateMessage( mesg )
        ret  = DispatchMessage( mesg )
    end if
end do
! Cleanup dialog box memory and exit the application
call DlgUninit(dlg)
WinMain = mesg%wParam
return

DLGSENDCTRLMESSAGE

Dialog Function: Sends a Windows* message to a dialog box control. This function is only available on Windows* systems on IA-32 processors.

Module: USE IFLOGM
Syntax

result = DKGSENDCTRLMESSAGE (dlg, controlid, msg, wparam, lparam)

dlg
(Input) Derived type dialog. Contains dialog box parameters. The components of the type
dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed
by the user.

controlid
(Input) Integer. Specifies the identifier of the control within the dialog box. Can be either the
symbolic name for the control or the identifier number, both listed in the Include file (with
extension .FD).

msg
(Input) Integer. Derived type T_MSG. Specifies the message to be sent.

wparam
(Input) Integer. Specifies additional message specific information.

lparam
(Input) Integer. Specifies additional message specific information.

Results:
The result type is INTEGER(4). The value specifies the result of the message processing and
depends upon the message sent.

The dialog box must be currently active by a call to DLGMODAL or DLGMODELESS. This
function does not return until the message has been processed by the control.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “DLGINIT, DLGINITWITHRESOURCEHANDLE”, “DLGSETSUB”,
“DLGMODAL, DLGMODALWITHPARENT”, “DLGMODELESS”

Example

use IFLOGM
include 'resource.fd'
type (dialog)  dlg
integer        callbacktype
integer        cref
integer        iret

if (callbacktype == dlg_init) then
    ! Change the color of the Progress bar to red
! NOTE: The following message succeeds only if Internet Explorer 4.0
! or later is installed
cref = Z'FF'    ! Red
iret = DlgSendCtrlMessage(dlg, IDC_PROGESS1, PBM_SETBARCOLOR, 0, cref)
endif

**DLGSET, DLGSETINT, DLGSETLOG, DLGSETCHAR**

**Dialog Functions:** Set the values of dialog control variables. These functions are only available on Windows* systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

```fortran
result = DLGSET (dlg, controlid, value [, index])
result = DLGSETINT (dlg, controlid, value [, index])
result = DLGSETLOG (dlg, controlid, value [, index])
result = DLGSETCHAR (dlg, controlid, value [, index])
```

**dlg**

(Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.

**controlid**

(Input) Integer. Specifies the identifier of a control within the dialog box. Can be either the symbolic name for the control or the identifier number, both listed in the Include file (with extension .FD).

**value**

(Input) Integer, logical, or character. The value of the control’s variable.

**index**

(Input; optional) Integer. Specifies the control variable whose value is set. Necessary if the control has more than one variable of the same data type and you do not want to set the value of the default for that type.

**Results:**

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, the result is .FALSE..
Use the DLGSET functions to set the values of variables associated with your dialog box controls. Each control has at least one of the integer, logical, or character variables associated with it, but not necessarily all. The control variables are listed in the table in "Control Indexes" in Building Applications. The types of controls they are associated with are listed in the table in "Available Indexes for Each Dialog Control" in Building Applications.

You can use DLGSET to set any control variable. You can also use DLGSETINT to set an integer variable, or DLGSETLOG and DLGSETCHAR to set logical and character values, respectively. If you use DLGSET, you do not have to worry about matching the function to the variable type. If you use the wrong function type for a variable or try to set a variable type that is not available, the DLGSET functions return .FALSE..

Calling DLGSET does not cause a callback routine to be called for the changing value of a control. In particular, when inside a callback, performing a DLGSET on a control does not cause the associated callback for that control to be called. Callbacks are invoked automatically only by user action on the controls in the dialog box. If the callback routine needs to be called, you can call it manually after the DLGSET is executed.

If two or more controls have the same controlid, you cannot use these controls in a DLGSET operation. In this case the function returns .FALSE..

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: "DLGSETSUB", "DLGGET, DLGGETINT, DLGGETLOG, DLGGETCHAR". "Using Dialog Controls", "Dialog Routines", and "Dialog Controls" in Building Applications

**Example**

USE IFLOGM
INCLUDE "DLGRADAR.FD"
TYPE (DIALOG)  dlg
LOGICAL        retlog
...
retlog = DLGSET (dlg, IDC_SCROLLBAR1, 400, dlg_range)
retlog = DLGSET (dlg, IDC_CHECKBOX1, .FALSE., dlg_status)
retlog = DLGSET (dlg, IDC_RADIOBUTTON1, "Hot Button", dlg_title)
...

**DLGSETCTRLLEVENTHANDLER**

**Dialog Function:** Assigns user-written event handlers to ActiveX® controls in a dialog box. This function is only available on Windows® systems on IA-32 processors.

**Module:** USE IFLOGM
Syntax

result = DLGSETCTRLEVENTHANDLER (dlg, controlid, handler, dispid [, iid])

dlg

(Input) Derived type dialog. Contains dialog box parameters. The components of the type DIALOG are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.

controlid

(Input) Integer. Specifies the identifier of a control within the dialog box. Can be the symbolic name for the control or the identifier number, both listed in the include (with extension .FD) file.

handler

(Input) Name of the routine to be called when the event occurs. It must be declared EXTERNAL.

dispid

(Input) Integer. Specifies the member id of the method in the event interface that identifies the event.

iid

(Input; optional) Derived type GUID, which is defined in the IFWINTY module. Specifies the interface identifier of the source (event) interface. If omitted, the default source interface of the ActiveX control is used.

Results:

The result type is INTEGER(4). The result is an HRESULT describing the status of the operation. When the ActiveX control event occurs, the handler associated with the event is called. You call DLGSETCTRLEVENTHANDLER to specify the handler to be called.

The events supported by an ActiveX control and the interfaces of the handlers are determined by the ActiveX control.

You can find this information in one of the following ways:

- By reading the documentation of the ActiveX control.
- By using a tool that lets you examine the type information of the ActiveX control. The OLE-COM Object Viewer in the Intel® Visual Fortran folder is one such tool.
- By using the Fortran Module Wizard to generate a module that contains Fortran interfaces to the ActiveX control, and examining the generated module.

The handler that you define in your application must have the interface that the ActiveX control expects, including calling convention and parameter passing mechanisms. Otherwise, your application will likely crash in unexpected ways because of the application’s stack getting corrupted.
Note that an object is always the first parameter in an event handler. This object value is a pointer to the control’s source (event) interface, not the IDispatch pointer of the control. You can use DLGGET with the DLG_IDISPATCH index to retrieve the control’s IDispatch pointer. For more information, see "Using ActiveX Controls" in *Building Applications*.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** “DLGINIT, DLGINITWITHRESOURCEHANDLE”, “DLGGET, DLGGETINT, DLGGETLOG, DLGGETCHAR”, “DLGMODAL, DLGMODALWITHPARENT”, “DLGMODELESS”, “DLGSETSUB”

**Example**

```fortran
USE IFLOGM
ret = DlgSetCtrlEventHandler(            &
  dlg,                         &
  IDC_ACTIVEMOVIECONTROL1,     & ! Identifies the control
  ReadyStateChange,            & ! Name of the event handling routine
  -609,                        & ! Member id of the ActiveMovie’s
  & ! control ReadyStateChange event.
  IID_DActiveMovieEvents2 )     ! Identifier of the source (event)
                                 ! interface.
```

---

**DLGSETRETURN**

**Dialog Subroutine:** Sets the return value for the DLGMODAL function from within a callback subroutine. This subroutine is only available on Windows* systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

```fortran
CALL DLGSETRETURN (dlg, retval)
```

*dlg*  
(Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.

*retval*  
(Input) Integer. Specifies the return value for DLGMODAL upon exiting.
DLGSETRETURN overrides the default return value with retval. You can set your own value as a means of determining the condition under which the dialog box was closed. The default return value for an error condition is −1, so you should not use −1 as your return value.

DLGSETRETURN should be called from within a callback routine, and is generally used with DLGEXIT, which causes the dialog box to be exited from a control callback rather than the user selecting the OK or Cancel button.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** “DLGEXIT”, “DLGMODAL, DLGMODALWITHPARENT”

**Example**

```fortran
SUBROUTINE SETRETSUB (dlg, button_id, callbacktype)
USE IFLOGM
INCLUDE "MYDLG.FD"
TYPE (DIALOG) dlg
LOGICAL     is_checked, retlog
INTEGER     return, button_id, callbacktype
...
retlog = DLGGET(dlg, IDC_CHECKBOX4, is_checked, dlg_state)
IF (is_checked) THEN
    return = 999 ELSE    return = -999
END IF
CALL DLGSETRETURN (dlg, return)
CALL DLGEXIT (dlg)
END SUBROUTINE SETRETSUB
```

**DLGSETSUB**

**Dialog Function:** Assigns your own callback subroutines to dialog controls and to the dialog box. This function is only available on Windows* systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

```fortran
result = DLGSETSUB (dlg, controlid, value [, index])
```

*dlg*  
(Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.
controlid
(Input) Integer. Specifies the identifier of a control within the dialog box. Can be the symbolic name for the control or the identifier number, both listed in the include (with extension .FD) file, or it can be the identifier of the dialog box.

value
(Input) EXTERNAL. Name of the routine to be called when the callback event occurs.

index
(Input; optional) Integer. Specifies which callback routine is executed when the callback event occurs. Necessary if the control has more than one callback routine.

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

When a callback event occurs (for example, when you select a check box), the callback routine associated with that callback event is called. You use DLGSETSUB to specify the subroutine to be called. All callback routines should have the following interface:

```plaintext
SUBROUTINE callbackname (dlg, controlid, callbacktype)
!DEC$ ATTRIBUTES DEFAULT :: callbackname
callbackname
Is the name of the callback routine.
dlg
Refers to the dialog box and allows the callback to change values of the dialog controls.
controlid
Is the name of the control that caused the callback.
callbacktype
Indicates what callback is occurring (for example, DLG_CLICKED, DLG_CHANGE, or DLG_DBLCLICK).
```

The `controlid` and `callbacktype` parameters let you write a single subroutine that can be used with multiple callbacks from more than one control. Typically, you do this for controls comprising a logical group. You can also associate more than one callback routine with the same control, but you must use then use `index` parameter to indicate which callback routine to use.

The `controlid` can also be the identifier of the dialog box. The dialog box supports two `callbacktypes`, DLG_INIT and DLG_SIZECHANGE. The DLG_INIT callback is executed immediately after the dialog box is created with `callbacktype` DLG_INIT, and immediately before the dialog box is destroyed with `callbacktype` DLG_DESTROY. DLG_SIZECHANGE is called when the size of a dialog is changed.
Callback routines for a control are called after the value of the control has been updated based on the user’s action.

If two or more controls have the same controlid, you cannot use these controls in a DLGSETSUB operation. In this case, the function returns .FALSE..

For more information, see "Dialog Callback Routines" in Building Applications.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “DLGSET, DLGSETINT, DLGSETLOG, DLGSETCHAR”, “DLGGET, DLGGETINT, DLGGETLOG, DLGGETCHAR”

Example
PROGRAM DLGPROG
USE IFLOGM
INCLUDE "MYDLG.FD"
TYPE (dialog) mydialog
LOGICAL retlog
INTEGER return
EXTERNAL RADIOSUB
retlog = DLGINIT(IDD_mydlg, dlg)
retlog = DLGSETSUB (mydialog, IDC_RADIO_BUTTON1, RADIOSUB)
retlog = DLGSETSUB (mydialog, IDC_RADIO_BUTTON2, RADIOSUB)
return = DLGMODAL(dlg)
END
SUBROUTINE RADIOSUB( dlg, id, callbacktype )
!DEC$ ATTRIBUTES DEFAULT :: callbackname
USE IFLOGM
TYPE (dialog) dlg
INTEGER id, callbacktype
INCLUDE 'MYDLG.FD'
CHARACTER(256) text
INTEGER cel, far, retint
LOGICAL retlog
SELECT CASE (id)
CASE (IDC_RADIO_BUTTON1)
! Radio button 1 selected by user so
! change text accordingly
  text = 'Statistics Package A'
Descriptions of the Library Routines

DLGSETTITLE

Dialog Subroutine: Sets the title of a dialog box. This subroutine is only available on Windows* systems on IA-32 processors.

Module: USE IFLOGM

Syntax

CALL DLGSETTITLE (dlg, title)

dlg
(Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.

title
(Input) Character*(*). Specifies text to be the title of the dialog box.

Use this routine when you want to specify the title for a dialog box.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “DLGINIT, DLGINITWITHRESOURCEHANDLE”, “DLGMODAL, DLGMODALWITHPARENT”, “DLGMODELESS”

Example

USE IFLOGM
INCLUDE "MYDLG.FD"
TYPE (DIALOG) mydialog
LOGICAL retlog
...
retlog = DLGINIT(IDD_mydlg, mydialog)
...
CALL DLGSETTITLE(mydialog, "New Title")
...

**DLGUNINIT**

**Dialog Subroutine:** Deallocates memory associated with an initialized dialog. This subroutine is only available on Windows® systems on IA-32 processors.

**Module:** USE IFLOGM

**Syntax**

```fortran
CALL DLGUNINIT (dlg)
```

*dlg* (Input) Derived type dialog. Contains dialog box parameters. The components of the type dialog are defined with the PRIVATE attribute, and cannot be changed or individually accessed by the user.

You should call DLGUNINIT when a dialog that was successfully initialized by DLGINIT is no longer needed. DLGUNINIT should only be called on a dialog initialized with DLGINIT. If it is called on an uninitialized dialog or one that has already been deallocated with DLGUNINIT, the result is undefined.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** "DLGINIT, DLGINITWITHRESOURCEHANDLE", "DLGMODAL, DLGMODALWITHPARENT", "DLGMODELESS", "DLGEXIT"

**Example**

```fortran
USE IFLOGM
INCLUDE "MYDLG.FD"
TYPE (DIALOG)  mydialog
LOGICAL        retlog
...
retlog = DLGINIT(IDD_mydlg, mydialog)
...
CALL DLGUNINIT (mydialog)
END
```

**DRAND, DRANDM**

**Portability Functions:** Return double-precision random numbers in the range 0.0 through 1.0.
Module: USE IFPORT

Syntax

\[
\text{result} = \text{DRAND} \ (i\text{f}l\text{ag}) \\
\text{result} = \text{DRANDM} \ (i\text{f}l\text{ag})
\]

\text{iflag}

(Input) INTEGER(4). Controls the way the random number is selected.

Results:

The result type is REAL(8). Return values are:

<table>
<thead>
<tr>
<th>Value of \textit{iflag}</th>
<th>Selection process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The generator is restarted and the first random 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 \textit{iflag}, then restarted, and the first random value is selected.</td>
</tr>
</tbody>
</table>

There is no difference between DRAND and DRANDM. Both functions are included to insure portability of existing code that references one or both of them.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: the RANDOM\_NUMBER and RANDOM\_SEED intrinsic procedures in the Language Reference

Example

USE IFPORT

REAL(8) num

INTEGER(4) f

f=1

CALL print_rand

f=0

CALL print_rand

f=22

CALL print_rand

CONTAINS

SUBROUTINE print_rand

\[
\text{num} = \text{drand} \ (f)
\]

print *, 'f= ',f,:',num
DRANSET

Portability Subroutine: Sets the seed for the random number generator.
Module: USE IFPORT
Syntax
   CALL DRANSET (seed)
   seed
   (Input) REAL(8). The reset value for the seed.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
See Also: “RANSET”

DTIME

Portability Function: Returns the elapsed CPU time since the start of program execution when first called, and the elapsed execution time since the last call to DTIME thereafter.
Module: USE IFPORT
Syntax
   result = DTIME (tarray)
   tarray
   (Output) REAL(4). A rank one array with two elements:
   • tarray(1) – Elapsed user time, which is time spent executing user code. This value includes time running protected Windows subsystem code.
   • tarray(2) – Elapsed system time, which is time spent executing privileged code (code in the Windows Executive).
Results:
The result type is REAL(4). The result is the total CPU time, which is the sum of tarray(1) and tarray(2). If an error occurs, –1 is returned.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
See Also: the DATE_AND_TIME and CPU_TIME intrinsic procedures in the Language Reference

Example

USE IFPORT
REAL(4) I, TA(2)
I = DTIME(TA)
write(*,*) 'Program has been running for', I, 'seconds.'
write(*,*) ' This includes', TA(1), 'seconds of user time and', &
 & TA(2), 'seconds of system time.'

ELLIPSE, ELLIPSE_W

Graphics Functions: Draw a circle or an ellipse using the current graphics color. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = ELLIPSE (control, x1, y1, x2, y2)
result = ELLIPSE_W (control, wx1, wy1, wx2, wy2)

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.

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.

Results:
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.
The border is drawn in the current color and line style.

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 \((x_2, y_2)\). When you use ELLIPSE_W, the center of the ellipse is the center of the bounding rectangle defined by the window-coordinate points \((wx_1, wy_1)\) and \((wx_2, wy_2)\). 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.

**NOTE.** The ELLIPSE routine described here is a QuickWin routine. If you are trying to use the Microsoft* Platform SDK version of the Ellipse routine by including the IFWIN module, you need to specify the routine name as MSFWIN$Ellipse. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “FLOODFILLRGB, FLOODFILLRGB_W”, “GRSTATUS”, “SETCOLORRGB”, “SETFILLMASK”

**Example**

This program draws the shape shown below.

```fortran
! compile as QuickWin or Standard Graphics application
USE IFQWIN
INTEGER(2) dummy, x1, y1, x2, y2
x1 = 80;  y1 = 50
x2 = 240; y2 = 150
dummy = ELLIPSE( $GFILLINTERIOR, x1, y1, x2, y2 )
END
```
ETIME

Portability Function: Returns the elapsed CPU time, in seconds, of the process that calls it.

Module: USE IFPORT

Syntax

result = ETIME (array)

tarray

(Output) REAL(4). Must be a rank one array with two elements:

• array(1) – Elapsed user time, which is time spent executing user code. This value includes
time running protected Windows subsystem code.

• array(2) – Elapsed system time, which is time spent executing privileged code (code in the
Windows Executive).

Results:
The result type is REAL(4). The result is the total CPU time, which is the sum of array(1) and
array(2). If an error occurs, –1 is returned.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: the DATE_AND_TIME and CPU_TIME intrinsic procedures in the Language
Reference

Example

USE IFPORT
REAL(4), I, TA(2)
I = ETIME(TA)
write(*,*) 'Program has used', I, 'seconds of CPU time.'
write(*,*) ' This includes', TA(1), 'seconds of user time and', &
FDATE

Portability Function or Subroutine: Returns the current date and time as an ASCII string.

Module: USE IFPORT

Function Syntax:

result = FDATE ( )

Subroutine Syntax:

CALL FDATE (string)

string

(Optional; Output) Character*(*). It is returned as a 24-character string in the form:

Mon Jan 31 04:37:23 2001

Any value in string before the call is destroyed.

Results:

The result of the function FDATE and the value of string returned by the subroutine FDATE(string) are identical. Newline and NULL are not included in the string.

When you use FDATE as a function, declare it as:

CHARACTER*24 FDATE

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: the DATE_AND_TIME intrinsic procedure in the Language Reference

Example

USE IFPORT

CHARACTER*24 today

! CALL FDATE(today)

write (*,*), 'Today is ', today

! write (*,*), 'Today is ', fdate()
Module: USE IFPORT

Syntax

result = FGETC (lunit, char)

lunit

(Input) INTEGER(4). Unit number of a file. Must be currently connected to a file when the function is called.

char

(Output) CHARACTER*1. Next available character in the file. If lunit is connected to a console device, then no characters are returned until the Enter key is pressed.

Results:

The result type is INTEGER(4). The result is zero if the read is successful, or –1 if an end-of-file is detected. A positive value is either a system error code or a Fortran I/O error code, such as:

EINVAL: The specified unit is invalid (either not already open, or an invalid unit number).

If you use WRITE, READ, or any other Fortran I/O statements with lunit, be sure to read "Portability Routines" in Building Applications.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: "GETCHARQQ", the READ statement in the Language Reference

Example

USE IFPORT
CHARACTER inchar
INTEGER istatus
istatus = FGETC(5,inchar)
PRINT *, inchar
END

FINDFILEQQ

Portability Function: Searches for a specified file in the directories listed in the path contained in the environment variable.

Module: USE IFPORT

Syntax

result = FINDFILEQQ (filename, varname, pathbuf)
filename
(Input) Character*(*) . Name of the file to be found.

varname
(Input) Character*(*) . Name of an environment variable containing the path to be searched.

pathbuf
(Output) Character*(*) . Buffer to receive the full path of the file found.

**Results:**
The result type is INTEGER(4). The result is the length of the string containing the full path of the found file returned in *pathbuf*, or 0 if no file is found.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** “FULLPATHQQ”, “GETFILEINFOQQ”, “SPLITPATHQQ”

**Example**
USE IFPORT
CHARACTER(256) pathname
INTEGER(4) pathlen
pathlen = FINDFILEQQ("libfmt.lib", "LIB", pathname)
WRITE (*,*) pathname
END

---

**FLOODFILL, FLOODFILL_W**

**Graphics Functions:** Fill an area using the current color index and fill mask. These functions are only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = FLOODFILL (x, y, bcolor)
result = FLOODFILL_W (wx, wy, bcolor)
```

**x, y**
(Input) INTEGER(2) . Viewport coordinates for fill starting point.

**bcolor**
(Input) INTEGER(2) . Color index of the boundary color.
wx, wy

(Input) REAL(8). Window coordinates for fill starting point.

**Results:**
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 begins filling at the viewport-coordinate point (x, y). FLOODFILL_W begins filling at the window-coordinate point (wx, wy). The fill color used by FLOODFILL and 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.

---

**NOTE.** The FLOODFILL routine described here is a QuickWin routine. If you are trying to use the Microsoft* Platform SDK version of the FloodFill routine by including the IFWIN module, you need to specify the routine name as MSFWIN$FloodFill. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

---

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:**  "FLOODFILLRGB, FLOODFILLRGB_W", "ELLIPSE, ELLIPSE_W", "GETCOLOR", "GETFILLMASK", "GRSTATUS", "PIE, PIE_W", "SETCLIPRGN", "SETCOLOR", "SETFILLMASK"

**Example**

USE IFQWIN

INTEGER(2) status, bcolor, red, blue
INTEGER(2) x1, y1, x2, y2, xinterior, yinterior
x1 = 80; y1 = 50
x2 = 240; y2 = 150
red = 4
blue = 1
status = SETCOLOR(red)
status = RECTANGLE( $GBORDER, x1, y1, x2, y2 )
bcolor = GETCOLOR()
status = SETCOLOR (blue)
xinterior = 160; yinterior = 100
status = FLOODFILL (xinterior, yinterior, bcolor)
END

FLOODFILLRGB, FLOODFILLRGB_W

Graphics Functions: Fill an area using the current Red-Green-Blue (RGB) color and fill mask. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = FLOODFILLRGB (x, y, color)
result = FLOODFILLRGB_W (wx, wy, color)

x, y
(Input) INTEGER(2). Viewport coordinates for fill starting point.

color
(Input) INTEGER(4). RGB value of the boundary color.

wx, wy
(Input) REAL(8). Window coordinates for fill starting point.

Results:
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).

FLOODFILLRGB begins filling at the viewport-coordinate point (x, y). FLOODFILLRGB_W begins filling at the window-coordinate point (wx, wy). The fill color used by FLOODFILLRGB and 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 color.
Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB


Example
! Build as a QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(2) status
INTEGER(4) result, bcolor
INTEGER(2) x1, y1, x2, y2, xinterior, yinterior
x1 = 80; y1 = 50
x2 = 240; y2 = 150
result = SETCOLORRGB(Z'008080') ! red
status = RECTANGLE( $GBORDER, x1, y1, x2, y2 )
bcolor = GETCOLORRGB( )
result = SETCOLORRGB (Z'FF0000') ! blue
xinterior = 160; yinterior = 100
result = FLOODFILLRGB (xinterior, yinterior, bcolor)
END

FLUSH

Portability Subroutine: Flushes the contents of an external unit buffer into its associated file.
Module: USE IFPORT

Syntax
CALL FLUSH (lunit)
lunit
(Input) INTEGER(4). Number of the external unit to be flushed. Must be currently connected to a file when the subroutine is called. This routine is thread-safe, and locks the associated stream before I/O is performed.
FOCUSQQ

QuickWin Function: Sets focus to the window with the specified unit number. This function is only available on Windows® systems.

Module: USE IFQWIN

Syntax

result = FOCUSQQ (iunit)

iunit

(Input) INTEGER(4). Unit number of the window to which the focus is set. Unit numbers 0, 5, and 6 refer to the default startup window.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero.

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.)

NOTE. The flush is performed in a non-blocking mode. In this mode, the command may return before the physical write is completed. If you want to use a blocking mode of FLUSH use COMMITQQ.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “COMMITQQ”
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.

**Compatibility**

QUICKWIN GRAPHICS LIB

See Also: "SETACTIVEQQ", "INQFOCUSQQ", "Using QuickWin" in *Building Applications*

### FOR_DESCRIPTOR_ASSIGN

**Run-time Subroutine:** Creates an array descriptor in memory. This subroutine is only available on Windows* systems.

**Module:** USE IFCORE

**Syntax**

```fortran
CALL FOR_DESCRIPTOR_ASSIGN (dp, base, size, reserved, rank, dims_info)
```

- **dp** (Input) A Fortran 95/90 pointer to an array; the array can be of any data type.
- **base** (Input) INTEGER(4) or INTEGER(8). The base address of the data being described by `dp`. Note that a Fortran 95/90 pointer describes both the location and type of the data item.
- **size** (Input) INTEGER(4). The size of the data type; for example, 4 for INTEGER(4).
- **reserved** (Input) INTEGER(4). A combination (using bitwise OR) of the following symbolic constants, which are defined in `IFCORE.F90`:
  - **FOR_DESCRIPTOR_ARRAY_DEFINED** – Specifies whether the array pointed to has been allocated or associated. If the bit is set, the array has been allocated or associated.
  - **FOR_DESCRIPTOR_ARRAY_NODEALLOC** – Specifies whether the array points to something that can be deallocated by a call to DEALLOCATE, or whether it points to something that cannot be deallocated. For example:
    ```fortran
    integer, pointer :: p(:)
    integer, target :: t
    p => t  ! t cannot be deallocated
    allocate(p(10))  ! t can be deallocated
    ```
    If the bit is set, the array cannot be deallocated.
• FOR_DESCRIPTOR_ARRAY_CONTIGUOUS – Specifies whether the array pointed to is completely contiguous in memory or whether it is a slice that is not contiguous. If the bit is set, the array is contiguous.

\textit{rank} \\
(Input) INTEGER(4). The rank of the array pointed to.

\textit{dims_info} \\
(Input) An array of derived type FOR_DIMS_INFO; you must specify a rank for this array. The derived type FOR_DIMS_INFO is defined in IFCORE.F90 as follows:

\begin{verbatim}
TYPE FOR_DIMS_INFO
  INTEGER(4) LOWERBOUND  !Lower bound for the dimension
  INTEGER(4) UPPERBOUND  !Upper bound for the dimension
  INTEGER(4) STRIDE      !Stride for the dimension
END TYPE FOR_DIMS_INFO
\end{verbatim}

The FOR_DESCRIPTOR_ASSIGN routine is similar to a Fortran 95/90 pointer assignment, but gives you more control over the assignment, allowing, for example, assignment to any location in memory.

You can also use this routine to create an array that can be used from both Fortran or C.

\textbf{See Also}: the POINTER Attribute and Statement in the \textit{Language Reference}

\textbf{Example}

\begin{verbatim}
use IFCORE
common/c_array/ array
real(8) array(5,5)
external  init_array
external  c_print_array
real(8),pointer :: p_array(:,:)
type(FOR_DIMS_INFO) dims_info(2)

  call init_array()

  do i=1,5
    do j=1,5
      print *,i,j, array(i,j)
    end do
  end do

  dims_info(1)%LOWERBOUND = 11
  dims_info(1)%UPPERBOUND = 15
\end{verbatim}
The following shows the C program containing init_array and c_print_array:

```c
#include <stdio.h>

#ifdef (!_WIN32) && (!_WIN64)
#define C_ARRAY c_array_
#define INIT_ARRAY init_array_
#define C_PRINT_ARRAY c_print_array_
#endif

double C_ARRAY[5][5];
void INIT_ARRAY(void);
void C_PRINT_ARRAY(void);

void INIT_ARRAY(void)
{
    int i, j;
    for (i = 0; i < 5; i++)
        for (j = 0; j < 5; j++)
            C_ARRAY[i][j] = j + 10 * i;
}

void C_PRINT_ARRAY(void)
```

```c
dims_info(1)%STRIDE = 1
dims_info(2)%LOWERBOUND = -5
dims_info(2)%UPPERBOUND = -1
dims_info(2)%STRIDE = 1

call FOR_DESCRIPTOR_ASSIGN(p_array, &
    LOC(array), &
    SIZEOF(array(1,1)), &
    FOR_DESCRIPTOR_ARRAY_DEFINED .or. &
    FOR_DESCRIPTOR_ARRAY_NODEALLOC .or. &
    FOR_DESCRIPTOR_ARRAY_CONTIGUOUS, &
    2, &
    dims_info )
p_array = p_array + 1

call c_print_array()
end
```
{ int i,j;
for(i=0;i<5;i++){
  for(j=0;j<5;j++)
    printf("%f ", C_ARRAY[i][j]);
  printf("\n");
}

FOR_GET_FPE

**Run-time Function:** Returns the current settings of floating-point exception flags. This routine can be called from a C or Fortran program.

**Module:** USE IFCORE

**Syntax**

result = FOR_GET_FPE ( )

**Results:**
The result type is INTEGER(4). The return value represents the settings of the current floating-point exception flags. The meanings of the bits are defined in the IFQWIN module file.

To set floating-point exception flags after program initialization, use “FOR_SET_FPE”.

**Example**

USE IFCORE
INTEGER*4 FPE_FLAGS
FPE_FLAGS = FOR_GET_FPE ( )

for_rtl_finish_

**Run-Time Function:** Cleans up the Fortran run-time environment; for example, flushing buffers and closing files. It also issues messages about floating-point exceptions, if any occur.

This routine should be called from a C main program; it is invoked by default from a Fortran main program.

**Syntax**

result = for_rtl_finish_ ( )
Results:
The result is an I/O status value. For information on these status values, see "Using the IOSTAT Value and Fortran Exit Codes" in Building Applications.

To initialize the Fortran run-time environment, use “for_rtl_init_."

Example
Consider the following C code:

```c
int io_status;
int for_rtl_finish_ ( );
io_status = for_rtl_finish_ ( );
```

for_rtl_init_

**Run-Time Subroutine:** Initializes the Fortran run-time environment. It establishes handlers and floating-point exception handling, so Fortran subroutines behave the same as when called from a Fortran main program.

This routine should be called from a C main program; it is invoked by default from a Fortran main program.

**Syntax**

```c
CALL for_rtl_init_ (argcount, actarg)
```

*argcount*

Is a command-line parameter describing the argument count.

*actarg*

Is a command-line parameter describing the actual arguments.

To clean up the Fortran run-time environment, use “for_rtl_finish_."

Example
Consider the following C code:

```c
int argc;
char **argv;
void for_rtl_init_ (int *, char **);
for_rtl_init_ (&argc, argv);
```

FOR_SET_FPE

**Run-time Function:** Sets the floating-point exception flags. This routine can be called from a C or Fortran program.
Module: USE IFCORE

Syntax

result = FOR_SET_FPE (a)

a

Must be of type INTEGER(4). It contains bit flags controlling floating-point exception trapping, reporting, and result handling.

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 IFCORE module file.

To get the current settings of the floating-point exception flags, use “FOR_GET_FPE”.

Example

USE IFCORE

INTEGER*4 OLD_FPE_FLAGS, NEW_FPE_FLAGS

OLD_FPE_FLAGS = FOR_SET_FPE (NEW_FPE_FLAGS)

FOR_SET_REENTRANCY

Run-Time Function: Controls the type of reentrancy protection that the Fortran Run-Time Library (RTL) exhibits. This routine can be called from a C or Fortran program.

Module: USE IFCORE

Syntax

result = FOR_SET_REENTRANCY (mode)

mode

Must be of type INTEGER(4) and contain one of the following options:

• FOR_K_REENTRANCY_NONE
  Tells the Fortran RTL to perform simple locking around critical sections of RTL code. This type of reentrancy should be used when the Fortran RTL will not be reentered due to asynchronous system traps (ASTs) or threads within the application.

• FOR_K_REENTRANCYASYNCH
  Tells the Fortran RTL to perform simple locking and disables ASTs around critical sections of RTL code. This type of reentrancy should be used when the application contains AST handlers that call the Fortran RTL.

• FOR_K_REENTRANCY_THREADED
  Tells the Fortran RTL to perform thread locking. This type of reentrancy should be used in multithreaded applications.
• FOR_K_REENTRANCY_INFO
  Tells the Fortran RTL to return the current reentrancy mode.

Results:
The result type is INTEGER(4). The return value represents the previous setting of the Fortran
Run-Time Library reentrancy mode, unless the argument is FOR_K_REENTRANCY_INFO, in
which case the return value represents the current setting.
You must be using an RTL that supports the level of reentrancy you desire. For example,
FOR_SET_REENTRANCY ignores a request for thread protection
(FOR_K_REENTRANCY_THREADED) if you do not build your program with the thread-safe
RTL.

Example
PROGRAM SETREENT
USE IFCORE

INTEGER*4 MODE
CHARACTER*10 REENT_TXT(3) /'NONE    ',',ASYNCH ','THREADED'/

PRINT*, 'Setting Reentrancy mode to ',REENT_TXT(MODE+1)
MODE = FOR_SET_REENTRANCY(FOR_K_REENTRANCY_NONE)
PRINT*, 'Previous Reentrancy mode was ',REENT_TXT(MODE+1)
MODE = FOR_SET_REENTRANCY(FOR_K_REENTRANCY_INFO)
PRINT*, 'Current Reentrancy mode is ',REENT_TXT(MODE+1)

END

FPUTC

Portability Function: Writes a character to the file specified by a Fortran external unit,
bypassing normal Fortran input/output.
Module: USE IFPORT
Syntax
  result = FPUTC (lunit, char)

lunit
(Input) INTEGER(4). Unit number of a file.
char
(Output) Character*(*) Variable whose value is to be written to the file corresponding to lunit.
Results:
The result type is INTEGER(4). The result is zero if the write was successful; otherwise, an error code, such as:

EINV AL - The specified unit is invalid (either not already open, or an invalid unit number)

If you use WRITE, READ, or any other Fortran I/O statements with lunit, be sure to read "Portability Routines" in Building Applications.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: "I/O Formatting" in the Language Reference, "Files, Devices, and I/O" in Building Applications

Example
use IFPORT
integer*4 lunit, i4
character*26 string
character*1 char1
lunit = 1
open (lunit,file = 'fputc.dat')
do i = 1,26
   char1 = char(123-i)
i4 = fputc(1,char1) !make valid writes
   if (i4.ne.0) iflag = 1
endo
rewind (1)
read (1,'(a)') string
print *, string

FSEEK

Portability Function: Repositions a file specified by a Fortran external unit.
Module: USE IFPORT
Syntax
   result = FSEEK (lunit, offset, from)
lunit
(Input) INTEGER(4). External unit number of a file.
**offset**

(Input) INTEGER(4) or INTEGER(8). Offset in bytes, relative to *from*, that is to be the new location of the file marker.

**from**

(Input) INTEGER(4). A position in the file. It must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Variable</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SEEK_SET</td>
<td>Positions the file relative to the beginning of the file.</td>
</tr>
<tr>
<td>1</td>
<td>SEEK_CUR</td>
<td>Positions the file relative to the current position.</td>
</tr>
<tr>
<td>2</td>
<td>SEEK_END</td>
<td>Positions the file relative to the end of the file.</td>
</tr>
</tbody>
</table>

**Results:**

The result type is INTEGER(4). The result is zero if the repositioning was successful; otherwise, an error code, such as:

EINVAL: The specified unit is invalid (either not already open, or an invalid unit number), or the *from* parameter is invalid.

The file specified in *lunit* must be open.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**Example**

```
USE IFPORT
integer(4) istat, offset, ipos
character ichar
OPEN (unit=1, file='datfile.dat')
offset = 5
ipos = 0
istat=fseek(1, offset, ipos)
if (.NOT. stat) then
  istat=fgetc(1, ichar)
  print *, 'data is ', ichar
end if
```

**FSTAT**

**Portability Function:** Returns detailed information about a file specified by a external unit number.
Module: USE IFPORT

Syntax

\[
\text{result} = \text{FSTAT} (\text{lunit}, \text{statb})
\]

\text{lunit}

(Input) INTEGER(4). External unit number of the file to examine.

\text{statb}

(Output) INTEGER(4) or INTEGER(8). One-dimensional array of size 12; where the system information is stored. The elements of \text{statb} contain the following values:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Values or Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>statb(1)</td>
<td>Device the file resides on</td>
<td>W<em>32,  W</em>64: Always 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(2)</td>
<td>File inode number</td>
<td>W<em>32,  W</em>64: Always 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(3)</td>
<td>Access mode of the file</td>
<td>See the table in Results</td>
</tr>
<tr>
<td>statb(4)</td>
<td>Number of hard links to the file</td>
<td>W<em>32,  W</em>64: Always 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(5)</td>
<td>User ID of owner</td>
<td>W<em>32,  W</em>64: Always 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(6)</td>
<td>Group ID of owner</td>
<td>W<em>32,  W</em>64: Always 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(7)</td>
<td>Raw device the file resides on</td>
<td>W<em>32,  W</em>64: Always 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(8)</td>
<td>Size of the file</td>
<td></td>
</tr>
<tr>
<td>statb(9)</td>
<td>Time when the file was last accessed (^1)</td>
<td>W<em>32,  W</em>64: Only available on non-FAT file systems; undefined on FAT systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(10)</td>
<td>Time when the file was last modified (^1)</td>
<td>W<em>32,  W</em>64: Same as stat(10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(11)</td>
<td>Time of last file status change (^1)</td>
<td>W<em>32,  W</em>64: Same as stat(10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
<tr>
<td>statb(12)</td>
<td>Blocksize for file system I/O operations</td>
<td>W<em>32,  W</em>64: Always 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L*X: System dependent</td>
</tr>
</tbody>
</table>

\(^1\) Times are in the same format returned by the TIME function (number of seconds since 00:00:00 Greenwich mean time, January 1, 1970).
Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, returns an error code equal to EINVAL (lunit is not a valid unit number, or is not open).

The access mode (the third element of statb) is a bitmap consisting of an IOR of the following constants:

<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></td>
</tr>
<tr>
<td>S_IFBLK</td>
<td>O’0060000’</td>
<td>Block special</td>
<td></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 on Windows systems</td>
</tr>
<tr>
<td>S_IFSOCK</td>
<td>O’0140000’</td>
<td>Socket</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_ISUID</td>
<td>O’0040000’</td>
<td>Set user ID on execution</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>O’0002000’</td>
<td>Set group ID on execution</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_ISVTX</td>
<td>O’0001000’</td>
<td>Save swapped text</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_IRWXU</td>
<td>O’0000700’</td>
<td>Owner's file permissions</td>
<td></td>
</tr>
<tr>
<td>S_IRUSR, S_IREAD</td>
<td>O’0000400’</td>
<td>Owner's read permission</td>
<td>Always true on Windows systems</td>
</tr>
<tr>
<td>S_IWUSR, S_IWRITE</td>
<td>O’0000200’</td>
<td>Owner's write permission</td>
<td></td>
</tr>
<tr>
<td>S_IXUSR, S_IEXEC</td>
<td>O’0000100’</td>
<td>Owner's execute permission</td>
<td>Based on file extension (.EXE, .COM, .CMD, or .BAT)</td>
</tr>
<tr>
<td>S_IRWXG</td>
<td>O’0000070’</td>
<td>Group's file permissions</td>
<td>Same as S_IRWXU on Windows systems</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>O’0000040’</td>
<td>Group's read permission</td>
<td>Same as S_IRUSR on Windows systems</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>O’0000020’</td>
<td>Group's write permission</td>
<td>Same as S_IWUSR on Windows systems</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>O’0000010’</td>
<td>Group's execute permission</td>
<td>Same as S_IXUSR on Windows systems</td>
</tr>
<tr>
<td>S_IRWXO</td>
<td>O’0000007’</td>
<td>Other's file permissions</td>
<td>Same as S_IRWXU on Windows systems</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>O’0000004’</td>
<td>Other's read permission</td>
<td>Same as S_IRUSR on Windows systems</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>O’0000002’</td>
<td>Other's write permission</td>
<td>Same as S_IWUSR on Windows systems</td>
</tr>
</tbody>
</table>
STAT returns the same information as FSTAT, but accesses files by name instead of external unit number.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** the INQUIRE statement in the *Language Reference*, “STAT”

**Example**

```fortran
USE IFPORT
integer(4) statarray(12), istat
OPEN (unit=1,file='datfile.dat')
ISTAT = FSTAT (1, statarray)
if (.NOT. istat) then
  print *, statarray
end if
```

## FTELL, FTELLI8

**Portability Functions:** Return the current position of a file.  

**Module:** USE IFPORT

**Syntax**

- `result = FTELL (lunit)`
- `result = FTELLI8 (lunit)`

**lunit**

(Input) INTEGER(4). External unit number of a file.

**Results:**

The result type is INTEGER(4) for FTELL; INTEGER(8) for FTELLI8. The result is the offset, in bytes, from the beginning of the file. A negative value indicates an error, which is the negation of the IERRNO error code. The following is an example of an error code:

**EINVAL:** `lunit` is not a valid unit number, or is not open.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB
**FULLPATHQQ**

**Portability Function:** Returns the full path for a specified file or directory.

**Module:** USE IFPORT

**Syntax**

\[
\text{result} = \text{FULLPATHQQ} (\text{name}, \text{pathbuf})
\]

**name**

(Input) Character(*(*). Item for which you want the full path. Can be the name of a file in the current directory, a relative directory or file name, or a network uniform naming convention (UNC) path.

**pathbuf**

(Output) Character(*(*). Buffer to receive full path of the item specified in \text{name}.

**Results:**

The result type is INTEGER(4). The result is the length of the full path in bytes, or 0 if the function fails (usually for an invalid name).

The length of the full path depends upon 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 (\text{pathbuf}), FULLPATHQQ returns only that portion of the path that fits into the buffer.

Check the length of the path before using the string returned in \text{pathbuf}. 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:

```fortran
USE IFPORT
CHARACTER($MAXPATH) pathbuf
$MAXPATH is a symbolic constant defined in IFQWIN.F90 as 260.
```

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** "SPLITPATHQQ"

**Example**

```fortran
USE IFPORT
USE IFCORE
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 IF
END DO
900  FORMAT (A, A)

GERROR

Run-Time Subroutine: Returns a message for the last error detected by a Fortran run-time routine.

Module: USE IFCORE
Descriptions of the Library Routines

Syntax

CALL GERROR (string)

string
(Output) Character*(*) Message corresponding to the last detected error.

The last detected error does not necessarily correspond to the most recent function call. The compiler resets string only when another error occurs.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “PERROR”, “IERRNO”

Example

USE IFCORE
character*40 errtext
character char1
integer*4 iflag, i4
..!Open unit 1 here
i4=fgetc(1,char1) if (i4) then
  iflag = 1
  Call GERROR (errtext)
  print *, errtext
end if

GETACTIVEQQ

QuickWin Function: Returns the unit number of the currently active child window. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = GETACTIVEQQ ( )

Results:
The result type is INTEGER(4). The result is the unit number of the currently active window. If no child window is active, it returns the parameter QWIN$NOACTIVEWINDOW (defined in IFQWIN.F90).

Compatibility

QUICKWIN GRAPHICS LIB

See Also: “SETACTIVEQQ”, “GETHWNDQQ”, “Using QuickWin” in Building Applications
GETARCINFO

**Graphics Function:** Determines the endpoints (in viewport coordinates) of the most recently drawn arc or pie. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = GETARCINFO (pstart, pend, ppaint)
```

- `pstart` (Output) Derived type `xycoord`. Viewport coordinates of the starting point of the arc.
- `pend` (Output) Derived type `xycoord`. Viewport coordinates of the end point of the arc.
- `ppaint` (Output) Derived type `xycoord`. Viewport coordinates of the point at which the fill begins.

**Results:**

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.

`GETARCINFO` updates the `pstart` and `pend` `xycoord` derived types to contain the endpoints (in viewport coordinates) of the arc drawn by the most recent call to the ARC or PIE functions. The `xycoord` derived type, defined in `IFQWIN.F90`, is:

```fortran
TYPE xycoord
  INTEGER(2) xcoord
  INTEGER(2) ycoord
END TYPE xycoord
```

The returned value in `ppaint` 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 SETCOLORRRGB. Use the new color, along with the coordinates in `ppaint`, as arguments for the FLOODFILLRGB function.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “ARC, ARC_W”, “FLOODFILLRGB, FLOODFILLRGB_W”, “GETCOLORRRGB”, “GRSTATUS”, “PIE, PIE_W”, “SETCOLORRRGB”

**Example**

```fortran
USE IFQWIN
INTEGER(2) status, x1, y1, x2, y2, x3, y3, x4, y4
```
TYPE (xycoord) xystart, xyend, xyfillpt
x1 = 80; y1 = 50
x2 = 240; y2 = 150
x3 = 120; y3 = 80
x4 = 90; y4 = 180
status = ARC(x1, y1, x2, y2, x3, y3, x4, y4)
status = GETARCINFO(xystart, xyend, xyfillpt)
END

GETBKCOLOR

**Graphics Function:** Returns the current background color index for both text and graphics output. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

result = GETBKCOLOR ( )

**Results:**
The result type is INTEGER(4). The result is the current background color index.

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.

**NOTE.** The GETBKCOLOR routine described here is a QuickWin routine. If you are trying to use the Microsoft* Platform SDK version of the GetBkColor routine by including the IFWIN module, you need to specify the routine name as MSFWINSGetBkColor. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.
GETBKCOLORRGB

**Graphics Function:** Returns the current background Red-Green-Blue (RGB) color value for both text and graphics. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = GETBKCOLORRGB()
```

**Results:**

The result type is INTEGER(4). The result is the RGB value of the current background color for both text and graphics.

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:

```
Bit   31 (MSB) 24 23 16 15 8 7 0
RGB  O O O O O O B B B B B G G G G G G G R R R R R R R R
```

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'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 restricted 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.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETCOLORRGB”, “GETTEXTCOLORRGB”, “SETBKCOLORRGB”, “GETBKCOLOR”

Example
! Build as a QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(4) back, fore, oldcolor
INTEGER(2) status, x1, y1, x2, y2
x1 = 80; y1 = 50
x2 = 240; y2 = 150
oldcolor = SETCOLORRGB(Z'FF') ! red
! reverse the screen
back = GETBKCOLORRGB()
fore = GETCOLORRGB()
oldcolor = SETBKCOLORRGB(fore)
oldcolor = SETCOLORRGB(back)
CALL CLEARSCREEN ($GCLEARSCREEN)
status = ELLIPSE($GBORDER, x1, y1, x2, y2)
END

GETC

Portability Function: Reads the next available character from external unit 5, which is normally connected to the console.
Module: USE IFPORT
Syntax
result = GETC (char)

char
(Output) Character*(*) . The 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 an end-of-file was detected.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “GETCHARQQ”, “GETSTROQ”

Example
use IFPORT
character ans, errtxt*40
print *, 'Enter a character: ' 
ISTAT = GETC (ans)
if (ISTAT) then
   call gerror(errtxt)
end if

GETCHARQQ

Run-Time Function: Returns the next keystroke.

Module: USE IFCORE

Syntax
result = GETCHARQQ ( )

Results:
The result type is character with length 1. The result is the character representing 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 Z'00' or Z'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 waits until there is one, and then returns it. Compare this to the function PEEKCHARQQ, which returns .TRUE. if there is a character waiting in the keyboard buffer, and .FALSE. if not. You can use PEEKCHARQQ to determine if GETCHARQQ should be called. This can prevent a program from hanging while GETCHARQQ waits for a keystroke that isn’t there. Note that PEEKCHARQQ is only supported in console applications.

If your application is a QuickWin or Standard Graphics application, you may want to put a call to PASSDIRKEYSSQQ in your program. This will enable the program to get characters that would otherwise be trapped. These extra characters are described in PASSDIRKEYSSQQ.

Note that the GETCHARQQ routine used in a console application is a different routine than the one used in a QuickWin or Standard Graphics application.

The GETCHARQQ used with a console application does not trap characters that are used in QuickWin for a special purpose, such as scrolling. Console applications do not need, and cannot use PASSDIRKEYSSQQ.

**Compatibility**

CONSOLE  STANDARD  GRAPHICS  QUICKWIN  GRAPHICS  LIB

**See Also:** "PASSDIRKEYSSQQ", "PEEKCHARQQ", "GETCHARQQ", "GETSTROQQ", "INCHARQQ", "MBINCHARQQ", "GETC", "FGETC"

**Example**

```fortran
! Program to demonstrate GETCHARQQ
USE IFCore
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
```
GETCOLOR

**Graphics Function:** Returns the current graphics color index. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

```
result = GETCOLOR ( )
```

**Results:**

The result type is INTEGER(2). The result is the current color index, if successful; otherwise, –1.

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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “GETCOLORRGB”, “GETBKCOLOR”, “GETTEXTCOLOR”, “SETCOLOR”

**Example**

```fortran
! Program to demonstrate GETCOLOR
PROGRAM COLORS
USE IFQWIN
INTEGER(2) loop, loop1, status, color
LOGICAL(4) winstat
REAL rnd1, rnd2, xnum, ynum
type (windowconfig) wc
status = SETCOLOR(INT2(0))
```

END IF
IF (key .EQ. 'q' ) THEN
   EXIT
END IF
END DO
900 FORMAT (1X, A, 
901 FORMAT (A)
END
! Color random pixels with 15 different colors
DO loop1 = 1, 15
    color = INT2(MOD(GETCOLOR()+1, 16))
    status = SETCOLOR(color) ! Set to next color
    DO loop = 1, 75
        ! Set color of random spot, normalized to be on screen
        CALL RANDOM(rnd1)
        CALL RANDOM(rnd2)
        winstat = GETWINDOWCONFIG(wc)
        xnum = wc%numxpixels
        ynum = wc%numypixels
        status = &
        SETPIXEL(INT2(rnd1*xnum+1),INT2(rnd2*ynum))
        status = &
        SETPIXEL(INT2(rnd1*xnum),INT2(rnd2*ynum+1))
        status = &
        SETPIXEL(INT2(rnd1*xnum-1),INT2(rnd2*ynum))
        status = &
        SETPIXEL(INT2(rnd1*xnum),INT2(rnd2*ynum-1))
    END DO
END DO
END

GETCOLORRGB

**Graphics Function:** Returns the current graphics color Red-Green-Blue (RGB) value (used by graphics functions such as ARC, ELLIPSE, and FLOODFILLRGB). This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```
result = GETCOLORRGB()
```

**Results:**
The result type is INTEGER(4). The result is the RGB value of the current graphics color. 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 111111 (hex FF) the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF000' full-intensity blue, and Z'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. GETBKCOLORRGB returns the RGB color value of the current background for both text and graphics, set with SETBKCOLORRGB. 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 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 restricted 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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “GETBKCOLORRGB”, “GETTEXTCOLORRGB”, “SETCOLORRGB”, “GETCOLOR”

**Example**

! Build as a QuickWin or Standard Graphics App.

USE IFQWIN
INTEGER(2) numfonts
INTEGER(4) fore, oldcolor

numfonts = INITIALIZEFONTS ( )
oldcolor = SETCOLORRGB(Z'FF')  ! set graphics
                   ! color to red
fore = GETCOLORRGB()
oldcolor = SETBKCOLORRGB(fore) ! set background
! to graphics color

CALL CLEARSCREEN($)GCLEARSCREEN)
oldcolor = SETCOLORRGB (Z'FF0000') ! set graphics
! color to blue

CALL OUTGTEXT("hello, world")
END

GETCONTROLFPQQ

Portability Subroutine: Returns the floating-point processor control word.

Module: USE IFPORT

Syntax

CALL GETCONTROLFPQQ (controlword)

controlword

(Output) INTEGER(2). Floating-point processor control word.

The floating-point control word is a bit flag that controls various modes of the floating-point coprocessor.

The control word can be any of the following constants (defined in IFPORT.F90):

<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$CHOP</td>
<td>Z'0C00'</td>
<td>Trimcet</td>
</tr>
<tr>
<td>FPCW$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>
</tbody>
</table>
The defaults for the floating-point control word 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. Exceptions can be disabled by setting the flags to 1 with SETCONTROLFPQQ.

If an exception is disabled, it does not cause an interrupt when it occurs. Instead, floating-point processes generate an appropriate special value (NaN or signed infinity), but the program continues.

You can find out which exceptions (if any) occurred by calling GETSTATUSFPQQ. If errors on floating-point exceptions are enabled (by clearing the flags to 0 with SETCONTROLFPQQ), the operating system generates an interrupt when the exception occurs. By default, these interrupts cause run-time 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. Most users do not need to change the default settings. For a full discussion of the floating-point control word, exceptions, and error handling, see "The Floating-Point Environment" in Building Applications.

### Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

### See Also:

- "SETCONTROLFPQQ"
- "GETSTATUSFPQQ"
- "SIGNALQQ"
- "CLEARSTATUSFPQQ"

### Example

```
USE IFPORT
INTEGER(2) control
CALL GETCONTROLFPQQ (control)
   !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
```

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Hex value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
GETCURRENTPOSITION, GETCURRENTPOSITION_W

**Graphics Subroutines:** Return the coordinates of the current graphics position. These subroutines are only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
CALL GETCURRENTPOSITION (t)
CALL GETCURRENTPOSITION_W (wt)
```

`t`

(Output) Derived type `xycoord`. Viewport coordinates of current graphics position. The derived type `xycoord` is defined in `IFQWIN.F90` as follows:

```fortran
TYPE xycoord
  INTEGER(2) xcoord ! x-coordinate
  INTEGER(2) ycoord  ! y-coordinate
END TYPE xycoord
```

`wt`

(Output) Derived type `wxycoord`. Window coordinates of current graphics position. The derived type `wxycoord` is defined in `IFQWIN.F90` as follows:

```fortran
TYPE wxycoord
  REAL(8) wx    ! x-coordinate
  REAL(8) wy    ! y-coordinate
END TYPE wxycoord
```

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` or `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.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB
See Also: “LINETO, LINETO_W”, “MOVETO, MOVETO_W”, “OUTGTEXT”, “SETTEXTPOSITION”, “GETTEXTPOSITION”

Example

! Program to demonstrate GETCURRENTPOSITION
USE IFQWIN
TYPE (xycoord) position
INTEGER(2) result
result = LINETO(INT2(300), INT2(200))
CALL GETCURRENTPOSITION( position )
IF (position%xcoord .GT. 50) THEN
   CALL MOVETO(INT2(50), position%ycoord, position)
   WRITE(*,*) "Text unaffected by graphics position"
END IF
result = LINETO(INT2(300), INT2(200))
END

GETCWD

Portability Function: Returns the path of the current working directory.
Module: USE IFPORT
Syntax
   result = GETCWD (dirname)

dirname
(Output) Character (*). Name of the current working directory path, including drive letter.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, an error code.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “GETDRIVEDIRQQ”

Example

use ifport
character*30 dirname
! variable dirname must be long enough to hold entire string
integer(4) istat
ISTAT = GETCWD (dirname)
IF (ISTAT == 0) write *, 'Current directory is ', dirame

**GETDAT**

**Portability Subroutine:** Returns the date.

**Module:** USE IFPORT

**Syntax**

CALL GETDAT (iyr, imon, iday)

*iyr*  
(Output) INTEGER(4) or INTEGER(2). Year (xxxx AD).

*imon*  
(Output) INTEGER(4) or INTEGER(2). Month (1-12).

*iday*  
(Output) INTEGER(4) or INTEGER(2). Day of the month (1-31).

This subroutine is thread-safe.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

**See Also:** “GETTIM”, “SETDAT”, “SETTIM”, “FDATE”, “JDATE4”, “JDATE”

**Example**

! Program to demonstrate GETDAT and GETTIM
USE IFPORT
INTEGER(4) tmpday, tmpmonth, tmpyear
INTEGER(4) tmphour, tmpminute, tmpsecond, tmphund
CHARACTER(1) mer

CALL GETDAT(tmpyear, tmpmonth, tmpday)
CALL GETTIM(tmphour, tmpminute, tmpsecond, tmphund)
IF (tmphour .GT. 12) THEN
  mer = 'p'
  tmphour = tmphour - 12
ELSE
  mer = 'a'
END IF
WRITE (*, 900) tmpmonth, tmpday, tmpyear
900  FORMAT(I2, '/', I2.2, '/', I4.4)
WRITE (*, 901) tmphour, tmpminute, tmpsecond, tmphund, mer
901 FORMAT (I2, ':', I2.2, ':', I2.2, ':', I2.2, ' ', &
A, 'm')
END

GETDRIVEDIRQQ

Portability Function: Returns the path of the current working directory on a specified drive.
Module: USE IFPORT
Syntax
   result = GETDRIVEDIRQQ (drivedir)
 drivedir
(Input; output) Character*(*) On input, drive whose current working directory path is to be
   returned. On output, string containing the current directory on that drive in the form d:\dir.
Results:
The result type is INTEGER(4). The result is the length (in bytes) of the full path of the directory
   on the specified drive. Zero is returned if the path is longer than the size of the character buffer
   drivedir.
You specify the drive from which to return the current working directory by putting the drive letter
   into drivedir before calling GETDRIVEDIRQQ. To make sure you get information about the
   current drive, put the symbolic constant FILE$CURDRIVE (defined in IFPORT.F90) into
   drivedir.
Because drives are identified by a single alphabetic character, GETDRIVEDIRQQ examines only
   the first letter of drivedir. For instance, if drivedir contains the path c:\fps90\bin, GETDRIVEDIRQQ
   (drivedir) returns the current working directory on drive C and disregards the
   rest of the path. The drive letter can be uppercase or lowercase.
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 $MAXPATH ($MAXPATH = 260).
On Linux* systems, the function gets a path only when symbolic constant FILE$CURDRIVE has
   been applied to drivedir.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB
See Also: “CHANGEDRIVEQQ”, “CHANGEDIRQQ”, “GETDRIVESIZEQQ”,
   “GETDRIVESQQ”, “GETLASTERRORQQ”, “SPLITPATHQQ”
Example

! Program to demonstrate GETDRIVEDIRQQ
USE IFPORT
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

Portability Function: Returns the total size of the specified drive and space available on it.
Module: USE IFPORT
Syntax

result = GETDRIVESIZEQQ (drive, total, avail)

*drive
(Input) Character*(*) String containing the letter of the drive to get information about.

*total
(Output) INTEGER(4) or INTEGER(4),DIMENSION(2) or INTEGER(8). Total number of bytes on the drive.

*avail
(Output) INTEGER(4) or INTEGER(4),DIMENSION(2) or INTEGER(8). Number of bytes of available space on the drive.

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
The data types and dimension (if any) specified for the *total and *avail arguments must be the same. Specifying an array of two INTEGER(4) elements, or an INTEGER(8) argument, allows drive sizes larger than 2147483647 to be returned.
If an array of two INTEGER(4) elements is specified, the least-significant 32 bits are returned in the first element, the most-significant 32 bits in the second element. If an INTEGER(4) scalar is specified, the least-significant 32 bits are returned.

Because drives are identified by a single alphabetic character, GETDRIVESIZEQQ examines only the first letter of *drive*. The drive letter can be uppercase or lowercase. You can use the constant FILE$CURDRIVE (defined in IFPORT.F90) to get the size of the current drive.

If GETDRIVESIZEQQ fails, use GETLASTERRORQQ to determine the reason.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "GETLASTERRORQQ", "GETDRIVESQQ", "GETDRIVEDIRQQ", "CHANGEDRIVEQQ", "CHANGEDIRQQ"

**Example**

```fortran
! Program to demonstrate GETDRIVESQQ and GETDRIVESIZEQQ
USE IFPORT
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
```

2-114
GETDRIVESQQ

**Portability Function:** Reports which drives are available to the system.

**Module:** USE IFPORT

**Syntax**

```fortran
result = GETDRIVESQQ ( )
```

**Results:**
The result type is character with length 26. It is the positional character string containing the letters of the drives available in the system.

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* systems, the function returns a string filled with spaces.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

**See Also:** "GETDRIVEDIRQQ", "GETDRIVESIZEQQ", "CHANGEDRIVEQQ"

**Example**
See the example for "GETDRIVESIZEQQ".

GETENV

**Portability Subroutine:** Returns the value of an environment variable.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL GETENV (ename, evalue)
```

*ename*
(Input) Character*(*) . Environment variable to search for.

*evalue*
(Output) Character*(*) . Value found for *ename*. Blank if *ename* is not found.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

**See Also:** "GETENVQQ"
Example

use IFPORT
character*40 libname
CALL GETENV ("LIB", libname)
TYPE *, "The LIB variable points to ", libname

GETENVQQ

Portability Function: Returns the value of an environment variable.
Module: USE IFPORT
Syntax

   result = GETENVQQ (varname, value)

varname
(Input) Character*(*). Name of environment variable.

value
(Output) Character*(*). Value of the specified environment variable, in uppercase.

Results:
The result type is INTEGER(4). The result is the length of the string 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.

GETENVQQ uses the C runtime routine getenv and SETENVQQ uses the C runtime routine _putenv. From the C documentation:

getenv and _putenv use the copy of the environment pointed to by the global variable _environ to access the environment. getenv operates only on the data structures accessible to the run-time library and not on the environment segment created for the process by the operating system.

In a program that uses the main function, _environ is initialized at program startup to settings taken from the operating system’s environment.

Changes made outside the program by the console SET command, for example, SET MY_VAR=ABCDE, will be reflected by GETENVQQ.
GETENVQ0 and SETENVQ0 will not work properly with the Windows* APIs GetEnvironmentVariable and SetEnvironmentVariable.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

See Also: "SETENVQ0", "GETLASTERRORQ0"

Example

!  Program to demonstrate GETENVQ0 and SETENVQ0
USE IFPORT
USE IFCORE
INTEGER(4) lenv, lval
CHARACTER(80) env, val, enval
WRITE (*,900) ' Enter environment variable name to create, &
    modify, or delete: '
     lenv = GETSTRQ0(env)
 IF (lenv .EQ. 0) STOP
 WRITE (*,900) ' Value of variable (ENTER to delete): '
  lval = GETSTRQ0(val)
 IF (lval .EQ. 0) val = ' '
enval = env(1:lenv) // '=' // val(1:lval)
 IF (SETENVQ0(enval)) THEN
   lval = GETENVQ0(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'
 END IF
900 FORMAT (A, \)
END
GETEXCEPTIONPTRSQQ

**Run-Time Function:** Returns a pointer to C run-time exception information pointers appropriate for use in signal handlers established with SIGNALQQ or direct calls to the C rtl signal( ) routine. This function is only available on Windows* systems.

**Module:** USE IFCORE

**Syntax**

```fortran
result = GETEXCEPTIONPTRSQQ ( )
```

**Results:**

The result type is INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. The return value is the address of a data structure whose members are pointers to exception information captured by the C runtime at the time of an exception. This result value can then be used as the EPTR argument to routine TRACEBACKQQ to generate a stack trace from a user-defined handler or to inspect the exception context record directly.

Calling GETEXCEPTIONPTRSQQ is only valid within a user-defined handler that was established with SIGNALQQ or a direct call to the C rtl signal( ) function.

For a full description of exceptions and error handling, see "The Floating-Point Environment" in Building Applications.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

**See Also:** “TRACEBACKQQ”, “GETSTATUSFPQQ”, “CLEARSTATUSFPQQ”, “SETCONTROLFPQQ”, “GETCONTROLFPQQ”, “SIGNALQQ”

**Example**

```fortran
PROGRAM SIGTEST
    USE IFCORE
    ...
    R3 = 0.0E0
    STS = SIGNALQQ(MY_HANDLER)
    ! Cause a divide by zero exception
    R1 = 3.0E0/R3
    ...
END

INTEGER(4) FUNCTION MY_HANDLER(SIGNUM, EXCNUM)
    USE IFCORE
    ...
    EPTRS = GETEXCEPTIONPTRSQQ()
```
CALL TRACEBACKQQ("Application SIGFPE error!",USER.EXIT_CODE=-1,EPTR=EPTRS)

MY_HANDLER = 1
END

A complete working example can be found in the online samples.

**GETEXITQQ**

**QuickWin Function:** Returns the setting for a QuickWin application’s exit behavior. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

\[
\text{result} = \text{GETEXITQQ}()
\]

**Results:**

The result type is INTEGER(4). The result is exit mode with one of the following constants (defined in IFQWIN.F90):

- **QWINEXITPROMPT**: Displays a message box that reads "Program exited with exit status \( n \). Exit Window?", where \( n \) is the exit status from the program.
  
  If you choose Yes, the application closes the window and terminates. If you choose No, the dialog box disappears and you can manipulate the window as usual. You must then close the window manually.

- **QWINEXITNOPERSIST**: Terminates the application without displaying a message box.

- **QWINEXITPERSIST**: Leaves the application open without displaying a message box.

The default for both QuickWin and Console Graphics applications is QWINEXITPROMPT.

**Compatibility**

STANDARD GRAPHICS QUICKWIN.EXE LIB

**See Also:** "SETEXITQQ", "Using QuickWin" in *Building Applications*

**Example**

! Program to demonstrate GETEXITQQ

\[
\begin{align*}
\text{USE IFQWIN} \\
\text{INTEGER } i \\
\text{i} = \text{GETEXITQQ}() \\
\text{SELECT CASE (i)} \\
\text{CASE (QWINEXITPROMPT)}
\end{align*}
\]
GETFILEINFOQQ

Portability Function: Returns information about the specified file. File names can contain wildcards (* and ?).

Module: USE IFPORT

Syntax
result = GETFILEINFOQQ (files, buffer, handle)

files
(Input) Character*(*). Name or pattern of files you are searching for. Can include a full path and wildcards (* and ?).

buffer
(Output) Derived type FILE$INFO or derived type FILE$INFOI8. Information about a file that matches the search criteria in files. The derived type FILE$INFO is defined in IFPORT.F90 as follows:

TYPE FILE$INFO
   INTEGER(4) CREATION       ! CREATION TIME (-1 ON FAT)
   INTEGER(4) LASTWRITE      ! LAST WRITE TO FILE
   INTEGER(4) LASTACCESS    ! LAST ACCESS (-1 ON FAT)
   INTEGER(4) LENGTH        ! LENGTH OF FILE
   INTEGER(4) PERMIT        ! FILE ACCESS MODE
   CHARACTER(255) NAME      ! FILE NAME
END TYPE FILE$INFO

The derived type FILE$INFOI8 is defined in IFPORT.F90 as follows:

TYPE FILE$INFOI8
   INTEGER(4) CREATION       ! CREATION TIME (-1 ON FAT)
   INTEGER(4) LASTWRITE      ! LAST WRITE TO FILE
   INTEGER(4) LASTACCESS    ! LAST ACCESS (-1 ON FAT)
   INTEGER(8) LENGTH        ! LENGTH OF FILE
INTEGER(4) PERMIT       ! FILE ACCESS MODE
CHARACTER(255) NAME     ! FILE NAME
END TYPE FILE$INFO

handle

(Input; output) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors.
Control mechanism. One of the following constants, defined in IFPORT.F90:
• FILE$FIRST – First matching file found.
• FILE$LAST – Previous file was the last valid file.
• FILE$ERROR – No matching file found.

Results:
The result type is INTEGER(4). The result is the nonblank length of the file name if a match was
found, or 0 if no matching files were found.
To get information about one or more files, set the 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 FILEINFO%CREATION, FILEINFO%LASTWRITE, and
FILEINFO%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. FILEINFO%LENGTH contains the length of the file
in bytes. FILEINFO%PERMIT contains a set of bit flags describing access information about the
file as follows:

<table>
<thead>
<tr>
<th>Bit flag</th>
<th>Access information for the 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$SYSTEM</td>
<td>Used by the operating system.</td>
</tr>
</tbody>
</table>
You can use the constant FILE$NORMAL to check that all bit flags are set to 0. If the derived-type element variable FILEINFO%PERMIT is equal to FILE$NORMAL, the file has no special attributes. The variable FILEINFO%NAME contains the short name of the file, not the full path of the file.

If an error occurs, call GETLASTERRORQQ 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, the available memory has been corrupted, or an invalid block exists, indicating that the process making the call was not allocated properly.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

See Also: “SETFILEACCESSQQ”, “SETFILETIMEQQ”, “UNPACKTIMEQQ”

Example
USE IFPORT
USE IFCORE
   CALL SHOWPERMISSION( )
END

! SUBROUTINE to demonstrate GETFILEINFOQQ
SUBROUTINE SHOWPERMISSION( )
USE IFPORT
   CHARACTER(80) files
   INTEGER(KIND=INT_PTR_KIND( )) handle
   INTEGER(4) length
   CHARACTER(5) permit
   TYPE (FILE$INFO) info
   WRITE (*, 900) ' Enter wildcard of files to view: '
   900 FORMAT (A, \)
   length = GETSTRQQ(files)
   handle = FILE$FIRST
   DO WHILE (.TRUE.)
      length = GETFILEINFOQQ(files, info, handle)
IF ((handle .EQ. FILE$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, A5, I9, ' ',A6)
END DO
END SUBROUTINE

GETFILLMASK

Graphics Subroutine:  Returns the current pattern used to fill shapes.  This subroutine is only available on Windows* systems.
Module:  USE IFQWIN
Syntax
  CALL GETFILLMASK (mask)
mask
(Output) INTEGER(1).  One-dimensional array of length 8.
There are 8 bytes in \textit{mask}, and each of the 8 bits in each byte represents a pixel, creating an 8x8 pattern. The first element (byte) of \textit{mask} becomes the top 8 bits of the pattern, and the eighth element (byte) of \textit{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 \textsc{setcolorrgb} or \textsc{setcolor}. The 8-byte mask is replicated over the entire fill area. If no fill mask is set (with \textsc{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 (\textsc{floodfillrgb}, \textsc{pie}, \textsc{ellipse}, \textsc{polygon}, and \textsc{rectangle}).

\textbf{Compatibility}

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

\textbf{See Also:} “\textsc{ellipse, ellipse_w}”, “\textsc{floodfillrgb, floodfillrgb_w}”, “\textsc{pie, pie_w}”, “\textsc{polygon, polygon_w}”, “\textsc{rectangle, rectangle_w}”, “\textsc{setfillmask}”

\textbf{Example}

```fortran
! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(1) style(8). array(8)
INTEGER(2) i
style = 0
style(1) = Z'F'
style(3) = Z'F'
style(5) = Z'F'
style(7) = Z'F'
CALL SETFILLMASK (style)
...
CALL GETFILLMASK (array)
WRITE (*, *) 'Fill mask in bits: '
DO i = 1, 8
   WRITE (*, '(B8)') array(i)
END DO
END
```

\section*{GetFontInfo}

\textbf{Graphics Function:} Returns the current font characteristics. This function is only available on Windows* systems.

\textbf{Module:} USE IFQWIN
Syntax
result = GETFONTINFO (font)

font
(Output) Derived type FONTINFO. Set of characteristics of the current font. The FONTINFO derived type is defined in IFQWIN.F90 as follows:

```fortran
TYPE 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) filename    ! File name including path
  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 TYPE FONTINFO
```

Results:
The result type is INTEGER(2). The result is zero if successful; otherwise, –1.

You must initialize fonts with INITIALIZEFONTS before calling any font-related function, including GETFONTINFO.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETGTEXTEXTENT”, “GETGTEXTROTATION”, “GRSTATUS”, “OUTGTEXT”, “INITIALIZEFONTS”, “SETFONT”, ”Using Fonts from the Graphics Library” in Building Applications

Example
! Build as QuickWin or Standard Graphics
USE IFQWIN
TYPE (FONTINFO) info
INTEGER(2)    numfonts, return, line_spacing
numfonts = INITIALIZEFONTS ( )
return = GETFONTINFO(info)
line_spacing = info%pixheight + 2
END

GETGID

Portability Function: Returns the group ID of the user of a process.
Module: USE IFPORT
Syntax
    result = GETGID ( )
Results:
The result type is INTEGER(4). The result corresponds to the primary group of the user under whose identity the program is running. The result is returned as follows:
• On Windows® systems, this function returns the last subauthority of the security identifier for the process. This is unique on a local machine and unique within a domain for domain accounts.
    Note that on Windows systems, domain accounts and local accounts can overlap.
• On Linux® systems, this function returns the group identity for the current process.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

Example
USE IFPORT
ISTAT = GETGID ( )

GETGTEXTEXTENT

Graphics Function: Returns the width in pixels that would be required to print a given string of text (including any trailing blanks) with OUTGTEXT using the current font. This function is only available on Windows® systems.
Module: USE IFQWIN
Syntax
    result = GETGTEXTEXTENT (text)
text

(Input) Character*(*). Text to be analyzed.

Results:
The result type is INTEGER(2). The result is the width of text in pixels if successful; otherwise, –1 (for example, if fonts have not been initialized with INITIALIZEFONTS).

This function is useful for determining the size of text that uses proportionally spaced fonts. You must initialize fonts with INITIALIZEFONTS before calling any font-related function, including GETGTEXTEXTENT.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETFONTINFO", "OUTGTEXT", "SETFONT", "INITIALIZEFONTS", "GETGTEXTEXTENT"

Example
! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(2) status, pwidth
CHARACTER(80) text
status= INITIALIZEFONTS( )
status= SETFONT('t''Arial''h22w10')
pwidth= GETGTEXTEXTENT('How many pixels wide is this?')
WRITE(*,*) pwidth
END

GETGTEXTEXTENT

Graphics Function: Returns the current orientation of the font text output by OUTGTEXT. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = GETGTEXTEXTENT()

Results:
The result is of type 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.
The orientation for text output with OUTGTEXT is set with SETGTEXTROTATION.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “OUTGTEXT”, “SETFONT”, “SETGTEXTROTATION”

**Example**

! Build as QuickWin or Standard Graphics

USE IFQWIN
INTEGER ang
REAL rang
ang = GETGTEXTROTATION()
rang = FLOAT(ang)/10.0
WRITE(*,*) "Text tilt in degrees is: ", rang
END

**GETHWNDQQ**

**QuickWin Function:** Converts a window unit number into a Windows* handle. This function is only available on Windows systems.

**Module:** USE IFQWIN

**Syntax**

```
result = GETHWNDQQ ( unit )
```

**unit**

(Input) INTEGER(4). The window unit number. If `unit` is set to QWIN$FRAMEWINDOW (defined in IFQWIN.F90), the handle of the frame window is returned.

**Results:**

The result type is INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. The result is a true Windows handle to the window. If `unit` is not open, it returns –1.

**Compatibility**

QUICKWIN GRAPHICS LIB

**See Also:** “GETACTIVEQQ”, “GETUNITQQ”, “SETACTIVEQQ”. "Using QuickWin" in Building Applications
GETIMAGE, GETIMAGE_W

**Graphics Subroutines:** Store the screen image defined by a specified bounding rectangle. These subroutines are only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
CALL GETIMAGE (x1, y1, x2, y2, image)
CALL GETIMAGE_W (wx1, wy1, wx2, wy2, image)
```

- `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.
- `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.
- `image` (Output) INTEGER(1). Array of single-byte integers. Stored image buffer.

GETIMAGE defines the bounding rectangle in viewport-coordinate points \((x1, y1)\) and \((x2, y2)\). 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.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** “IMAGESIZE, IMAGESIZE_W”, “PUTIMAGE, PUTIMAGE_W”

**Example**

```fortran
! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(1), ALLOCATABLE:: buffer (:)
INTEGER(2) status, x, y, error
INTEGER(4) imsize
x = 50
```
\texttt{y = 30}
\texttt{status = ELLIPSE \($\text{GFILLINTERIOR, INT2(x-15), \&}
\text{INT2(y-15), INT2(x+15), INT2(y+15)}\))}
\texttt{imsise = IMAGESIZE \(\text{INT2(x-16), INT2(y-16), \&}
\text{INT2(x+16), INT2(y+16)}\))}
\texttt{ALLOCATE(buffer (imsise), STAT = error)}
\texttt{IF (error .NE. 0) THEN}
\texttt{\quad STOP 'ERROR: Insufficient memory'}
\texttt{END IF}
\texttt{CALL GETIMAGE \(\text{INT2(x-16), INT2(y-16), \&}
\text{INT2(x+16), INT2(y+16), buffer}\)}
\texttt{END}

\textbf{GETLASTERROR}

\textbf{Portability Function: } Returns the last error set.
\textbf{Module: } \texttt{USE IFPORT}
\textbf{Syntax}
\begin{verbatim}
result = GETLASTERROR()
\end{verbatim}
\textbf{Results:}
The result type is INTEGER(4). The result is the integer corresponding to the last run-time error value that was set.

For example, if you use an ERR= specifier in an I/O statement, your program will not abort if an error occurs. GETLASTERROR provides a way to determine what the error condition was, with a better degree of certainty than just examining \texttt{errno}. Your application can then take appropriate action based upon the error number.

\textbf{Compatibility}
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

\textbf{GETLASTERRORQQ}

\textbf{Portability Function: } Returns the last error set by a run-time procedure.
\textbf{Module: } \texttt{USE IFPORT}
\textbf{Syntax}
\begin{verbatim}
result = GETLASTERRORQQ()
\end{verbatim}
Results:
The result type is INTEGER(4). The result is the most recent error code generated by a run-time procedure.

Library functions that return a logical or integer value sometimes also provide an error code that identifies the cause of errors. GETLASTERRORQ retrieves the most recent error message. The error constants are in IFORT.F90. The following table shows some library routines and the errors each routine produces:

<table>
<thead>
<tr>
<th>Library routine</th>
<th>Errors produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEEPQQ</td>
<td>no error</td>
</tr>
<tr>
<td>BSEARCHQQ</td>
<td>ERR$INVAL</td>
</tr>
<tr>
<td>CHANGEDIRQQ</td>
<td>ERR$NOMEM, ERR$NOENT</td>
</tr>
<tr>
<td>CHANGEDRIVEQQ</td>
<td>ERR$INVAL, ERR$NOENT</td>
</tr>
<tr>
<td>COMMITQQ</td>
<td>ERR$BADF</td>
</tr>
<tr>
<td>DELDIRQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$NOENT</td>
</tr>
<tr>
<td>DELFILESQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$NOENT, ERR$INVAL</td>
</tr>
<tr>
<td>FINDFILEQQ</td>
<td>ERR$NOMEM, ERR$NOENT</td>
</tr>
<tr>
<td>FULLPATHQQ</td>
<td>ERR$NOMEM, ERR$INVAL</td>
</tr>
<tr>
<td>GETCHARQQ</td>
<td>no error</td>
</tr>
<tr>
<td>GETDRIVEDIRQQ</td>
<td>ERR$NOMEM, ERR$RANGE</td>
</tr>
<tr>
<td>GETDRIVESIZEQQ</td>
<td>ERR$INVAL, ERR$NOENT</td>
</tr>
<tr>
<td>GETDRIVESQQ</td>
<td>no error</td>
</tr>
<tr>
<td>GETENVQQ</td>
<td>ERR$NOMEM, ERR$NOENT</td>
</tr>
<tr>
<td>GETFILEINFOQQ</td>
<td>ERR$NOMEM, ERR$NOENT, ERR$INVAL</td>
</tr>
<tr>
<td>GETLASTERRORQQ</td>
<td>no error</td>
</tr>
<tr>
<td>GETSTROQ</td>
<td>no error</td>
</tr>
<tr>
<td>MADEIRQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$EXIST, ERR$NOENT</td>
</tr>
<tr>
<td>PACKTIMEQQ</td>
<td>no error</td>
</tr>
<tr>
<td>PEEKCHARQQ</td>
<td>no error</td>
</tr>
<tr>
<td>RENAMEFILEQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$NOENT, ERR$XDEV</td>
</tr>
<tr>
<td>RANQQ</td>
<td>ERR$NOMEM, ERR$2BIG, ERR$INVAL, ERR$NOENT, ERR$NOEXEC</td>
</tr>
<tr>
<td>SETERRORMODEQQ</td>
<td>no error</td>
</tr>
<tr>
<td>SETENVQQ</td>
<td>ERR$NOMEM, ERR$INVAL</td>
</tr>
<tr>
<td>SETFILEACCESSQQ</td>
<td>ERR$NOMEM, ERR$INVAL, ERR$ACCES</td>
</tr>
</tbody>
</table>
GETLINESTYLE

**Graphics Function:** Returns the current graphics line style. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

result = GETLINESTYLE ( )

**Results:**

The result type is INTEGER(2). The result is the current line style.

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 Z'FFFF' (a solid line). A dashed line can be represented by Z'FF00' (long dashes) or Z'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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

<table>
<thead>
<tr>
<th>Library routine</th>
<th>Errors produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETFILETIMEQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$INVAL, ERR$MFILE, ERR$NOENT</td>
</tr>
<tr>
<td>SLEEPQQ</td>
<td>no error</td>
</tr>
<tr>
<td>SORTQQ</td>
<td>ERR$INVAL</td>
</tr>
<tr>
<td>SPLITPATHQQ</td>
<td>ERR$NOMEM, ERR$INVAL</td>
</tr>
<tr>
<td>SYSTEMQQ</td>
<td>ERR$NOMEM, ERR$2BIG, ERR$NOENT, ERR$NOEXEC</td>
</tr>
<tr>
<td>UNPACKTIMEQQ</td>
<td>no error</td>
</tr>
</tbody>
</table>

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** “LINETO, LINETO_W”, “POLYGON, POLYGON_W”, “RECTANGLE, RECTANGLE_W”, “SETCOLORRGB”, “SETFILLMASK”, “SETLINESTYLE”, “SETWRITEMODE”
Example

! Build as Graphics
USE IFQWIN
INTEGER(2) lstyle
lstyle = GETLINESTYLE()
WRITE (*, 100) lstyle, lstyle
100 FORMAT (1X, 'Line mask in Hex ', Z4, ' and binary ', B16)
END

GETLOG

Portability Subroutine: Returns the user’s login name.
Module: USE IFPORT
Syntax
CALL GETLOG (name)
name
(Output) Character*(*) User’s login name.
The login name must be less than or equal to 64 characters. If the login name is longer than 64 characters, it is truncated. The actual parameter corresponding to name 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 is truncated.
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.
If the login name cannot be determined, all blanks are returned.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
Example
use IFPORT
character*20 username
CALL GETLOG (username)
print *, "You logged in as ", username

GETPHYSCOORD

Graphics Subroutine: Translates viewport coordinates to physical coordinates. This subroutine is only available on Windows* systems.
Module: USE IFQWIN

Syntax

CALL GETPHYSCOORD (x, y, i)

x, y

(Input) INTEGER(2). Viewport coordinates to be translated to physical coordinates.

i

(Output) Derived type xycoord. Physical coordinates of the input viewport position. The

xycoord derived type is defined in IFQWIN.F90 as follows:

TYPE xycoord
  INTEGER(2) xcoord ! x-coordinate
  INTEGER(2) ycoord ! y-coordinate
END TYPE xycoord

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. For a
more complete discussion of coordinate systems, see "Understanding Coordinate Systems" in
Building Applications.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETVIEWCOORD, GETVIEWCOORD_W”, “GETWINDOWCOORD”,
“SETCLIPRGN”, “SETVIEWPORT”

Example

! Program to demonstrate GETPHYSCOORD, GETVIEWCOORD,
! and GETWINDOWCOORD. Build as QuickWin or Standard
! Graphics
USE IFQWIN
TYPE (xycoord) viewxy, physxy
TYPE (wxycoord) windxy
CALL SETVIEWPORT(INT2(80), INT2(50), &
  INT2(240), INT2(150))
! Get viewport equivalent of point (100, 90)
CALL GETVIEWCOORD (INT2(100), INT2(90), viewxy)
! Get physical equivalent of viewport coordinates
CALL GETPHYSCOORD (viewxy%xcoord, viewxy%ycoord, &
  physxy)
Descriptions of the Library Routines

GETWINDOWCOORD (viewxy%xcoord, viewxy%ycoord, & windxy)

! Write viewport coordinates
WRITE (*,*) viewxy%xcoord, viewxy%ycoord
! Write physical coordinates
WRITE (*,*) physxy%xcoord, physxy%ycoord
! Write window coordinates
WRITE (*,*) windxy%wx, windxy%wy
END

GETPID

Portability Function: Returns the process ID of the current process.
Module: USE IFPORT
Syntax
result = GETPID ( )
Results:
The result type is INTEGER(4). The result is the process ID number of the current process.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
Example
USE IFPORT
INTEGER(4) istat
istat = GETPID()

GETPIXEL, GETPIXEL_W

Graphics Functions: Return the color index of the pixel at a specified location. These functions are only available on Windows® systems.
Module: USE IFQWIN
Syntax
result = GETPIXEL (x, y)
result = GETPIXEL_W (wx, wy)
x, y
(Input) INTEGER(2). Viewport coordinates for pixel position.

wx, wy
(Input) REAL(8). Window coordinates for pixel position.

Results:
The result type is INTEGER(2). The result is the pixel color index if successful; otherwise, –1 (for example, if the pixel lies outside the clipping region).

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.

NOTE. The GETPIXEL routine described here is a QuickWin routine. If you are trying to use the Microsoft® Platform SDK version of the GetPixel routine by including the IFWIN module, you need to specify the routine name as MSFWIN$GetPixel. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETPIXELRGB, GETPIXELRGB_W", "GRSTATUS", "REMAPPALETTERGB", "REMAPALLPALETTERGB", "SETCOLOR", "GETPIECELS", "SETPIXEL, SETPIXEL_W"

GETPIXELRGB, GETPIXELRGB_W

Graphics Functions: Return the Red-Green-Blue (RGB) color value of the pixel at a specified location. These functions are only available on Windows® systems.

Module: USE IFQWIN

Syntax
result = GETPIXELRGB (x, y)
result = GETPIXELRGB_W (wx, wy)

x, y
(Input) INTEGER(2). Viewport coordinates for pixel position.
wx, wy

(Input) REAL(8). Window coordinates for pixel position.

Results:
The result type is INTEGER(4). The result is the pixel’s current RGB color value.
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:

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td></td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
</tbody>
</table>
```

Larger numbers correspond to stronger color intensity with binary 11111111 (hex Z'FF') the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'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 restricted 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.

Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: “SETPIXELRGB, SETPIXELRGB_W”, “GETPIXELRGB”, “SETPIXELRGBW”, “GETPIXEL, GETPIXEL_W”

Example
! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(4) pixcolor, rseed
INTEGER(2) status
REAL rnd1, rnd2
LOGICAL(4) winstat
TYPE (windowconfig) wc
CALL GETTIM (status, status, status, INT2(rseed))
CALL SEED (rseed)
CALL RANDOM (rnd1)
CALL RANDOM (rnd2)
! Get the color index of a random pixel, normalized to
! be in the window. Then set current color to that
! pixel color.
winstat = GETWINDOWCONFIG(wc)
xnum = wc%numxpixels
ynum = wc%numypixels
pixcolor = GETPIXELRGB( INT2( rnd1*xnum ), INT2( rnd2*ynum ))
status = SETCOLORRGB (pixcolor)
END

GETPIXELS

Graphics Subroutine: Returns the color indexes of multiple pixels. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL GETPIXELS (n, x, y, color)

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.

color
(Output) INTEGER(2). Array to be filled with the color indexes of the pixels at x and y.

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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:**  "GETPIXELSRGB", "SETPIXELSRGB", "GETPIXEL, GETPIXEL_W", "SETPIXELS"

### GETPIXELSRGB

**Graphics Subroutine:** Returns the Red-Green-Blue (RGB) color values of multiple pixels. This subroutine is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
CALL GETPIXELSRGB (n, x, y, color)
```

\( n \)

(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 \( y \).

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 values 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 11111111 (hex Z'FF') the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'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 restricted 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.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “SETPIXELSRGB”, “GETPIXELRGB, GETPIXELRGB_W”, “GETPIXELS”, “SETPIXELS”

Example

! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(4) color(50), result
INTEGER(2) x(50), y(50), status
TYPE(xycoord) pos
result = SETCOLORRGB(Z'FF')
CALL MOVETO(INT2(0), INT2(0), pos)
status = LINETO(INT2(100), INT2(200))

! Get 50 pixels at line 30 in viewport
DO i = 1, 50
\[ x(i) = i-1 \]
\[ y(i) = 30 \]

END DO
CALL GETPIXELSRGB(300, x, y, color)
! Move down 30 pixels and redisplay pixels
DO i = 1, 50
    \[ y(i) = y(i) + 30 \]
END DO
CALL SETPIXELSRGB (50, x, y, color)
END

**GETPOS, GETPOSI8**

*Portability Functions:* Return the current position of a file.

*Module:* USE IFPORT

*Syntax*

\[
\text{result} = \text{GETPOS (} \text{lunit} \text{)} \\
\text{result} = \text{GETPOSI8 (} \text{lunit} \text{)}
\]

\[ \text{lunit} \]

(Input) INTEGER(4). External unit number of a file. The value must be in the range 0 to 100 and the file must be connected.

*Results:*

The result type is INTEGER(4) for GETPOS; INTEGER(8) for GETPOSI8. The result is the offset, in bytes, from the beginning of the file. If an error occurs, the result value is –1 and the following error code is returned in \[ \text{errno} \]:

- \[ \text{EINVAL} \]: \text{lunit} is not a valid unit number, or is not open.

These functions are equivalent to “FTELL, FTELL16”.

*Compatibility*

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**GETSTATUSFPQQ**

*Portability Subroutine:* Returns the floating-point processor status word.

*Module:* USE IFPORT
Syntax

CALL GETSTATUSFPQQ (status)

status

(Output) INTEGER(2). Floating-point processor status word.

The floating-point status word shows whether various floating-point exception conditions have occurred. Intel® Visual Fortran initially clears (sets to 0) all status flags, but after an exception occurs it does not reset the 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$DENORMAL</td>
<td>Z'0002'</td>
<td>A denormal (very small number) occurred</td>
</tr>
<tr>
<td>FPSW$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 logical comparison on the status word returned by GETSTATUSFPQQ to determine which of the six floating-point exceptions listed in the table 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 SETCONTROLFPQQ. You can use 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 (NaN or signed infinity), but the program continues. You can find out which exceptions (if any) occurred by calling GETSTATUSFPQQ.

If errors on floating-point exceptions are enabled (by clearing the flags to 0 with SETCONTROLFPQQ), the operating system generates an interrupt when the exception occurs. By default, these interrupts cause run-time errors, but you can capture the interrupts with SIGNALQQ and branch to your own error-handling routines.

For a full discussion of the floating-point status word, exceptions, and error handling, see "The Floating-Point Environment" in *Building Applications.*
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

See Also: “SETCONTROLFPQQ”, “GETCONTROLFPQQ”, “SIGNALQQ”, “CLEARSTATUSFPQQ”

Example
! Program to demonstrate GETSTATUSFPQQ
USE IFPORT
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.'
END IF
END

GETSTRQQ

Run-time Function: Reads a character string from the keyboard using buffered input.
Module: USE IFCORE

Syntax
   result = GETSTRQQ (buffer)

buffer
(Output) Character*(*). Character string returned from keyboard, padded on the right with blanks.

Results:
The result type is INTEGER(4). The result is the number of characters placed in buffer.
The function does not complete until you press Return or Enter.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “GETCHARQQ”, “PEEKCHARQQ”, the READ statement in the Language Reference

Example
! Program to demonstrate GETSTRQQ
USE IFCORE
USE IFPORT
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

GETTEXTCOLOR

Graphics Function: Returns the current text color index. This function is only available on Windows® systems.
Module: USE IFQWIN
Syntax
   result = GETTEXTCOLOR( )
Results:
The result type is INTEGER(2). It is the current text color index.
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.
NOTE. The GETTEXTCOLOR routine described here is a QuickWin routine. If you are trying to use the Microsoft® Platform SDK version of the GetTextColor routine by including the IFWIN module, you need to specify the routine name as MSFWINS$GetTextColor. For more information, see “Special Naming Convention for Certain QuickWin and Win32 Graphics Routines” in Building Applications.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “OUTTEXT”, “REMAPPALETTERGB”, “SETCOLOR”, “SETTEXTCOLOR”

GETTEXTCOLORRGB

Graphics Function: Returns the Red-Green-Blue (RGB) value of the current text color (used with OUTTEXT, WRITE and PRINT). This function is only available on Windows® systems.

Module: USE IFQWIN

Syntax

result = GETTEXTCOLORRGB ( )

Results:

The result type is INTEGER(4). It is the RGB value of the current text color.

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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O O O O</td>
<td>B B B B B B B B</td>
<td>G G G G G G G G</td>
<td>R R R R R R R R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Larger numbers correspond to stronger color intensity with binary (hex Z'FF') the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'FFFF00' 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 restricted 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.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “SETTEXTCOLORRGB”, “GETBKCOLORRGB”, “GETCOLORRGB”, “GETTEXTCOLOR”

Example
! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(4) oldtextc, oldbackc, temp
TYPE (rccoord) curpos
! Save color settings
oldtextc = GETTEXTCOLORRGB()
oldbackc = GETBKCOLORRGB()
CALL CLEARSCREEN( $GCLEARSCREEN )
! Reset colors
temp = SETTEXTCOLORRGB(Z'00FFFF')  ! full red + full green
                                              ! = full yellow text
       temp = SETBKCOLORRGB(Z'FF0000')    ! blue background
CALL SETTEXTPOSITION( INT2(4), INT2(15), curpos)
CALL OUTTEXT( 'Hello, world')
! Restore colors
temp = SETTEXTCOLORRGB(oldtextc)
temp = SETBKCOLORRGB(oldbackc)
GETTEXTPOSITION

Graphics Subroutine: Returns the current text position. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL GETTEXTPOSITION (t)

 t (Output) Derived type rcoord. Current text position. The derived type rcoord is defined in IFQWIN.F90 as follows:

TYPE rcoord
  INTEGER(2) row   ! Row coordinate
  INTEGER(2) col   ! Column coordinate
END TYPE rcoord

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.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "SETTEXTPOSITION", "OUTTEXT", "GETCURRENTPOSITION", "GETCURRENTPOSITION_W", "SETTEXTWINDOW", the WRITE statement in the Language Reference

Example

! Build as QuickWin or Standard Graphics
  USE IFQWIN
  TYPE (rcoord) textpos
  CALL GETTEXTPOSITION (textpos)
  END
GETTEXTWINDOW

Graphics Subroutine: Finds the boundaries of the current text window. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL GETTEXTWINDOW (r1, c1, r2, c2)

r1, c1  
(Output) INTEGER(2). Row and column coordinates for upper-left corner of the text window.

r2, c2  
(Output) INTEGER(2). Row and column coordinates for lower-right corner of the text window.

Output from OUTTEXT and WRITE is limited to the text window. By default, this is the entire window, unless the text window is redefined by SETTEXTWINDOW.

The window defined by SETTEXTWINDOW has no effect on output from OUTGTEXT.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETTEXTPOSITION", "OUTTEXT", "SCROLLTEXTWINDOW", "SETTEXTPOSITION", "SETTEXTWINDOW", "WRAPON", the WRITE statement in the Language Reference

Example

! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(2) top, left, bottom, right
DO i = 1, 10
   WRITE(*,*) "Hello, world"
END DO

! Save text window position
CALL GETTEXTWINDOW (top, left, bottom, right)

! Scroll text window down seven lines
CALL SCROLLTEXTWINDOW (INT2(-7))

! Restore text window
CALL SETTEXTWINDOW (top, left, bottom, right)
WRITE(*,*) "At beginning again"
END
GETTIM

**Portability Subroutine:** Returns the time.

**Module:** USE IFPORT

**Syntax**

CALL GETTIM (ihr, imin, isec, i100th)

*ihr*

(Output) INTEGER(4) or INTEGER(2). Hour (0-23).

*imin*

(Output) INTEGER(4) or INTEGER(2). Minute (0-59).

*isec*

(Output) INTEGER(4) or INTEGER(2). Second (0-59).

*i100th*

(Output) INTEGER(4) or INTEGER(2). Hundredths of a second (0-99).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

**See Also:** “GETDAT”, “SETDAT”, “SETTIM”

**Example**

See the example in “GETDAT”.

GETTIMEOFDAY

**Portability Subroutine:** Returns seconds and microseconds since 00:00 Jan 1, 1970.

**Module:** USE IFPORT

**Syntax**

CALL GETTIMEOFDAY (ret, err)

*ret*

(Output) INTEGER(4). One-dimensional array with 2 elements used to contain numeric time data. The elements of *ret* are returned as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret(1)</td>
<td>Seconds</td>
</tr>
<tr>
<td>ret(2)</td>
<td>Microseconds</td>
</tr>
</tbody>
</table>
err
(Output) INTEGER(4).
If an error occurs, err contains a value equal to –1 and array ret contains zeros.
On Windows* systems, this subroutine has millisecond precision, and the last three digits of the returned value are not significant.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

GETUID
Portability Function: Returns the user ID of the calling process.
Module: USE IFPORT
Syntax
result = GETUID( )
Results:
The result type is INTEGER(4). The result corresponds to the user identity under which the program is running. The result is returned as follows:
• On Windows* systems, this function returns the last subauthority of the security identifier for the process. This is unique on a local machine and unique within a domain for domain accounts.
  Note that on Windows systems, domain accounts and local accounts can overlap.
• On Linux* systems, this function returns the user identity for the current process.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
Example
USE IFPORT
integer(4) istat
ISTAT = GETUID( )

GETUNITQQ
QuickWin Function: Returns the unit number corresponding to the specified Windows* handle. This function is only available on Windows systems.
Module: USE IFQWIN
Syntax

\[
\text{result} = \text{GETUNITQQ} (\text{whandle})
\]

\text{whandle}

(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. The Windows handle to the window; this is a unique ID.

Results:

The result type is INTEGER(4). The result is the unit number corresponding to the specified Windows handle. If \text{whandle} does not exist, it returns –1.

This routine is the inverse of GETHWNDQQ.

Compatibility

QUICKWIN GRAPHICS LIB

See Also: “GETHWNDQQ”, "Using QuickWin" in Building Applications

GETVIEWCOORD, GETVIEWCOORD_W

Graphics Subroutines: Translate physical coordinates or window coordinates to viewport coordinates. These subroutines are only available on Windows* systems.

Module: USE IFQWIN

Syntax

\[
\begin{align*}
\text{CALL GETVIEWCOORD} & (x, y, t) \\
\text{CALL GETVIEWCOORD_W} & (wx, wy, wt)
\end{align*}
\]

\text{x, y}

(Input) INTEGER(2). Physical coordinates to be converted to viewport coordinates.

\text{t}

(Output) Derived type \text{xycoord}. Viewport coordinates. The \text{xycoord} derived type is defined in IFQWIN.F90 as follows:

\text{TYPE xycoord}

\text{INTEGER(2) xcoord} \quad ! x-coordinate

\text{INTEGER(2) ycoord} \quad ! y-coordinate

\text{END TYPE xycoord}

\text{wx, wy}

(Input) REAL(8). Window coordinates to be converted to viewport coordinates.
wt
(Output) Derived type wxycoord. Window coordinates. The derived type wxycoord is defined in IFQWIN.F90 as follows:

```fortran
TYPE wxycoord
  REAL(8) wx    ! x-coordinate
  REAL(8) wy    ! y-coordinate
END TYPE wxycoord
```

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. For a more complete discussion of coordinate systems, see "Understanding Coordinate Systems" in Building Applications.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

*See Also:* ["GETPHYSCOORD", "GETWINDOWCOORD"]

**Example**

See the example program in ["GETPHYSCOORD"].

---

**GETWINDOWCONFIG**

**QuickWin Function:** Returns the properties of the current window. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = GETWINDOWCONFIG (wc)
```

**wc**

(Output) Derived type windowconfig. Contains window properties. The windowconfig derived type is defined in IFQWIN.F90 as follows:

```fortran
TYPE 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
```

---

2-152
INTEGER(4) fontsize           ! Size of default font. Set to
! QWIN$EXTENDFONT when specifying
! extended attributes, in which
! case extendfontsize sets the
! font size

CHARACTER(80) title               ! The window title

INTEGER(2) bitsperpixel           ! The number of bits per pixel

INTEGER(2) numvideopages          ! Unused

INTEGER(2) mode                   ! Controls scrolling mode

INTEGER(2) adapter                ! Unused

INTEGER(2) monitor                ! Unused

INTEGER(2) memory                 ! Unused

INTEGER(2) environment            ! Unused

! The next three parameters provide extended font attributes.
!

CHARACTER(32) extendfontname      ! The name of the desired font

INTEGER(4) extendfontsize         ! Takes the same values as fontsize,
! when fontsize is set to
! QWIN$EXTENDFONT

INTEGER(4) extendfontattributes   ! Font attributes such as bold
! and italic

END TYPE windowconfig

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE. (for example, if there is no active child window).

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 GETWSIZEQQ. In this case, 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 SETWINDOWCONFIG.

Note that the bitsperpixel field in the windowconfig derived type is an output field only, while the other fields return output values to GETWINDOWCONFIG and accept input values from SETWINDOWCONFIG.

Compatibility

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: "GETWSIZEQQ", "SETWINDOWCONFIG", "SETACTIVEQQ", "Using QuickWin" in Building Applications

Example

!Build as QuickWin or Standard Graphics App.
USE IFQWIN
LOGICAL(4) status
TYPE (windowconfig) wc
status = GETWINDOWCONFIG(wc)
IF(wc%numtextrows .LT. 10) THEN
   wc%numtextrows = 10
   status = SETWINDOWCONFIG(wc)
   IF(.NOT. status ) THEN  ! if setwindowconfig error
      status = SETWINDOWCONFIG(wc)   ! reset
      ! setwindowconfig with corrected values
      status = GETWINDOWCONFIG(wc)
   END IF
   IF(wc%numtextrows .NE. 10) THEN
      WRITE(*,*) 'Error: Cannot increase text rows to 10'
   END IF
END IF
END IF
END IF
END

GETWINDOWCOORD

Graphics Subroutine: Converts viewport coordinates to window coordinates. This subroutine is only available on Windows® systems.

Module: USE IFQWIN
CALL GETWINDOWCOORD (x, y, wt)

x, y
(Input) INTEGER(2). Viewport coordinates to be converted to window coordinates.

wt
(Output) Derived type wxycoord. Window coordinates. The wxycoord derived type is defined in IFQWIN.F90 as follows:

```fortran
TYPE wxycoord
   REAL(8) wx   ! x-coordinate
   REAL(8) wy   ! y-coordinate
END TYPE wxycoord
```

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. For a more complete discussion of coordinate systems, see "Understanding Coordinate Systems" in Building Applications.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETCURRENTPOSITION, GETCURRENTPOSITION_W", "GETPHYSCOORD", "GETVIEWCOORD, GETVIEWCOORD_W", "MOVETO, MOVETO_W", "SETVIEWPORT", "SETWINDOW"

Example

See the example program in "GETPHYSCOORD".

---

**GETWRITEMODE**

**Graphics Function:** Returns the current logical write mode, which is used when drawing lines with the LINETO, POLYGON, and RECTANGLE functions. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = GETWRITEMODE ()
```

**Results:**

The result type is INTEGER(2). The result is the current write mode. 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.
• \$GRESET – 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.

The default value is \$GPSET. These constants are defined in \texttt{IFQWIN.F90}.

The write mode is set with \texttt{SETWRITEMODE}.

\textbf{Compatibility}

\texttt{STANDARD GRAPHICS QUICKWIN GRAPHICS LIB}

\textbf{See Also:} \texttt{"SETWRITEMODE", \"SETLINESTYLE", \"LINETO, LINETO W", \"POLYGON, POLYGON W", \"PUTIMAGE, PUTIMAGE W", \"RECTANGLE, RECTANGLE W", \"SETCOLORRGB", \"SETFILLMASK", \"GRSTATUS"}

\textbf{Example}

\begin{verbatim}
! Build as QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(2) mode
mode = GETWRITEMODE()
END
\end{verbatim}

\textbf{GETWSIZEQQ}

\begin{quote}
\textbf{QuickWin Function:} Returns the size and position of a window. This function is only available on Windows* systems.
\end{quote}

\begin{quote}
\textbf{Module:} USE IFQWIN
\end{quote}

\begin{quote}
\textbf{Syntax}
\begin{verbatim}
result = GETWSIZEQQ (unit, ireq, wino)
\end{verbatim}
\end{quote}

\begin{quote}
\textit{unit}
\end{quote}

(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 \textit{unit} to the symbolic constant \texttt{QWIN$FRAMEWINDOW}, defined in \texttt{IFQWIN.F90}.

\texttt{GETWSIZEQQ}
ireq
(Input) INTEGER(4). Specifies what information is obtained. The following symbolic constants, defined in IFQWIN.F90, are available:

- QWIN$SIZEMAX – Gets information about the maximum window size.
- QWIN$SIZECURR – Gets information about the current window size.

winfo
(Output) Derived type 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). The derived type qwinfo is defined in IFQWIN.F90 as follows:

```fortran
TYPE QWINFO
  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 TYPE QWINFO
```

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero.
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.

Compatibility
QUICKWIN GRAPHICS LIB

See Also: "GETWINDOWCONFIG", "SETWSIZEQQ", "Using QuickWin" in Building Applications

GMTIME

Portability Subroutine: Returns the Greenwich mean time in an array of time elements.
Module: USE IFPORT
Syntax

    CALL GMTIME (stime, tarray)

stime

(Input) INTEGER(4). Numeric time data to be formatted. Number of seconds since 00:00:00 Greenwich mean time, January 1, 1970.

tarray

(Output) INTEGER(4). One-dimensional array with 9 elements used to contain numeric time data. The elements of tarray are returned as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tarray(1)</td>
<td>Seconds (0-59)</td>
</tr>
<tr>
<td>tarray(2)</td>
<td>Minutes (0-59)</td>
</tr>
<tr>
<td>tarray(3)</td>
<td>Hours (0-23)</td>
</tr>
<tr>
<td>tarray(4)</td>
<td>Day of month (1-31)</td>
</tr>
<tr>
<td>tarray(5)</td>
<td>Month (0-11)</td>
</tr>
<tr>
<td>tarray(6)</td>
<td>Number of years since 1900</td>
</tr>
<tr>
<td>tarray(7)</td>
<td>Day of week (0-6, where 0 is Sunday)</td>
</tr>
<tr>
<td>tarray(8)</td>
<td>Day of year (0-365)</td>
</tr>
<tr>
<td>tarray(9)</td>
<td>Daylight saving flag (0 if standard time, 1 if daylight saving time)</td>
</tr>
</tbody>
</table>

CAUTION. This subroutine may cause problems with the year 2000. Use the DATE_AND_TIME intrinsic subroutine instead (see the Language Reference).

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

Example

use IFPORT
integer(4) stime, timearray(9)
! initialize stime to number of seconds since
! 00:00:00 GMT January 1, 1970
stime = time()
CALL GMTIME (stime, timearray)
print *, timearray
GRSTATUS

Graphics Function: Returns the status of the most recently used graphics routine. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax
result = GRSTATUS()

Results:
The result type is INTEGER(2). The result is the status of the most recently used graphics function.

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.

The following symbolic constants are defined in the IFQWIN.F90 module file for use with GRSTATUS:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GRFILEWRITEERROR</td>
<td>Error writing bitmap file</td>
</tr>
<tr>
<td>$GRFILEOPENERROR</td>
<td>Error opening bitmap file</td>
</tr>
<tr>
<td>$GRIMAGEREADERERROR</td>
<td>Error reading image</td>
</tr>
<tr>
<td>$GRBITMAPDISPLAYERROR</td>
<td>Error displaying bitmap</td>
</tr>
<tr>
<td>$GRBITMAPTOOLARGE</td>
<td>Bitmap too large</td>
</tr>
<tr>
<td>$GRIMPROPERBITMAPFORMAT</td>
<td>Improper format for bitmap file</td>
</tr>
<tr>
<td>$GRFILEREADERROR</td>
<td>Error reading file</td>
</tr>
<tr>
<td>$GRNOMETAPFILE</td>
<td>No bitmap file</td>
</tr>
<tr>
<td>$GRINVALIDIMAGEBUFFER</td>
<td>Image buffer data inconsistent</td>
</tr>
<tr>
<td>$GRINSUFFICIENTMEMORY</td>
<td>Not enough memory to allocate buffer or to complete a fill operation</td>
</tr>
<tr>
<td>$GRINVALIDPARAMETER</td>
<td>One or more parameters invalid</td>
</tr>
<tr>
<td>$GRMODENOTSUPPORTED</td>
<td>Requested video mode not supported</td>
</tr>
<tr>
<td>$GRERROR</td>
<td>Graphics error</td>
</tr>
<tr>
<td>$GROK</td>
<td>Success</td>
</tr>
<tr>
<td>$GRNOOUTPUT</td>
<td>No action taken</td>
</tr>
</tbody>
</table>
After a graphics call, compare the return value of GRSTATUS to $GROK. to determine if an error has occurred. For example:

```
IF ( GRSTATUS .LT. $GROK ) THEN
  ! Code to handle graphics error goes here
ENDIF
```

The following routines cannot give errors, and they all set GRSTATUS to $GROK:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Possible GRSTATUS error codes</th>
<th>Possible GRSTATUS warning codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAYCURSOR</td>
<td>GETCOLORRGB</td>
<td>GETTEXTWINDOW</td>
</tr>
<tr>
<td>GETBKCOLOR</td>
<td>GETTEXTCOLOR</td>
<td>OUTTEXT</td>
</tr>
<tr>
<td>GETBKCOLORRGB</td>
<td>GETTEXTCOLORRGB</td>
<td>WRAPON</td>
</tr>
<tr>
<td>GETCOLOR</td>
<td>GETTEXTPosition</td>
<td></td>
</tr>
</tbody>
</table>

The following table lists some other routines with the error or warning messages they produce for GRSTATUS:

<table>
<thead>
<tr>
<th>Function</th>
<th>Possible GRSTATUS error codes</th>
<th>Possible GRSTATUS warning codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC, ARC_W</td>
<td>$GRINVALIDPARAMETER</td>
<td>$GRNOOUTPUT</td>
</tr>
<tr>
<td>CLEARSCREEN</td>
<td>$GRINVALIDPARAMETER</td>
<td>$GRNOOUTPUT</td>
</tr>
<tr>
<td>ELLIPSE, ELLIPSE_W</td>
<td>$GRINVALIDPARAMETER, $GRINSUFFICIENTMEMORY</td>
<td>$GRNOOUTPUT</td>
</tr>
<tr>
<td>FLOODFILLRGB</td>
<td>$GRINVALIDPARAMETER, $GRINSUFFICIENTMEMORY</td>
<td>$GRNOOUTPUT</td>
</tr>
<tr>
<td>GETARCINFO</td>
<td>$GRERROR</td>
<td></td>
</tr>
<tr>
<td>GETFILLMASK</td>
<td>$GRERROR, $GRINVALIDPARAMETER</td>
<td></td>
</tr>
<tr>
<td>GETFONTINFO</td>
<td>$GRERROR</td>
<td></td>
</tr>
<tr>
<td>GETGTEXTEXTENT</td>
<td>$GRERROR</td>
<td></td>
</tr>
<tr>
<td>GETIMAGE</td>
<td>$GRINSUFFICIENTMEMORY</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>GETPIXEL</td>
<td>$GRBITMAPTOOLARGE</td>
<td></td>
</tr>
<tr>
<td>GETPIXELRGB</td>
<td>$GRBITMAPTOOLARGE</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Possible GRSTATUS error codes</td>
<td>Possible GRSTATUS warning codes</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>LINETO, LINETO_W</td>
<td>$GRFILEOPENERROR, $GRNBITMAPFILE, $GRALEREADERROR, $GRIMPROPERBITMAPFORMAT, $GRBITMAPTOOLARGE, $GRIMAGEREADERROR</td>
<td>$GRNNOOUTPUT, $GRCLIPPED</td>
</tr>
<tr>
<td>LOADIMAGE</td>
<td>$GRFILEOPENERROR, $GRNBITMAPFILE, $GRALEREADERROR, $GRIMPROPERBITMAPFORMAT, $GRBITMAPTOOLARGE, $GRIMAGEREADERROR</td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>OUTGTEXT</td>
<td></td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>PIE, PIE_W</td>
<td>$GRINVALIDPARAMETER, $GRINSUFFICIENTMEMORY</td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>POLYGON, POLYGON_W</td>
<td>$GRINVALIDPARAMETER, $GRINSUFFICIENTMEMORY</td>
<td>$GRNNOOUTPUT, $GRCLIPPED</td>
</tr>
<tr>
<td>PUTIMAGE, PUTIMAGE_W</td>
<td>$GRERROR, $GRINVALIDPARAMETER, $GRINVALIDIMAGEBUFFER, $GRBITMAPDISPLAYERROR</td>
<td>$GRPARAMETERALTERED, $GRNNOOUTPUT</td>
</tr>
<tr>
<td>RECTANGLE, RECTANGLE_W</td>
<td>$GRINVALIDPARAMETER, $GRINSUFFICIENTMEMORY</td>
<td>$GRNNOOUTPUT, $GRCLIPPED</td>
</tr>
<tr>
<td>REMAPPALETTERGB</td>
<td>$GRERROR, $GRINVALIDPARAMETER</td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>SAVEIMAGE</td>
<td>$GRFILEOPENERROR</td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>SCROLLTEXTWINDOW</td>
<td></td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>SETBKCOLOR</td>
<td>$GRINVALIDPARAMETER</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETBKCOLORRGB</td>
<td>$GRINVALIDPARAMETER</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETCLIPRGN</td>
<td>$GRNNOOUTPUT</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETCOLOR</td>
<td>$GRNNOOUTPUT</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETCOLORRGB</td>
<td>$GRNNOOUTPUT</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETFONT</td>
<td>$GRERROR, $GRINSUFFICIENTMEMORY</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETPIXEL, SETPIXEL_W</td>
<td>$GRNNOOUTPUT</td>
<td>$GRPARAMETERALTERED</td>
</tr>
<tr>
<td>SETPIXELRGB, SETPIXELRGB_W</td>
<td></td>
<td>$GRNNOOUTPUT</td>
</tr>
<tr>
<td>SETTEXTCOLOR</td>
<td>$GRPARAMETERALTERED</td>
<td></td>
</tr>
<tr>
<td>SETTEXTCOLORRGB</td>
<td>$GRPARAMETERALTERED</td>
<td></td>
</tr>
</tbody>
</table>
HOSTNAM

**Portability Function:** Returns the current host computer name. This function can also be specified as HOSTNM.

**Module:** USE IFPORT

**Syntax**

```fortran
result = HOSTNAM (name)
```

* name

(Output) Character*(*) . Name of the current host. Should be at least as long as MAX_HOSTNAM_LENGTH + 1. MAX_HOSTNAM_LENGTH is defined in the IFPORT module.

**Results:**

The result type is INTEGER(4). 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**Example**

```fortran
use IFPORT
character(MAX_HOSTNAM_LENGTH + 1) hostname
integer(4) istat
ISTAT = HOSTNAM (hostname)
```
IDATE

**Portability Subroutine:** Returns the month, day, and year of the current system.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL IDATE (i, j, k)
-or-
CALL IDATE (iarray)
```

- *i* (Output) INTEGER(4). The current system month.
- *j* (Output) INTEGER(4). The current system day.
- *k* (Output) INTEGER(4). The current system year as an offset from 1900.
- *iarray* (Output) INTEGER(4). 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. The year is greater than or equal to 1969 and is returned as 2 digits.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

**See Also:** "DATE", "GETDAT", the DATE_AND_TIME and IDATE subroutines in the Language Reference

**Example**

```fortran
use IFPORT
integer(4) imonth, iday, iyear, datarray(3)
! If the date is July 11, 1999:
   CALL IDATE (IMONTH, IDAY, IYEAR)
! sets IMONTH to 7, IDAY to 11 and IYEAR to 99.
   CALL IDATE (DATARRAY)
! datarray is (/11,7,99/)
```

**NOTE.** IDATE is an intrinsic procedure unless you specify USE IFPORT.
IDATE4

**Portability Subroutine:** Returns the month, day, and year of the current system.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL IDATE4 (i, j, k)
~or~
CALL IDATE4 (iarray)
```

- **i** (Output) INTEGER(4). The current system month.
- **j** (Output) INTEGER(4). The current system day.
- **k** (Output) INTEGER(4). The current system year as an offset from 1900.
- **iarray** (Output) INTEGER(4). A 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. The year is returned 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

IDFLOAT

**Portability Function:** Converts an INTEGER(4) variable to double-precision real type.

**Module:** USE IFPORT

**Syntax**

```fortran
result = IDFLOAT (i)
```

- **i** (Input) Must be of type INTEGER(4).

**Results:**

The result type is double-precision real (REAL(8) or REAL*8).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
IEEE_FLAGS

Portability Function: Gets, sets or clears IEEE® flags for rounding direction and precision as well as queries or controls exception status. This function provides easy access to the modes and status required to use the features of IEEE Standard 754-1985 arithmetic in a Fortran program.

Module: USE IFPORT

Syntax

\[ \text{result} = \text{IEEE_FLAGS} (\text{action, mode, in, out}) \]

action

(Input) Character*(*). One of the following literal values: 'GET', 'SET', 'CLEAR', or 'CLEARALL'.

mode

(Input) Character*(*). One of the following literal values: 'direction', 'precision', or 'exception'. The value 'precision' is only allowed on IA-32 systems.

in

(Input) Character*(*). One of the following literal values: 'inexact', 'division', 'underflow', 'overflow', 'invalid', 'all', 'common', 'nearest', 'tozero', 'negative', 'positive', 'extended', 'double', 'single', or ' ', which represents an unused (null) value.

out

(Output) Must be at least CHARACTER*9. One of the literal values listed for in.

The descriptions for the values allowed for in and out can be summarized as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'nearest'</td>
<td>Rounding direction flags</td>
</tr>
<tr>
<td>'tozero'</td>
<td></td>
</tr>
<tr>
<td>'negative'</td>
<td></td>
</tr>
<tr>
<td>'positive'</td>
<td></td>
</tr>
<tr>
<td>'single'</td>
<td>Rounding precision flags</td>
</tr>
<tr>
<td>'double'</td>
<td></td>
</tr>
<tr>
<td>'extended'</td>
<td></td>
</tr>
<tr>
<td>'inexact'</td>
<td>Math exception flags</td>
</tr>
<tr>
<td>'underflow'</td>
<td></td>
</tr>
<tr>
<td>'overflow'</td>
<td></td>
</tr>
<tr>
<td>'division'</td>
<td></td>
</tr>
<tr>
<td>'invalid'</td>
<td></td>
</tr>
</tbody>
</table>

See Also: the intrinsic function DFLOAT in the Language Reference
The values for `in` and `out` depend on the `action` and `mode` they are used with. The interaction of the parameters can be summarized as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'all'</td>
<td>All five math exception flags above</td>
</tr>
<tr>
<td>'common'</td>
<td>The math exception flags: 'invalid', 'division', 'overflow', and 'underflow'</td>
</tr>
</tbody>
</table>

The values for `in` and `out` depend on the `action` and `mode` they are used with. The interaction of the parameters can be summarized as follows:

<table>
<thead>
<tr>
<th><code>action</code></th>
<th><code>mode</code></th>
<th><code>in</code></th>
<th><code>out</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>'direction'</td>
<td>Null (&quot;&quot;&quot;)</td>
<td>One of 'nearest', 'tozero', 'negative', or 'positive'</td>
<td>Tests rounding direction settings. Returns the current setting, or 'not available'.</td>
</tr>
<tr>
<td>GET</td>
<td>'exception'</td>
<td>Null (&quot;&quot;&quot;)</td>
<td>One of 'inexact', 'division', 'underflow', 'overflow', 'invalid', 'all', or 'common'</td>
<td>Tests math exception settings. Returns the current setting, or 0.</td>
</tr>
<tr>
<td>GET</td>
<td>'precision'</td>
<td>Null (&quot;&quot;&quot;)</td>
<td>One of 'single ', 'double ', or 'extended'</td>
<td>Tests rounding precision settings. Returns the current setting, or 'not available'.</td>
</tr>
<tr>
<td>SET</td>
<td>'direction'</td>
<td>One of 'nearest', 'tozero', 'negative', or 'positive'</td>
<td>Null (&quot;&quot;&quot;)</td>
<td>Sets a rounding direction.</td>
</tr>
<tr>
<td>SET</td>
<td>'exception'</td>
<td>One of 'inexact', 'division', 'underflow', 'overflow', 'invalid', 'all', or 'common'</td>
<td>Null (&quot;&quot;&quot;)</td>
<td>Sets a floating-point math exception.</td>
</tr>
<tr>
<td>SET</td>
<td>'precision'</td>
<td>One of 'single ', 'double ', or 'extended'</td>
<td>Null (&quot;&quot;&quot;)</td>
<td>Sets a rounding precision.</td>
</tr>
</tbody>
</table>
IEEE_FLAGS is an elemental, integer-valued function that sets IEEE flags for GET, SET, CLEAR, or CLEARALL procedures. It lets you control rounding direction and rounding precision, query exception status, and control exception enabling or disabling by using the SET or CLEAR procedures, respectively.

The flags information is returned as a set of 1-bit flags.

### Examples

The following example gets the highest priority exception that has a flag raised. It passes the input argument in as a null string:

```
USE IFPORT
INTEGER*4 iflag
CHARACTER*9 out
iflag = ieee_flags('get', 'exception', '', out)
PRINT *, out, ' flag raised'
```

The following example sets the rounding direction to round toward zero, unless the hardware does not support directed rounding modes:

```
USE IFPORT
INTEGER*4 iflag
```
CHARACTER*1 mode, out, in
iflag = ieee_flags('set', 'direction', 'tozero', out)

The following example sets the rounding direction to the default ('nearest'):

USE IFPORT
INTEGER*4 iflag
CHARACTER*1 out, in
iflag = ieee_flags('clear', 'direction', '', '')

The following example clears all exceptions:

USE IFPORT
INTEGER*4 iflag
CHARACTER*10 out
iflag = ieee_flags('clear', 'exception', 'all', '')

The following example restores default direction and precision settings, and sets all exception flags to 0:

USE IFPORT
INTEGER*4 iflag
CHARACTER*10 mode, out, in
iflag = ieee_flags('clearall', '', '', '')

The following example detects an underflow exception:

USE IFPORT
CHARACTER*20 out, in
excep_detect = ieee_flags('get', 'exception', 'underflow', out)
if (out .eq.'underflow') stop 'underflow'

IEEE_HANDLER

Portability Function: Establishes a handler for IEEE exceptions.
Module: USE IFPORT
Syntax
result = IEEE_HANDLER (action, exception, handler)

action
(Input) Character*(*)$. One of the following IEEE actions: 'GET', 'SET', or 'CLEAR'. For more details on these actions, see IEEE_FLAGS.
Descriptions of the Library Routines

exception

(Input) Character*(*)

One of the following IEEE exception flags: 'inexact', 'underflow', 'overflow', 'division', 'invalid', 'all' (which equals the previous five flags), or 'common' (which equals 'invalid', 'overflow', 'underflow', and 'division'). The flags 'all' or 'common' should only be used for actions SET or CLEAR. For more details on these flags, see IEEE_FLAGS.

handler

(Input) The address of an external signal-handling routine.

Results:

The result type is INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors.

The result is 0 if successful; otherwise, 1.

IEEE_HANDLER calls a signal-handling routine to establish a handler for IEEE exceptions. It also enables an FPU trap corresponding to the required exception.

The state of the FPU is not defined in the handler routine. When the FPU trap occurs, the program invokes the handler routine. After the handler routine is executed, the program terminates.

The handler routine gets the exception code in the SIGINFO argument. SIGNO is the number of the system signal. The meaning of the SIGINFO constants appear in the following table (defined in the IFPORT module):

<table>
<thead>
<tr>
<th>FPE$INVALID</th>
<th>Invalid operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPE$ZERODIVIDE</td>
<td>Divide-by-zero</td>
</tr>
<tr>
<td>FPE$OVERFLOW</td>
<td>Numeric overflow</td>
</tr>
<tr>
<td>FPE$UNDERFLOW</td>
<td>Numeric underflow</td>
</tr>
<tr>
<td>FPE$INEXACT</td>
<td>Inexact result (precision)</td>
</tr>
</tbody>
</table>

'GET' returns the location of the current handler routine for exception cast to an INTEGER.

See Also: "IEEE_FLAGS"

Example

The following example creates a handler routine and sets it to trap divide-by-zero:

```
PROGRAM TEST_IEEE
  REAL :: X, Y, Z
  CALL FPE_SETUP
  X = 0.
  Y = 1.
  Z = Y / X
```

---

2-169
END PROGRAM

SUBROUTINE FPE_SETUP
USE IFPORT
IMPLICIT NONE
INTERFACE
  SUBROUTINE FPE_HANDLER(SIGNO, SIGINFO)
    INTEGER(4), INTENT(IN) :: SIGNO, SIGINFO
  END SUBROUTINE
END INTERFACE

INTEGER IR
IR = IEEE_HANDLER('set','division',FPE_HANDLER)
END SUBROUTINE FPE_SETUP

SUBROUTINE FPE_HANDLER(SIG, CODE)
USE IFPORT
IMPLICIT NONE
INTEGER SIG, CODE
IF(CODE.EQ.FPE$ZERODIVIDE) PRINT *, 'Occurred divide by zero.'
CALL ABORT
END SUBROUTINE FPE_HANDLER

IERRNO

Portability Function: Returns the number of the last detected error from any routines in the IFPORT module that return error codes.

Module: USE IFPORT

Syntax
result = IERRNO ( )

Results:
The result type is INTEGER(4). The result value is the last error code from any portability routines that return error codes. These error codes are analogous to errno on a Linux* system. The module IFPORT.F90 provides parameter definitions for the following errno names (typically found in errno.h on Linux systems):

<table>
<thead>
<tr>
<th>Symbolic name</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPERM</td>
<td>1</td>
<td>Insufficient permission for operation</td>
</tr>
<tr>
<td>ENOENT</td>
<td>2</td>
<td>No such file or directory</td>
</tr>
</tbody>
</table>
The value returned by IERRNO is updated only when an error occurs. For example, if an error occurs on a GETLOG call and then two CHMOD calls succeed, a subsequent call to IERRNO returns the error for the GETLOG call.

Examine IERRNO immediately after returning from a portability routine. Other Fortran routines, as well as any Windows* APIs, can also change the error code to an undefined value. IERRNO is set on a per thread basis.

**Compatibility**

<table>
<thead>
<tr>
<th>Console</th>
<th>Standard Graphics</th>
<th>QuickWin Graphics</th>
<th>Windows</th>
<th>DLL</th>
<th>Lib</th>
</tr>
</thead>
</table>

**Example**

```fortran
USE IFPORT
CHARACTER*20 username
INTEGER(4) ierrval
ierrval=0 !initialize return value
CALL GETLOG(username)
IF (IERRNO( ) == ierrval) then
   print *, 'User name is ',username
   exit
ELSE
   ierrval = ierrno()
   print *, 'Error is ',ierrval
END IF
```

**IFLOATI, IFLOATJ**

**Portability Functions:** Convert an integer to single-precision real type.
Module: USE IFPORT
Syntax
   result = IFLOATI (i)
   result = IFLOATJ (j)
i
   (Input) Must be of type INTEGER(2).

j
   (Input) Must be of type INTEGER(4).

Results:
The result type is single-precision real (REAL(4) or REAL*4).

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: the DFLOAT intrinsic function in the Language Reference

IMAGESIZE, IMAGESIZE_W

Graphics Functions: Return 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. These functions are only available on Windows* systems.

Module: USE IFQWIN
Syntax
   result = IMAGESIZE (x1, y1, x2, y2)
   result = IMAGESIZE_W (wx1, wy1, wx2, wy2)
x1, y1
   (Input) INTEGER(2). Viewport coordinates for upper-left corner of image.
x2, y2
   (Input) INTEGER(2). Viewport coordinates for lower-right corner of image.
wx1, wy1
   (Input) REAL(8). Window coordinates for upper-left corner of image.
wx2, wy2
   (Input) REAL(8). Window coordinates for lower-right corner of image.

Results:
The result type is INTEGER(4). The result is the storage size of an image in bytes.
IMAGESIZE defines the bounding rectangle in viewport-coordinate points \((x1, \ y1)\) and \((x2, \ y2)\).

IMAGESIZE_W defines the bounding rectangle in window-coordinate points \((wx1, \ wy1)\) and \((wx2, \ wy2)\).

IMAGESIZE_W defines the bounding rectangle in terms of window-coordinate points \((wx1, \ wy1)\) and \((wx2, \ wy2)\).

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** "GETIMAGE, GETIMAGE_W", "GRSTATUS", "PUTIMAGE, PUTIMAGE_W"

**Example**

See the example in "GETIMAGE, GETIMAGE_W".

---

**INCHARQQ**

**QuickWin Function:** Reads a single character input from the keyboard and returns the ASCII value of that character without any buffering. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

\[
\text{result} = \text{INCHARQQ}() 
\]

**Results:**

The result type is INTEGER(2). The result is the ASCII key code.

The keystroke is read from the child window that currently has the focus. You must call INCHARQQ before the keystroke is made (INCHARQQ does not read the keyboard buffer). This function does not echo its input. For function keys, INCHARQQ returns 0xE0 as the upper 8 bits, and the ASCII code as the lower 8 bits.

For direction keys, INCHARQQ returns 0xF0 as the upper 8 bits, and the ASCII code as the lower 8 bits. To allow direction keys to be read, you must use the PASSDIRKEYSQQ function. The escape characters (the upper 8 bits) are different from those of GETCHARQQ. Note that console applications do not need, and cannot use PASSDIRKEYSQQ.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** "GETCHARQQ", "MBINCHARQQ", "GETC", "PASSDIRKEYSQQ", the READ statement in the Language Reference, "Using QuickWin" in Building Applications

**Example**

use IFQWIN
integer*4 res
integer*2 exchar
character*1 ch, ch1

Print *, "Type X to exit, S to scroll, D to pass Direction keys"
123 continue
exchar = incharqq()
! check for escapes
! 0xE0 0x?? is a function key
! 0xF0 0x?? is a direction key
ch = char(rshift(exchar,8) .and. Z'00FF')
ch1= char(exchar .and. Z'00FF')
if (ichar(ch) .eq. 224) then
  print *, "function key = ",ichar(ch), " ",ichar(ch1)," ",ch1
  goto 123
endif
if (ichar(ch) .eq. 240) then
  print *, "direction key = ",ichar(ch), " ",ichar(ch1)," ",ch1
  goto 123
endif
print *, "other key  = ",ichar(ch), " ",ichar(ch1)," ",ch1
if(ch1 .eq. 'S') then
  res = passdirkeysqq(.false.)
  print *, "Entering Scroll mode"
endif
if(ch1 .eq. 'D')  then
  res = passdirkeysqq(.true.)
  print *, "Entering Direction keys mode"
endif
if(ch1 .ne. 'X')
  go to 123
end

**INITIALIZEFONTS**

**Graphics Function:** Initializes Windows* fonts. This function is only available on Windows* systems.

**Module:** USE IFQWIN
Syntax

result = INITIALIZEFONTS ( )

Results:
The result type is INTEGER(2). The result is the number of fonts initialized.

All fonts on Windows systems 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 more information, see "Using Fonts from the Graphics Library" in Building Applications.

The font functions affect the output of OUTGTEXT only. They do not affect other Fortran I/O functions (such as WRITE) or graphics output functions (such as OUTTEXT).

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.

Compatibility

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: "SETFONT", "OUTGTEXT", "Using QuickWin" in Building Applications

Example

! build as a QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(2) numfonts
numfonts = INITIALIZEFONTS() WRITE (*,*) numfonts
END
If no user-defined INITIAL SETTINGS function is supplied, QuickWin calls a predefined INITIAL SETTINGS routine to control the default frame window and menu appearance. You do not need to call INITIAL SETTINGS if you define it, since it will be called automatically during initialization.

For more information, see "Program Control of Menus" in Building Applications.

Compatibility

QUICKWIN GRAPHICS WINDOWS LIB

See Also: “APPENDMENU”, “INSERTMENU”, “DELETEMENU”, “SETWSIZE”, "Using QuickWin" in Building Applications

INMAX

Portability Function: Returns the maximum positive value for an integer.

Module: USE IFPORT

Syntax

result = INMAX (i)

i

(Input) INTEGER(4).

Results:
The result type is INTEGER(4). The result is the maximum 4-byte signed integer value for the argument.

INQFOCUS

QuickWin Function: Determines which window has the focus. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = INQFOCUS (unit)

unit

(Output) INTEGER(4). Unit number of the window that has the I/O focus.

Results:
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.
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:

```fortran
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.

**Compatibility**

QUICKWIN GRAPHICS LIB

**See Also:** "FOCUSQQ", "Using QuickWin" in *Building Applications*

### INSERTMENUQQ

**QuickWin Function:** Inserts a menu item into a QuickWin menu and registers its callback routine. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = INSERTMENUQQ (menuID, itemID, flag, text, routine)
```

**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.

**flag**

(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*(*) Menu item name. Must be a null-terminated C string, for example, words of text’C’.

**routine**

(Input) EXERNAL. Callback subroutine that is 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. You can assign the following predefined routines to menus:

- WINPRINT – Prints the program.
- WINSAVE – Saves the program.
- WINEXIT – Terminates the program.
- WINSELECTTEXT – Selects text from the current window.
- WINSELECTGRAPHICS – Selects graphics from the current window.
- WINSELECTALL – Selects the entire contents of the current window.
- WININPUT – Brings to the top the child window requesting input and makes it 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.

**Results:**
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE.

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.

For more information on customizing QuickWin menus, see "Using QuickWin" in *Building Applications*.

**Compatibility**
QUICKWIN GRAPHICS LIB

**See Also:** “APPENDMENUQQ”, “DELETEMENUQQ”, “MODIFYMENUFLAGSQQ”, “MODIFYMENUROUTINEQQ”, “MODIFYMENUSTRINGQQ”

**Example**
! build as a QuickWin App.
USE IFQWIN
LOGICAL(4) status
! insert new item into Menu 5 (Window)
status= INSERTMENUQQ(5, 5, $MENUCHECKED, ‘New Item’C, &
WINSTATUS)

! insert new menu in position 2
status= INSERTMENUQQ(2, 0, $MENUENABLED, ‘New Menu’C, &
WINSAVE)
END

**INTC**

**Portability Function:** Converts an INTEGER(4) argument to INTEGER(2) type.
Module: USE IFPORT
Syntax
result = INTC (i)
i
(Input) INTEGER(4). A value or expression.

Results:
The result type is INTEGER(2). The result is the value of i with type INTEGER(2). Overflow is ignored.

INTEGERTORGB

QuickWin Subroutine: Converts an RGB color value into its red, green, and blue components. This subroutine is only available on Windows* systems.
Module: USE IFQWIN
Syntax
CALL INTEGERTORGB (rgb, red, green, blue)
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.
INTEGERTORGB separates the four-byte RGB color value into the three components as follows:

<table>
<thead>
<tr>
<th>Bit 31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O O O</td>
<td>B B B B B B</td>
<td>G G G G G G</td>
<td>G G G G G G</td>
<td>R R R R R R</td>
<td>R R R R R R</td>
<td></td>
</tr>
</tbody>
</table>

Compatibility
QUICKWIN GRAPHICS WINDOWS LIB
See Also: “RGBTOINTEGER”, “GETCOLORRGB”, “GETBKCOLORRGB”,
“GETPIXELRGB, GETPIXELRGB_W”, “GETPIXELSRGB”, “GETTEXTCOLORRGB”.
“Using QuickWin” in Building Applications

Example
! build as a QuickWin App.
USE IFQWIN
INTEGER(4) r, g, b
CALL INTEGERTORGB(2456, r, g, b)
write(*,*) r, g, b
END

IPXFARGC

POSIX Function: Returns the index of the last command-line argument.
Module: USE IFPOSIX
Syntax
result = IPXFARGC ( )

Results:
The result type is INTEGER(4). The result value is 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.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “PXFGETARG”

IPXFCONST

POSIX Function: Returns the value associated with a constant defined in the C POSIX standard.
Module: USE IFPOSIX
Syntax
result = IPXFCONST (constname)
constname
(Input) Character. The name of a C POSIX standard constant.
Results:
The result type is INTEGER(4). If constname corresponds to a defined constant in the C POSIX standard, the result value is the integer that is associated with the constant. Otherwise, the result value is −1.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
See Also: “PXFGETARG”, “PXFCNST”

IPXFLENTRIM

POSIX Function: Returns the index of the last non-blank character in an input string.
Module: USE IFPOSIX
Syntax
   result = IPXFLENTRIM (string)
string
(Input) Character. A character string.
Results:
The result type is INTEGER(4). The result value is the index of the last non-blank character in the input argument string, or zero if all characters in string are blank characters.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

IPXFWEXITSTATUS

POSIX Function: Returns the exit code of a child process. This function is only available on Linux* systems.
Module: USE IFPOSIX
Syntax
   result = IPXFWEXITSTATUS (istat)
istat
(Input) INTEGER(4). The value of output argument istat from PXFWAIT or PXFWAITPID.
Results:
The result type is INTEGER(4). The result is the low-order eight bits of the output argument of PXFWAIT or PXFWAITPID.
The IPXFWEXITSTATUS function should only be used if PXFWEIXITED returns TRUE.

See Also: “PXFWAIT”, “PXFWAITPID”, “PXFWEIXITED”

Example

```fortran
program t1
use ifposix
integer(4) ipid, istat, ierror, ipid_ret, istat_ret
  print *," the child process will be born"
  call PXFFORK(IPID, IERROR)
  call PXFGETPID(IPID_RET,IERROR)
  if(IPID.EQ.0) then
    print *," I am a child process"
    print *," My child's pid is", IPID_RET
    call PXFGETPPID(IPID_RET,IERROR)
    print *," The pid of my parent is",IPID_RET
    print *," Now I have exited with code 0xABCD"
    call PXFEXIT(Z'ABCD')
  else
    print *," I am a parent process"
    print *," My parent pid is ", IPID_RET
    print *," I am creating the process with pid", IPID
    print *," Now I am waiting for the end of the child process"
    call PXFWAIT(ISTAT, IPID_RET, IERROR)
    print *," The child with pid ", IPID_RET," has exited"
    if( PXFWEIXITED(ISTAT) ) then
      print *, " The child exited normally"
      istat_ret = IPXFWEXITSTATUS(ISTAT)
      print 10," The low byte of the child exit code is", istat_ret
    end if
  end if
10 FORMAT (A,Z)
end program
```

**IPXFWSSTOPSIG**

**POSIX Function:** Returns the number of the signal that caused a child process to stop. This function is only available on Linux* systems.

**Module:** USE IFPOSIX
Syntax

result = IPXFWSTOPSIG (istat)

istat
(Input) INTEGER(4). The value of output argument istat from PXFWAIT or PXFWAITPID.

Results:
The result type is INTEGER(4). The result is the number of the signal that caused the child process to stop.
The IPXFWSTOPSIG function should only be used if PXFWIFSTOPPED returns TRUE.

See Also: “PXFWAIT”, “PXFWAITPID”, “PXFWIFSTOPPED”

IPXFWTERMSIG

POSIX Function: Returns the number of the signal that caused a child process to terminate. This function is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

result = IPXFWTERMSIG (istat)

istat
(Input) INTEGER(4). The value of output argument istat from PXFWAIT or PXFWAITPID.

Results:
The result type is INTEGER(4). The result is the number of the signal that caused the child process to terminate.
The IPXFWTERMSIG function should only be used if PXFWIFSIGNALED returns TRUE.

See Also: “PXFWAIT”, “PXFWAITPID”, “PXFWIFSIGNALED”

IRAND, IRANDM

Portability Functions: Return random numbers in the range 0 through (2**31)–1, or 0 through (2**15)–1 if called without an argument.

Module: USE IFPORT

Syntax

result = IRAND ([iflag])
result = IRANDM (iflag)
iflag
(Input) INTEGER(4). Optional for IRAND. Controls the way the returned random number is chosen. If iflag is omitted, it is assumed to be 0, and the return range is 0 through \((2^{**15})-1\) (inclusive).

**Results:**
The result type is INTEGER(4). If iflag is 1, the generator is restarted and the first random value is returned. If iflag is 0, the next random number in the sequence is returned. If iflag is neither zero nor 1, it is used as a new seed for the random number generator, and the functions return the first new random value.

IRAND and IRANDM are equivalent and return the same random numbers. Both functions are included to ensure portability of existing code that references one or both of them.

You can use SRAND to restart the pseudorandom number generator used by these functions.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "SRAND", the RANDOM_NUMBER and RANDOM_SEED intrinsic routines in the Language Reference

**Example**
USE IFPORT
INTEGER(4) istat, flag_value, r_nums(20)
flag_value=1
r_nums(1) = IRAND (flag_value)
flag_value=0
do istat=2,20
   r_nums(istat) = irand(flag_value)
end do

**IRANGET**

**Portability Subroutine:** Returns the current seed.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL IRANGET (seed)
```

**seed**

(Output) INTEGER(4). The current seed value.
IRANSET

Portability Subroutine: Sets the seed for the random number generator.
Module: USE IFPORT
Syntax
   CALL IRANSET (seed)

seed
(Input) INTEGER(4). The reset value for the seed.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: "IRANGET"

ISATTY

Portability Function: Checks whether a logical unit number is a terminal.
Module: USE IFPORT
Syntax
   result = ISATTY (lunit)

lunit
(Input) INTEGER(4). An integer expression corresponding to a Fortran logical unit number. Must be in the range 0 to 100 and must be connected.

Results:
The result type is LOGICAL(4). The result is .TRUE. if the specified logical unit is connected to a terminal device; otherwise, .FALSE..
If lunit is out of range or is not connected, zero is returned.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
ITIME

**Portability Subroutine:** Returns the time in numeric form.

**Module:** USE IFPORT

**Syntax**

`CALL ITIME (array)`

array

(Output) INTEGER(4). A rank one array with three elements used to store numeric time data:

- array(1) – the hour
- array(2) – the minute
- array(3) – the second

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** the DATE_AND_TIME intrinsic procedure in the Language Reference

**Example**

```fortran
USE IFPORT
INTEGER(4) time_array(3)
CALL ITIME (time_array)
write(*,10) time_array
10 format (1X,I2,':',I2,':',I2)
END
```

JABS

**Portability Function:** Returns an absolute value.

**Module:** USE IFPORT

**Syntax**

`result = JABS (i)`

i

(Input) INTEGER(4). A value.

**Results:**

The result type is INTEGER(4). The value of the result is |i|.
JDATE

**Portability Function:** Returns an 8-character string with the Julian date in the form "yyddd". Three spaces terminate this string.

**Module:** USE IFPORT

**Syntax**

```fortran
result = JDATE ( )
```

**Results:**

The result type is character with length 8. The result is the Julian date, in the form YYDDD, followed by three spaces.

The Julian date is a five-digit number whose first two digits are the last two digits of the year, and whose final three digits represent the day of the year (1 for January 1, 366 for December 31 of a leap year, and so on). For example, the Julian date for February 1, 1999 is 99032.

---

**CAUTION.** The two-digit year return value may cause problems with the year 2000. Use the DATE_AND_TIME intrinsic subroutine instead (see the Language Reference).

---

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** the DATE_AND_TIME intrinsic procedure in the Language Reference

**Example**

```fortran
! Sets julian to today’s julian date
USE IFPORT
CHARACTER*8 julian
julian = JDATE( )
```

---

JDATE4

**Portability Function:** Returns a 10-character string with the Julian date in the form "yyyyddd". Three spaces terminate this string.

**Module:** USE IFPORT

**Syntax**

```fortran
result = JDATE4 ( )
```

---

2-188
Results:
The result type is character with length 10. The result is the Julian date, in the form YYYYDDD, followed by three spaces.

The Julian date is a seven-digit number whose first four digits are the year, and whose final three digits represent the day of the year (1 for January 1, 366 for December 31 of a leap year, and so on). For example, the Julian date for February 1, 1999 is 1999032.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: the DATE_AND_TIME intrinsic procedure in the Language Reference

KILL

Portability Function: Sends a signal to the process given by ID.

Module: USE IFPORT

Syntax

\[
\text{result} = \text{KILL} \left( \text{pid}, \text{signum} \right)
\]

\text{pid}

(Input) INTEGER(4). ID of a process to be signaled.

\text{signum}

(Input) INTEGER(4). A signal value. For the definition of signal values, see the “SIGNAL” function.

Results:
The result type is INTEGER(4). The result is zero if the call was successful; otherwise, an error code. Possible error codes are:

- EINV AL: The \text{signum} is not a valid signal number, or PID is not the same as getpid( ) and \text{signum} does not equal SIGKILL.
- ESRCH: The given PID could not be found.
- EPERM: The current process does not have permission to send a signal to the process given by PID.

On Windows* systems, arbitrary signals can be sent only to the calling process (where \text{pid} = getpid( )). Other processes can send only the SIGKILL signal (\text{signum} = 9), and only if the calling process has permission.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “RAISEQQ”, “SIGNALQQ”

Example

USE IFPORT
integer(4) id_number, sig_val, istat
id_number=getpid()
ISTAT = KILL (id_number, sig_val)

LCWRQQ

Portability Subroutine: Sets the value of the floating-point processor control word.
Module: USE IFPORT
Syntax
CALL LCWRQQ (controlword)
controlword 
(Input) INTEGER(2). Floating-point processor control word.
LCWRQQ performs the same function as the run-time subroutine SETCONTROLFPQQ and is
provided for compatibility.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “SETCONTROLFPQQ”

Example
USE IFPORT
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)
WRITE (*, 9000) 'Control word: ', control
9000 FORMAT (1X, A, Z4)
END
LINETO, LINETO_W

**Graphics Function:** Draws a line from the current graphics position up to and including the end point. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```lisp
result = LINETO (x, y)
result = LINETO_W (wx, wy)
```

*x, y*  
(Input) INTEGER(2). Viewport coordinates of end point.

*wx, wy*  
(Input) REAL(8). Window coordinates of end point.

**Results:**

The result type is INTEGER(2). The result is a nonzero value if successful; otherwise, 0.

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*), and 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, the figure must be drawn with a solid line style. Line style is solid by default and can be changed with SETLINESTYLE.

---

**NOTE.** The LINETO routine described here is a QuickWin routine. If you are trying to use the Microsoft* Platform SDK version of the LineTo routine by including the IFWIN module, you need to specify the routine name as MSFWINS$LineTo. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

---

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS

**See Also:**  "GETCURRENTPOSITION, GETCURRENTPOSITION_W", "GETLINESTYLE", "GRSTATUS", "MOVETO, MOVETO_W", "POLYGON, POLYGON_W", "POLYLINEQQ", "SETLINESTYLE", "SETWRITEMODE"
Example

This program draws the figure shown below.

! Build as QuickWin or Standard Graphics
USE IFQWIN
INTEGER(2) status
TYPE (xycoord) xy
CALL MOVETO(INT2(80), INT2(50), xy)
status = LINETO(INT2(240), INT2(150))
status = LINETO(INT2(240), INT2(50))
END

LINETOAR

Graphics Function: Draws a line between each x,y point in the from-array to each corresponding x,y point in the to-array. This function is only available on Windows® systems.

Module: USE IFQWIN

Syntax

result = LINETOAR(loc(fx), loc(fy), loc(tx) loc(ty), cnt)

fx (Input) INTEGER(2). From x viewport coordinate array.
fy (Input) INTEGER(2). From y viewport coordinate array.

tx (Input) INTEGER(2). To x viewport coordinate array.

ty (Input) INTEGER(2). To y viewport coordinate array.


**cnt**

(Input) INTEGER(4). Length of each coordinate array; all should be the same size.

**Results:**

The result is of type INTEGER(2). The result is a nonzero value if successful; otherwise, zero.

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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS

**See Also:** "LINETO, LINETO_W", "LINETOAREX", "SETCOLORRGB", "SETLINESTYLE", "SETWRITEMODE", the LOC intrinsic function in the Language Reference

**Example**

! Build for QuickWin or Standard Graphics
USE IFQWIN

integer(2) fx(3), fy(3), tx(3), ty(3), result
integer(4) cnt, i
! load the points

do i = 1, 3
  ! from here
  fx(i) = 20*i
  fy(i) = 10
  ! to there
  tx(i) = 20*i
  ty(i) = 60
end do

! draw the lines all at once
! 3 white vertical lines in upper left corner
result = LINETOAR(loc(fx), loc(fy), loc(tx), loc(ty), 3)
end

**LINETOAREX**

**Graphics Function:** 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. This function is only available on Windows* systems.

**Module:** USE IFQWIN
**Syntax**

```fortran
result = LINETOAREX (loc(fx), loc(fy), loc(tx), loc(ty), loc(C), loc(S), cnt)
```

*fx*  
(Input) INTEGER(2). From x viewport coordinate array.

*fy*  
(Input) INTEGER(2). From y viewport coordinate array.

*tx*  
(Input) INTEGER(2). To x viewport coordinate array.

*ty*  
(Input) INTEGER(2). To y viewport coordinate array.

*C*  
(Input) INTEGER(4). Color array.

*S*  
(Input) INTEGER(4). Style array.

*cnt*  
(Input) INTEGER(4). Length of each coordinate array; also the length of the color array and style array. All of the arrays should be the same size.

**Results:**

The result is of type INTEGER(2). The result is a nonzero value if successful; otherwise, zero.

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.

If the color has the Z'80000000' bit set, the color is an RGB color; otherwise, the color is a palette color.

The styles are as follows from wingdi.h:

<table>
<thead>
<tr>
<th>Style</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLID</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DASH</td>
<td>1</td>
<td>/* -- ------ */</td>
</tr>
<tr>
<td>DOT</td>
<td>2</td>
<td>/* ......... */</td>
</tr>
<tr>
<td>DASHDOT</td>
<td>3</td>
<td>/* <em>.</em>.<em>.</em> _ */</td>
</tr>
<tr>
<td>DASHDOTDOT</td>
<td>4</td>
<td>/* <em>.</em>.<em>.</em>._ _ _ */</td>
</tr>
<tr>
<td>NULL</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS
See Also: "LINETO, LINETO_W", "LINETOAR", "POLYLINEQQ", "SETWRITEMODE", the LOC intrinsic function in the Language Reference

Example

! Build for QuickWin or Standard Graphics
USE IFQWIN
integer(2) fx(3), fy(3), tx(3), ty(3), result
integer(4) C(3), S(3), cnt, i, color

color = Z'000000FF'
! load the points
do i = 1, 3
    S(i) = 0  ! all lines solid
    C(i) = IOR(Z'80000000',color)
    color = color*256 ! pick another of RGB
    if(IAND(color,Z'00FFFFFF').eq.0) color = Z'000000FF'
    ! from here
    fx(i) = 20*i
    fy(i) = 10
    ! to there
    tx(i) = 20*i
    ty(i) = 60
end do
! draw the lines all at once
! 3 vertical lines in upper left corner, Red, Green, and Blue
result = LINETOAREX(loc(fx),loc(fy),loc(tx),loc(ty),loc(C),loc(S),3)
end

LNBLNK

Portability Function: Locates the position of the last nonblank character in a string.
Module: USE IFPORT
Syntax
result = LNBLNK (string)
string
(Input) Character*(*) String to be searched. Cannot be an array.
Results:
The result type is INTEGER(4). The result is the index of the last nonblank character in string.
LNBLNK is very similar to the intrinsic function LEN_TRIM, except that string cannot be an array.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

See Also: the LEN_TRIM intrinsic function in the Language Reference

Example
USE IFPORT
integer(4) p
p = LNBLNK(‘ GOOD DAY ’) ! returns 9
p = LNBLNK(‘ ’) ! returns 0

LOADIMAGE, LOADIMAGE_W

Graphics Functions: Read an image from a Windows bitmap file and display it at a specified location. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax
result = LOADIMAGE (filename, xcoord, ycoord)
result = LOADIMAGE_W (filename, wxcoord, wycoord)

filename
(Input) Character*(*). Path of the bitmap file.

xcoord, ycoord
(Input) INTEGER(4). Viewport coordinates for upper-left corner of image display.

wxcoord, wycoord
(Input) REAL(8). Window coordinates for upper-left corner of image display.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a negative value.
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.
LOADIMAGE_W specifies the screen placement of the image in window coordinates.
Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB
See Also: “SAVEIMAGE, SAVEIMAGE_W”

LONG

Portability Function: Converts an INTEGER(2) argument to INTEGER(4) type.
Module: USE IFPORT
Syntax
   result = LONG (int2)

int2  
(Input) INTEGER(2). Value to be converted.

Results:
The result type is INTEGER(4). The result is the value of int2 with type INTEGER(4). The upper
16 bits of the result are zeros and the lower 16 are equal to int2.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB
See Also: the INT and KIND intrinsic functions in the Language Reference

LSTAT

Portability Function: Returns detailed information about a file.
Module: USE IFPORT
Syntax
   result = LSTAT (name, statb)

name  
(Input) Character*(*). Name of the file to examine.
statb  
(Output) INTEGER(4) or INTEGER(8). One-dimensional array of size 12; where the system
information is stored. See “STAT” for the possible values returned in statb.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, an error code (see
“IERRNO”).
LSTAT returns detailed information about the file named in `name`.

On Linux* systems, 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.

On Windows* systems, LSTAT returns exactly the same information as STAT (because there are no symbolic links on these systems). STAT is the preferred function.

The INQUIRE statement also provides information about file properties.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "GETFILEINFOQQ", "STAT", "FSTAT" the INQUIRE statement in the Language Reference

**Example**

```fortran
USE IFPORT
INTEGER(4) info_array(12), istatus
character*20 file_name
print *, "Enter name of file to examine: ", file_name
read *, file_name
ISTATUS = LSTAT (file_name, info_array)
if (.NOT. ISTATUS) then
  print *, info_array
else
  print *, 'Error ', istatus
end if
```

**LTIME**

**Portability Subroutine:** Returns the components of the local time zone time in a nine-element array.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL LTIME (time, array)
```

*time*  
(Input) INTEGER(4). An elapsed time in seconds since 00:00:00 Greenwich mean time, January 1, 1970.
array
(Output) INTEGER(4). One-dimensional array with 9 elements to contain local date and time data derived from time.

The elements of array are returned as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>array(1)</td>
<td>Seconds (0 - 59)</td>
</tr>
<tr>
<td>array(2)</td>
<td>Minutes (0 - 59)</td>
</tr>
<tr>
<td>array(3)</td>
<td>Hours (0 - 23)</td>
</tr>
<tr>
<td>array(4)</td>
<td>Day of month (1 - 31)</td>
</tr>
<tr>
<td>array(5)</td>
<td>Month (0 - 11)</td>
</tr>
<tr>
<td>array(6)</td>
<td>Years since 1900</td>
</tr>
<tr>
<td>array(7)</td>
<td>Day of week (0 - 6, where 0 is Sunday)</td>
</tr>
<tr>
<td>array(8)</td>
<td>Day of year (1 - 365)</td>
</tr>
<tr>
<td>array(9)</td>
<td>1 if daylight saving time is in effect; otherwise, 0.</td>
</tr>
</tbody>
</table>

CAUTION. This subroutine is not year-2000 compliant, use the DATE_AND_TIME intrinsic subroutine instead (see the Language Reference).

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: the DATE_AND_TIME intrinsic procedure in the Language Reference

Example
USE IFPORT
INTEGER(4) input_time, time_array(9)
! find number of seconds since 1/1/70
input_time=TIME()
! convert number of seconds to time array
CALL LTIME (input_time, time_array)
PRINT *, time_array
MAKEDIRQQ

Portability Function: Creates a new directory with a specified name.

Module: USE IFPORT

Syntax

\[
\text{result} = \text{MAKEDIRQQ} (\text{dirname})
\]

dirname

(Input) Character*(*). Name of directory to be created.

Results:

The result type is LOGICAL(4). The result is .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, call GETLASTERRORQQ to retrieve the error message. 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.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "DELDIRQQ", "CHANGEDIRQQ", "GETLASTERRORQQ"

Example

USE IFPORT
LOGICAL(4) result
result = MAKEDIRQQ('mynewdir')
IF (result) THEN
  WRITE (*,*) 'New subdirectory successfully created'
ELSE
  WRITE (*,*) 'Failed to create subdirectory'
END IF
END
**MBCharLen**

**NLS Function:** Returns the length, in bytes, of the first character in a multibyte-character string. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

\[
\text{result} = \text{MBCharLen} \ (\text{string})
\]

*string* (Input) Character*(*). String containing the character whose length is to be determined. Can contain multibyte characters.

**Results:**

The result type is INTEGER(4). The result is the number of bytes in the first character contained in *string*. The function returns 0 if *string* has no characters (is length 0).

MBCharLen does not test for multibyte character validity.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “MBCurMax”, “MBLead”, “MBLen”, “MBLen_Trim”

**MBConvertMBToUnicode**

**NLS Function:** Converts a multibyte-character string from the current codepage to a Unicode string. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

\[
\text{result} = \text{MBConvertMBToUnicode} \ (\text{mbstr}, \text{unicodestr} \ [, \text{flags}] )
\]

*mbstr* (Input) Character*(*). 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 (defined in IFNLS.F90) are:

- NLS$Precomposed: Use precomposed characters always. This is the default.
- NLS$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).

Results:
The result type is INTEGER(4). If no error occurs, the result is 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 space characters. 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.

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.

Compatibility
CONSOL STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “MBConvertUnicodeToMB”

MBConvertUnicodeToMB

NLS Function: Converts a Unicode string to a multibyte-character string from the current codepage. This function is only available on Windows® systems.

Module: USE IFNLS

Syntax
result = MBConvertUnicodeToMB (unicodestr, mbstr [, flags])
Descriptions of the Library Routines

unicodestr
(Input) INTEGER(2). Array of integers holding the Unicode string to be translated.

mbstr
(Output) Character*( ). 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 (defined in \texttt{IFNLS.F90}) are:

- NLS$CompositeCheck: Convert composite characters to precomposed.
- NLS$SepChars: Generate separate characters.
- NLS$DiscardDns: Discard nonspacing characters.
- NLS$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.

Results:
The result type is INTEGER(4). 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.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: \texttt{“MBConvertMBToUnicode”}
**MBCurMax**

**NLS Function:** Returns the longest possible multibyte character length, in bytes, for the current codepage. This function is only available on Windows® systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = MBCurMax ( )
```

**Results:**

The result type is INTEGER(4). The result is the longest possible multibyte character, in bytes, for the current codepage.

The MBLenMax parameter, defined in the module IFNLS.F90, is the longest length, in bytes, of any character in any codepage installed on the system.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “MBCharLen”

---

**MBINCHARQQ**

**NLS Function:** 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. This function is only available on Windows® systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = MBINCHARQQ (string)
```

**string**

(Output) CHARACTER(MBLenMax). String containing the read characters, padded with blanks up to the length MBLenMax. The MBLenMax parameter, defined in the module IFNLS.F90, is the longest length, in bytes, of any character in any codepage installed on the system.

**Results:**

The result type is INTEGER(4). The result is the number of characters read.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “INCHARQQ”, “MBCurMax”, “MBCharLen”, “MBLead”
MBINDEX

**NLS Function:** Performs the same function as the INDEX intrinsic function except that the strings manipulated can contain multibyte characters. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = MBINDEX (string, substring [, back])
```

**string**

(Input) CHARACTER*(*)). String to be searched for the presence of *substring*. Can contain multibyte characters.

**substring**

(Input) CHARACTER*(*)). 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 at end of *string* and moves toward the beginning.

**Results:**

The result type is INTEGER(4). 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*).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "MBSCAN", "MBVERIFY", the INDEX intrinsic function in the Language Reference

MBJISToJMS, MBJMSToJIS

**NLS Functions:** Converts Japan Industry Standard (JIS) characters to Microsoft Kanji (JMS) characters, or converts JMS characters to JIS characters. These functions are only available on Windows* systems.

**Module:** USE IFNLS
Syntax

result = MBJISToJMS (char)
result = MBJMSToJIS (char)

char
(Input) CHARACTER(2). JIS or JMS character to be converted.

A JIS character is converted only if the lead and trail bytes are in the hexadecimal range 21 through 7E.

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.

Results:
The result type is character with length 2. MBJISToJMS returns a Microsoft Kanji (Shift JIS or JMS) character. MBJMSToJIS returns a Japan Industry Standard (JIS) character.

Only computers with Japanese installed as one of the available languages can use the MBJISToJMS and MBJMSToJIS conversion functions.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSEnumLocales”, “NLSEnumCodepages”, “NLSGetLocale”, “NLSSetLocale”

MBLead

NLS Function: Determines whether a given character is the lead (first) byte of a multibyte character sequence. This function is only available on Windows* systems.

Module: USE IFNLS

Syntax

result = MBLead (char)

char
(Input) CHARACTER(1). Character to be tested for lead status.

Results:
The result type is LOGICAL(4). The result is .TRUE. if char is the first character of a multibyte character sequence; otherwise, .FALSE.. MBLead only works stepping forward through a whole multibyte character string. For example:

```
DO i = 1, LEN(str)  ! LEN returns the number of bytes, not the 
   ! number of characters in str
   WRITE(*, 100) MBLead (str(i:i))
```
MBLead is passed only one character at a time and must start on a lead byte and step through a string to establish context for the character. MBLead does not correctly identify a nonlead byte if it is passed only the second byte of a multibyte character because the status of lead byte or trail byte depends on context.

The function 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. So, MBStrLead can be much slower than MBLead (up to \( n \) times slower, where \( n \) is the length of the string).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also:  "MBStrLead", "MBCharLen"

**MBLen**

**NLS Function:**  Returns the number of characters in a multibyte-character string, including trailing blanks. This function is only available on Windows* systems.

**Module:**  USE IFNLS

**Syntax**

\[
\text{result} = \text{MBLen} \left( \text{string} \right)
\]

**string**

(Input) CHARACTER*(*). String whose characters are to be counted. Can contain multibyte characters.

**Results:**

The result type is INTEGER(4). The result is the number of characters in \( \text{string} \).

MBLen recognizes multibyte-character sequences according to the multibyte codepage currently in use. It does not test for multibyte-character validity.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also:  "MBLen_Trim", "MBStrLead"
MBLen_Trim

**NLS Function:** Returns the number of characters in a multibyte-character string, not including trailing blanks. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = MBLen_Trim (string)
```

**string**

(Input) Character*(*) String whose characters are to be counted. Can contain multibyte characters.

**Results:**

The result type is INTEGER(4). The result is the number of characters in string minus any trailing blanks (blanks are bytes containing character 32 (hex 20) in the ASCII collating sequence). MBLen_Trim recognizes multibyte-character sequences according to the multibyte codepage currently in use. It does not test for multibyte-character validity.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “MBLen”, “MBStrLead”

MBLGE, MBLGT, MBLLE, MBLLT, MBLEQ, MBLNE

**NLS Functions:** Perform the same functions as the LGE, LGT, LLE, and LLT intrinsic functions and the logical operators .EQ. and .NE., except that the strings being compared can include multibyte characters, and optional flags can modify the comparison. These functions are only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = MBLGE (string_a, string_b, [flags])
result = MBLGT (string_a, string_b, [flags])
result = MBLLE (string_a, string_b, [flags])
result = MBLLT (string_a, string_b, [flags])
result = MBLEQ (string_a, string_b, [flags])
result = MBLNE (string_a, string_b, [flags])
```

**string_a, string_b**

(Input) Character*(*) 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 (defined in IFNLS.F90) 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 the Note below).

Results:
The result type is LOGICAL(4). 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 results of these functions are as follows:
- MBLGE returns .TRUE. if the strings are equal or string_a comes last in the collating sequence; otherwise, .FALSE..
- MBLGT returns .TRUE. if string_a comes last in the collating sequence; otherwise, .FALSE..
- MBLLE returns .TRUE. if the strings are equal or string_a comes first in the collating sequence; otherwise, .FALSE..
- MBLLT returns .TRUE. if string_a comes first in the collating sequence; otherwise, .FALSE..
- MBLEQ returns .TRUE. if the strings are equal in the collating sequence; otherwise, .FALSE..
- MBLNE returns .TRUE. if the strings are not equal in the collating sequence; otherwise, .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..

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 NLSSMB_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 NLSSMB_StringSort flag, hyphen and apostrophe sort at the beginning also.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: the LGE, LGT, LLE, and LLT intrinsic functions in the Language Reference

MBNext

NLS Function: Returns the position of the first lead byte or single-byte character immediately following the given position in a multibyte-character string. This function is only available on Windows® systems.

Module: USE IFNLS

Syntax
result = MBNext(string, position)

string
(Input) Character*(*). 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.

Results:
The result type is INTEGER(4). The result is the 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.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “MBPrev”
MBPrev

**NLS Function:** Returns the position of the first lead byte or single-byte character immediately preceding the given string position in a multibyte-character string. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```
result = MBPrev (string, position)
```

`string`
(Input) Character*(*). 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.

**Results:**

The result type is INTEGER(4). The result is the 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

See Also: “MBNext”

---

MBSCAN

**NLS Function:** Performs the same function as the SCAN intrinsic function except that the strings manipulated can contain multibyte characters. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```
result = MBSCAN (string, set [, back])
```

`string`
(Input) Character*(*). String to be searched for the presence of any character in set.

`set`
(Input) Character*(*). 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.

Results:
The result type is INTEGER(4). If back is .FALSE. or is omitted, it returns the position of the
leftmost character in string that is in set. If back is .TRUE., it returns the rightmost character in
string that is in set. If no characters in string are in set, it returns 0.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “MBINDEX”, “MBVERIFY”, the SCAN intrinsic function in the Language Reference

**MBStrLead**

**NLS Function:** Performs a context-sensitive test to determine whether a given character byte in a
string is a multibyte-character lead byte. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```
result = MBStrLead (string, position)
```

**string**
(Input) Character*(*). 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.

**Results:**
The result type is LOGICAL(4). The result is .TRUE. if the character byte in position of string is a
lead byte; otherwise, .FALSE..

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. So, MBStrLead can be much
slower than MBLead (up to n times slower, where n is the length of the string).

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
MBVERIFY

NLS Function: Performs the same function as the VERIFY intrinsic function except that the strings manipulated can contain multibyte characters. This function is only available on Windows* systems.

Module: USE IFNLS

Syntax
   result = MBVERIFY (string, set [, back])

string
(Input) Character*(*). String to be searched for presence of any character not in set.

set
(Input) Character*(*). 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.

Results:
The result type is INTEGER(4). If back is .FALSE. or is omitted, it returns the position of the leftmost character in string that is not in set. If back is .TRUE., it returns the rightmost character in string that is not in set. If all the characters in string are in set, it returns 0.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “MBINDEX”, “MBSCAN”, the VERIFY intrinsic function in the Language Reference

MESSAGEBOXQQ

QuickWin Function: Displays a message box in a QuickWin window. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax
   result = MESSAGEBOXQQ (msg, caption, mtype)
**msg**
(Input) Character*(*)). Null-terminated C string. Message the box displays.

**caption**
(Input) Character*(*)). 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 properties of the message box. You can combine several constants (defined in IFQWIN.F90) using an inclusive OR (IOR or OR). The symbolic constants and their associated objects or properties are as follows:

- 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$ICONASTERISK, MB$ICONINFORMATION – Lowercase i in blue circle icon.
- MB$ICONEXCLAMATION – The exclamation-mark icon.
- MB$ICONHAND, MB$ICONSTOP – The stop-sign icon.
- MB$ICONQUESTION – The question-mark 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.

**Results:**
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.
Compatiblity
QUICKWIN GRAPHICS LIB

See Also: “ABOUTBOXQQ”, “SETMESSAGEQQ”, “Using QuickWin” in Building Applications

Example
! Build as QuickWin app
USE IFQWIN
message = MESSAGEBOXQQ('Do you want to continue?'C, &
'Matrix'C, &
MB$ICONQUESTION.OR.MB$YESNO.OR.MB$DEFBUTTON1)
END

MODIFYMENUFLAGSQQ

QuickWin Function: Modifies a menu item’s state. This function is only available on Windows* systems.
Module: USE IFQWIN
Syntax
result = MODIFYMENUFLAGSQQ (menuID, itemID, flag)

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.

flag
(Input) INTEGER(4). Constant indicating the menu state. Flags can be combined with an inclusive OR (see the 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.
• $MENUSEPARATOR – Draws a separator bar.
• $MENUCHECKED – Puts a check by the menu item.
• $MENUUNCHECKED – Removes the check by the menu item.
Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
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.

Compatibility
QUICKWIN GRAPHICS LIB

See Also: “APPENDMENUQQ”, “DELETEMENUQQ”, “INSERTMENUQQ”, “MODIFYMENUROUTINEQQ”, “MODIFYMENUSTRINGQQ”, "Using QuickWin” in Building Applications

Example
USE IFQWIN
LOGICAL(4) result
CHARACTER(20) str

! Append item to the bottom of the first (FILE) menu
str = 'Add to File Menu'
result = APPENDMENUQQ(1, $MENUENABLED, str, WINSTATUS)

! Gray out and disable the first two menu items in the
! first (FILE) menu
result = MODIFYMENUFLAGSQQ (1, 1, $MENUGRAYED)
result = MODIFYMENUFLAGSQQ (1, 2, $MENUGRAYED)
END

MODIFYMENUROUTINEQQ

QuickWin Function: Changes a menu item’s callback routine. This function is only available on Windows® systems.
Module: USE IFQWIN
Syntax
result = MODIFYMENUROUTINEQQ (menuIdD, itemID, routine)

menuIdD
(Input) INTEGER(4). Identifies the menu that contains the item whose callback routine is 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 take a single LOGICAL parameter that 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.
- WINSELECTTEXT – Selects text from the current window.
- WINSELECTGRAPHICS – Selects graphics from the current window.
- WINSELECTALL – Selects the entire contents of the current window.
- WININPUT – Brings to the top the child window requesting input and makes it 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.

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
MODIFYMENUSTRINGQQ

QuickWin Function: Changes a menu item’s text string. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = MODIFYMENUSTRINGQQ (menuID, itemID, text)

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*(*) Menu item name. Must be a null-terminated C string. For example, words of text'C.

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

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 "P&rint"C as text.

Compatibility
QUICKWIN GRAPHICS LIB

See Also: "APPENDMENUQQ", "DELETEMENUQQ", "INSERTMENUQQ", "MODIFYMENUFLAGSQQ", "MODIFYMENUSTRINGQQ". "Using QuickWin" in Building Applications

Example
USE IFQWIN
LOGICAL(4) result
CHARACTER(25) str
! Append item to the bottom of the first (FILE) menu
str = 'Add to File Menu'C
result = APPENDMENUQQ(1, $MENUENABLED, str, WINSTATUS)

! Change the name of the first item in the first menu
str = 'Browse'C
result = MODIFYMENUSTRINGQQ(1, 1, str)
END

MOVETO, MOVETO_W

Graphics Subroutines: Move the current graphics position to a specified point. No drawing occurs. These subroutines are only available on Windows* systems.

Module: USE IFQWIN

Syntax
CALL MOVETO (x, y, t)
CALL MOVETO_W (wx, wy, wt)

x, y
(Input) INTEGER(2). Viewport coordinates of the new graphics position.

(t)
(Output) Derived type xycoord. Viewport coordinates of the previous graphics position. The derived type xycoord is defined in IFQWIN.F90 as follows:

TYPE xycoord
   INTEGER(2) xcoord  ! x coordinate
   INTEGER(2) ycoord  ! y coordinate
END TYPE xycoord

wx, wy
(Input) REAL(8). Window coordinates of the new graphics position.

(wt)
(Output) Derived type wxycoord. Window coordinates of the previous graphics position. The derived type wxycoord is defined in IFQWIN.F90 as follows:

TYPE wxycoord
   REAL(8) wx  ! x window coordinate
   REAL(8) wy  ! y window coordinate
END TYPE wxycoord
MOVETO sets the current graphics position to the viewport coordinate \((x, y)\). MOVETO_W sets the current graphics position to the window coordinate \((wx, wy)\).

MOVETO and MOVETO_W assign the coordinates of the previous position to \(t\) and \(wt\), respectively.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** “GETCURRENTPOSITION, GETCURRENTPOSITION_W”, “LINETO, LINETO_W”, “OUTGTEXT”

**Example**

```fortran
! Build as QuickWin or Standard Graphics ap.
USE IFQWIN
INTEGER(2) status, x, y
INTEGER(4) result
TYPE (xycoord) xy
RESULT = SETCOLORRGB(Z'FF0000') ! blue
x = 60
! Draw a series of lines
DO y = 50, 92, 3
   CALL MOVETO(x, y, xy)
   status = LINETO(INT2(x + 20), y)
END DO
END
```

**NLSEnumCodepages**

**NLS Function:** Returns an array containing the codepages supported by the system, with each array element describing one valid codepage. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```fortran
ptr => NLSEnumCodepages ( )
```

**Results:**

The result is a pointer to an array of codepages, with each element describing one supported codepage.
NOTE. After use, the pointer returned by NLSEnumCodepages should be deallocated with the DEALLOCATE statement.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSEnumLocales”, the DEALLOCATE statement in the Language Reference

NLSEnumLocales

NLS Function: Returns an array containing the language and country combinations supported by the system, in which each array element describes one valid combination. This function is only available on Windows* systems.

Module: USE IFNLS

Syntax

\[ ptr \rightarrow \text{NLSEnumLocales}() \]

Results:
The result is a pointer to an array of locales, in which each array element describes one supported language and country combination. Each element has the following structure:

```fortran
TYPE NLS$EnumLocale
    CHARACTER*(NLS$MaxLanguageLen)  Language
    CHARACTER*(NLS$MaxCountryLen)   Country
    INTEGER(4)                      DefaultWindowsCodepage
    INTEGER(4)                      DefaultConsoleCodepage
END TYPE
```

If the application is a Windows or 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.

NOTE. After use, the pointer returned by NLSEnumLocales should be deallocated with the DEALLOCATE statement.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSEnumCodepages”, the DEALLOCATE statement in the Language Reference

NLSFormatCurrency

NLS Function: Returns a correctly formatted currency string for the current locale. This function is only available on Windows* systems.

Module: USE IFNLS

Syntax
result = NLSFormatCurrency (outstr, instr [, flags ])

outstr
(Output) Character*(*)). 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*(*)). 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, the flag NLS$Normal is used. Available values (defined in IFNLS.F90) are:
• NLS$Normal – No special formatting
• NLS$NoUserOverride – Do not use user overrides

Results:
The result type is INTEGER(4). The result is the number of characters written to outstr (bytes are counted, not multibyte characters). If an error occurs, the result is one of the following negative values:
• NLS$ErrorInsufficientBuffer – outstr buffer is too small
• NLS$ErrorInvalidFlags – flags has an illegal value
• NLS$ErrorInvalidInput – instr has an illegal value

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSFormatNumber”, “NLSFormatDate”, “NLSFormatTime”
**Example**

```fortran
USE IFNLS
CHARACTER(40) str
INTEGER(4) i
i = NLSFormatCurrency(str, "1.23")
print *, str ! prints $1.23
i = NLSFormatCurrency(str, "1000000.99")
print *, str ! prints $1,000,000.99
i = NLSSetLocale("Spanish", "Spain")
i = NLSFormatCurrency(str, "1.23")
print *, str ! prints 1 Pts
i = NLSFormatCurrency(str, "1000000.99")
print *, str ! prints 1,000,001 Pts
```

**NLSFormatDate**

**NLS Function:** Returns a correctly formatted string containing the date for the current locale. This function is only available on Windows® systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = NLSFormatDate (outstr [, intime ] [, flags])
```

- `outstr` (Output) Character*(*). 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$Normal is used. Available values (defined in IFNLS.F90) are:
  - NLS$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

Results:
The result type is INTEGER(4). The result is the number of characters written to outstr (bytes are counted, not multibyte characters). If an error occurs, the result is one of the following negative values:

- NLS$ErrorInsufficientBuffer – outstr buffer is too small
- NLS$ErrorInvalidFlags – flags has an illegal value
- NLS$ErrorInvalidInput – intime has an illegal value

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSFormatTime”, “NLSFormatCurrency”, “NLSFormatNumber”

Example
USE IFNLS
INTEGER(4) i
CHARACTER(40) str
i = NLSFORMATDATE(str, NLS$NORMAL)           ! 8/1/99
i = NLSFORMATDATE(str, NLS$USEALTCALENDAR)   ! 8/1/99
i = NLSFORMATDATE(str, NLS$LONGDATE)         ! Monday, August 1, 1999
i = NLSFORMATDATE(str, NLS$SHORTDATE)        ! 8/1/99
END

NLSFormatNumber

NLS Function: Returns a correctly formatted number string for the current locale. This function is only available on Windows* systems.

Module: USE IFNLS

Syntax
result = NLSFormatNumber (outstr, instr [, flags ])

outstr
(Output) Character*(*) 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*(*) 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 (defined in IFNLS.F90) are:

• NLS$Normal – No special formatting
• NLS$NoUserOverride – Do not use user overrides

Results:
The result type is INTEGER(4). The result is the number of characters written to outstr (bytes are counted, not multibyte characters). If an error occurs, the result is one of the following negative values:

• NLS$ErrorInsufficientBuffer – outstr buffer is too small
• NLS$ErrorInvalidFlags – flags has an illegal value
• NLS$ErrorInvalidInput – instr has an illegal value

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSFormatTime”, “NLSFormatCurrency”, “NLSFormatDate”

Example
USE IFNLS
CHARACTER(40) str
INTEGER(4) i
i = NLSFormatNumber(str, "1.23")
print *, str                   ! prints 1.23
i = NLSFormatNumber(str, "1000000.99")
print *, str                   ! prints 1,000,000.99
i = NLSSetLocale("Spanish", "Spain")
i = NLSFormatNumber(str, "1.23")
print *, str                   ! prints 1,23
i = NLSFormatNumber(str, "1000000.99")
print *, str                   ! prints 1.000.000,99

NLSFormatTime

NLS Function: Returns a correctly formatted string containing the time for the current locale. This function is only available on Windows* systems.

Module: USE IFNLS
Syntax

\[
\text{result} \leftarrow \text{NLSFormatTime} \left( \text{outstr} \[, \text{intime} \[, \text{flags} \] \right)
\]

\text{outstr}

(Output) Character*(*) String containing the correctly formatted time for the current locale. If \text{outstr} is longer than the formatted time, it is blank-padded.

\text{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 \text{PACKTIMEQQ}. If you omit \text{intime}, the current system time is formatted and returned in \text{outstr}.

\text{flags}

(Optional; input) INTEGER(4). If specified, modifies the time conversion. If you omit \text{flags}, the flag NLS$Normal is used. Available values (defined in \text{IFnls.F90}) 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

Results:

The result type is INTEGER(4). The result is the number of characters written to \text{outstr} (bytes are counted, not multibyte characters). If an error occurs, the result is one of the following negative values:

- NLS$ErrorInsufficientBuffer – \text{outstr} buffer is too small
- NLS$ErrorInvalidFlags – \text{flags} has an illegal value
- NLS$ErrorInvalidInput – \text{intime} has an illegal value

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSFormatCurrency”, “NLSFormatDate”, “NLSFormatNumber”

Example

\begin{verbatim}
USE IFNLS
INTEGER (4) i
CHARACTER (20) str
i = NLSFORMATTIME (str, NLS$NORMAL)                  ! 11:38:28 PM
i = NLSFORMATTIME (str, NLS$NOMINUTESORSECONDS)    ! 11 PM
\end{verbatim}
**NLSGetEnvironmentCodepage**

**NLS Function:** Returns the codepage number for the system (Window) codepage or the console codepage. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```fortran
result = NLSGetEnvironmentCodepage (flags)
```

**flags**

(Input) INTEGER(4). Tells the function which codepage number to return. Available values (defined in IFNLS.F90) are:

- NLS$ConsoleEnvironmentCodepage – Gets the codepage for the console
- NLS$WindowsEnvironmentCodepage – Gets the current Windows codepage

**Results:**

The result type is INTEGER(4). The result is zero if successful; otherwise, it returns one of the following error codes:

- NLS$ErrorInvalidFlags – flags has an illegal value
- NLS$ErrorNoConsole – There is no console associated with the given application; so, operations with the console codepage are not possible

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “NLSSetEnvironmentCodepage”

**NLSGetLocale**

**NLS Subroutine:** Returns the current language, country, or codepage. This subroutine is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

```fortran
CALL NLSGetLocale ([language] [, country] [, codepage])
```
language
(Optional; output) Character*(*) Current language.
country
(Optional; output) Character*(*) Current country.
codepage
(Optional; output) INTEGER(4). Current codepage.
NLSGetLocale returns a valid codepage in codepage. It does not return one of the NLS$... symbolic constants that can be used with NLSSetLocale.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSSetLocale”

Example
USE IFNS
CHARACTER(50) cntry, lang
INTEGER(4) code
CALL NLSGetLocale (lang, cntry, code) ! get all three
CALL NLSGetLocale (CODEPAGE = code) ! get the codepage
CALL NLSGetLocale (COUNTRY=cntry,CODEPAGE=code) ! get country and codepage

NLSGetLocaleInfo

NLS Function: Returns information about the current locale. This function is only available on Windows* systems.
Module: USE IFNS

Syntax
result = NLSGetLocaleInfo (type, outstr)

type
(Input) INTEGER(4). NLS parameter requested. A list of parameter names is given in the Table 2-1.
outstr
(Output) Character*(*) 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
(W*32, W*64)”.

Results:
The result type is INTEGER(4). The result is the number of characters written to outstr if
successful, or if outstr has 0 length, the number of characters required to hold the requested
information. Otherwise, the result is one of the following error codes (defined in IFNLS.F90):
• NLS$ErrorInvalidLIType – The given is invalid
• NLS$ErrorInsufficientBuffer – The outstr buffer was too small, but was not 0 (so that the
needed size would be returned)
The NLS$LI parameters are used for the argument and select the locale information returned by
NLSGetLocaleInfo in outstr. You can perform an inclusive OR with NLS$NoUserOverride and
any NLS$LI parameter. This causes NLSGetLocaleInfo to bypass any user overrides and always
return the system default value.

Table 2-1 lists and briefly describes the NLS$LI parameters.

<table>
<thead>
<tr>
<th>Parameter</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$LI_SENGLANGUAGE</td>
<td>The full English name of the language from the ISO Standard 639. This will always be restricted to characters that map into the ASCII 127 character subset.</td>
</tr>
<tr>
<td>NLS$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$LI_SNATIVELANGNAME</td>
<td>The native name of the language.</td>
</tr>
<tr>
<td>NLS$LI_ICOUNTRY</td>
<td>The country code, based on international phone codes, also referred to as IBM country codes.</td>
</tr>
<tr>
<td>NLS$LI_SCOUNTRY</td>
<td>The full localized name of the country.</td>
</tr>
<tr>
<td>NLS$LI_SENGCOUNTRY</td>
<td>The full English name of the country. This will always be restricted to characters that map into the ASCII 127 character subset.</td>
</tr>
<tr>
<td>NLS$LI_SABBREVCTRYNAME</td>
<td>The abbreviated name of the country as per ISO Standard 3166.</td>
</tr>
<tr>
<td>NLS$LI_SNATIVECTRYNAME</td>
<td>The native name of the country.</td>
</tr>
</tbody>
</table>
Table 2-1  NLS LocaleInfo Parameters (W*32, W*64)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>NLS$LI_SDECIMAL</td>
<td>The character(s) used as decimal separator. This is restricted such that it cannot be set to digits 0 - 9.</td>
</tr>
<tr>
<td>NLS$LI_STHOUSAND</td>
<td>The character(s) used as separator between groups of digits left of the decimal. This is restricted such that it cannot be set to digits 0 - 9.</td>
</tr>
<tr>
<td>NLS$LI_SGROUPING</td>
<td>Sizes for each group of digits to the left of the decimal. An explicit size is needed for each group; sizes are separated by semicolons. If the last value is 0 the preceding value is repeated. To group thousands, specify &quot;3;0&quot;.</td>
</tr>
<tr>
<td>NLS$LI_IDIGITS</td>
<td>The number of decimal digits.</td>
</tr>
<tr>
<td>NLS$LI_ILZERO</td>
<td>Determines whether to use leading zeros in decimal fields: 0 - Use no leading zeros 1 - Use leading zeros</td>
</tr>
<tr>
<td>NLS$LI_INEGNUMBER</td>
<td>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 –</td>
</tr>
<tr>
<td>NLS$LI_SNATIVEDIGITS</td>
<td>The ten characters that are the native equivalent to the ASCII 0-9.</td>
</tr>
<tr>
<td>NLS$LI_SSCURRENCY</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 &quot;Codes for the Representation of Currencies and Funds&quot;, followed by the character separating this string from the amount.</td>
</tr>
</tbody>
</table>
Table 2-1  NLS LocaleInfo Parameters (W*32, W*64)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_SMONDECIMALSEP</td>
<td>The character(s) used as monetary decimal separator. This is restricted such that it cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$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$LI_IINTLCURRDIGITS</td>
<td>Number of decimal digits for the international monetary format.</td>
</tr>
<tr>
<td>NLS$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_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$LI_IPOSSIGNPOSN</td>
<td>Determines the formatting index for positive values:</td>
</tr>
<tr>
<td></td>
<td>0 - Parenthesis surround the amount and the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>1 - The sign string precedes the amount and the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>2 - The sign string follows the amount and the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>3 - The sign string immediately precedes the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>4 - The sign string immediately follows the monetary symbol</td>
</tr>
<tr>
<td>NLS$LI_INEGCURR</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 after $)</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 $)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLS$LI_INEGSIGNPOSN</td>
<td>Determines the formatting index for negative values. Same values as for NLS$LI_IPOSSIGNPOSN.</td>
</tr>
<tr>
<td>NLS$LI_IPOSSYM/precedes</td>
<td>1 if the monetary symbol precedes, 0 if it follows a positive amount.</td>
</tr>
<tr>
<td>NLS$LI_IPOSSEP/BYSPACE</td>
<td>1 if the monetary symbol is separated by a space from a positive amount; otherwise, 0.</td>
</tr>
<tr>
<td>NLS$LI_INEGSYM/precedes</td>
<td>1 if the monetary symbol precedes, 0 if it follows a negative amount.</td>
</tr>
<tr>
<td>NLS$LI_INEGSEP/BYSPACE</td>
<td>1 if the monetary symbol is separated by a space from a negative amount; otherwise, 0.</td>
</tr>
<tr>
<td>NLS$LI_STIMEFORMAT</td>
<td>Time formatting string. See “NLS Date and Time Format (W<em>32, W</em>64)” for explanations of the valid strings.</td>
</tr>
<tr>
<td>NLS$LI_STIME</td>
<td>Character(s) for the time separator. Cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_ITIME</td>
<td>Time format:</td>
</tr>
<tr>
<td></td>
<td>0 - Use 12-hour format</td>
</tr>
<tr>
<td></td>
<td>1 - Use 24-hour format</td>
</tr>
<tr>
<td>NLS$LI_ITLZERO</td>
<td>Determines whether to use leading zeros in time fields:</td>
</tr>
<tr>
<td></td>
<td>0 - Use no leading zeros</td>
</tr>
<tr>
<td></td>
<td>1 - Use leading zeros for hours</td>
</tr>
<tr>
<td>NLS$LI_S1159</td>
<td>String for the AM designator.</td>
</tr>
<tr>
<td>NLS$LI_S2359</td>
<td>String for the PM designator.</td>
</tr>
<tr>
<td>NLS$LI_SSHORTDATE</td>
<td>Short Date formatting string for this locale. The d, M and y should have the day, month, and year substituted, respectively. See “NLS Date and Time Format (W<em>32, W</em>64)” for explanations of the valid strings.</td>
</tr>
<tr>
<td>NLS$LI_SDATE</td>
<td>Character(s) for the date separator. Cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_IDATE</td>
<td>Short 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$LI_ICENTURY</td>
<td>Specifies whether to use full 4-digit century for the short date only:</td>
</tr>
<tr>
<td></td>
<td>0 - Two-digit year</td>
</tr>
<tr>
<td></td>
<td>1 - Full century</td>
</tr>
<tr>
<td>NLS$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>
</tbody>
</table>
### Table 2-1 NLS LocaleInfo Parameters (W*32, W*64)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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. The string returned may contain a string within single quotes (&quot;'&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>
<tr>
<td>NLS$LI_IOPTIONALCALENDAR</td>
<td>Specifies which additional calendar types are valid and available for this locale. This can be a null separated list of all valid optional calendars:</td>
</tr>
<tr>
<td></td>
<td>0 - No additional types valid</td>
</tr>
<tr>
<td></td>
<td>1 - Gregorian (localized)</td>
</tr>
<tr>
<td></td>
<td>2 - Gregorian (English strings always)</td>
</tr>
<tr>
<td></td>
<td>3 - Era: Year of the Emperor (Japan)</td>
</tr>
<tr>
<td></td>
<td>4 - Era: Year of the Republic of China</td>
</tr>
<tr>
<td></td>
<td>5 - Tangun Era (Korea)</td>
</tr>
<tr>
<td>NLS$LI_IFIRSTDAYOFWEEK</td>
<td>Specifies which day is considered first in a week:</td>
</tr>
<tr>
<td></td>
<td>0 - SDAYNAME1</td>
</tr>
<tr>
<td></td>
<td>1 - SDAYNAME2</td>
</tr>
<tr>
<td></td>
<td>2 - SDAYNAME3</td>
</tr>
<tr>
<td></td>
<td>3 - SDAYNAME4</td>
</tr>
<tr>
<td></td>
<td>4 - SDAYNAME5</td>
</tr>
<tr>
<td></td>
<td>5 - SDAYNAME6</td>
</tr>
<tr>
<td></td>
<td>6 - SDAYNAME7</td>
</tr>
<tr>
<td>NLS$LI_IFIRSTWEEKOFYEAR</td>
<td>Specifies which week of the year is considered first:</td>
</tr>
<tr>
<td></td>
<td>0 - Week containing 1/1</td>
</tr>
<tr>
<td></td>
<td>1 - First full week following 1/1</td>
</tr>
<tr>
<td></td>
<td>2 - First week containing at least 4 days</td>
</tr>
</tbody>
</table>
**Table 2-1**  NLS LocaleInfo Parameters (W*32, W*64)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_SDAYNAME1 - NLS$LI_SDAYNAME7</td>
<td>Native name for each day of the week. 1 = Monday, 2 = Tuesday, etc.</td>
</tr>
<tr>
<td>NLS$LI_SABBREVVDAYNAME1 - NLS$LI_SABBREVVDAYNAME7</td>
<td>Native abbreviated name for each day of the week. 1 = Mon, 2 = Tue, etc.</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME1 - NLS$LI_SMONTHNAME13</td>
<td>Native name for each month. 1 = January, 2 = February, etc. 13 = the 13th month, if it exists in the locale.</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME1 - NLS$LI_SABBREVMONTHNAME13</td>
<td>Native abbreviated name for each month. 1 = Jan, 2 = Feb, etc. 13 = the 13th month, if it exists in the locale.</td>
</tr>
</tbody>
</table>

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**NLSSetEnvironmentCodepage**

**NLS Function**: 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. This function is only available on Windows® systems.

**Module**: USE IFNLS

**Syntax**

```fortran
result = NLSSetEnvironmentCodepage (codepage, flags)
```

`codepage`

(Input) INTEGER(4). Number of the codepage to set as the console codepage.

`flags`

(Input) INTEGER(4). Must be set to NLS$ConsoleEnvironmentCodepage.

**Results**: The result type is INTEGER(4). The result is zero if successful. Otherwise, returns one of the following error codes (defined in IFNLS.F90):

- NLS$ErrorInvalidCodepage – codepage is invalid or not installed on the system
- NLS$ErrorInvalidFlags – flags is not valid
- NLS$ErrorNoConsole – There is no console associated with the given application; so operations, with the console codepage are not possible
The flags argument must be NLS$ConsoleEnvironmentCodepage; it cannot be NLS$WindowsEnvironmentCodepage. NLSSetEnvironmentCodepage does not affect the Windows* codepage.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSSetEnvironmentCodepage”

**NLSSetLocale**

**NLS Function:** Sets the current language, country, or codepage. This function is only available on Windows* systems.

**Module:** USE IFNLS

**Syntax**

\[
\text{result} = \text{NLSSetLocale} (\text{language}[, \text{country}][, \text{codepage}])
\]

- **language**
  (Input) Character*(*). One of the languages supported by the Windows* NLS APIs.

- **country**
  (Optional; input) Character*(*). 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 (defined in IFNLS.F90):
  - 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$WindowsLanguageCodepage. At program startup, NLSS$WindowsEnvironmentCodepage is used to set the codepage.
Results:
The result type is INTEGER(4). The result is zero if successful. Otherwise, one of the following error codes (defined in IFNLS.F90) may be returned:

- NLSS$ErrorInvalidLanguage – language is invalid or not supported
- NLSS$ErrorInvalidCountry – country is invalid or is not valid with the language specified
- NLSS$ErrorInvalidCodepage – codepage is invalid or not installed on the system

NOTE. NLSSetLocale works on installed locales only. Windows systems support many locales, but they must be installed through the system Control Panel/International menu.

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’s 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’s setlocale( ) routine is independent of NLSSetLocale.

Calling NLSSetLocale with the default environment console codepage, NLSSConsoleEnvironmentCodepage, causes an implicit call to the Windows API SetFileApisToOEM( ). Calling NLSSetLocale with any other codepage causes a call to SetFileApisToANSI( ).

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “NLSSetLocale”

OUTGTEXT

Graphics Subroutine: In graphics mode, sends a string of text to the screen, including any trailing blanks. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL OUTGTEXT (text)

(text)

(Input) Character*(*). String to be displayed.
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 the INITIALIZEFONTS function.

Because OUTGTEXT is a graphics function, the color of text is affected by the SETCOLORRGB function, not by SETTEXTCOLORRGB.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** \[ "GETFONINFORON", "GETGTEXTEXTENT", "INITIALIZEFONTS", "MOVETO, MOVETO_W", "SETCOLORRGB", "SETFONT", "SETGTEXTROTATION" \]

**Example**

! build as a QuickWin App.

USE IFQWIN

INTEGER(2) result

INTEGER(4) i

TYPE (xycoord) xys

result = INITIALIZEFONTS()

result = SETFONT('t''Arial''h18w10pvib')

do i=1,6

CALL MOVETO(INT2(0),INT2(30*(i-1)),xys)

grstat=SETCOLOR(INT2(i))

CALL OUTGTEXT('This should be ')

SELECT CASE (i)

CASE (1)

CALL OUTGTEXT('Blue')

CASE (2)

CALL OUTGTEXT('Green')

CASE (3)

CALL OUTGTEXT('Cyan')

CASE (4)

CALL OUTGTEXT('Red')

CASE (5)

CALL OUTGTEXT('Magenta')

CASE (6)

CALL OUTGTEXT('Orange')

END SELECT
OUTTEXT

**Graphics Subroutine:** In text or graphics mode, sends a string of text to the screen, including any trailing blanks. This subroutine is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
CALL OUTTEXT (text)
```

**text**

(Input) Character*(*) String to be displayed.

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.

To output text using special fonts, you must use the OUTGTEXT subroutine.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS LIB

**See Also:** "OUTGTEXT", "SETTEXTPosition", "SETTEXTCOLORRGB", the WRITE statement in the Language Reference

**Example**

```fortran
USE IFQWIN
INTEGER(2) oldcolor
TYPE (rccoord) rc

CALL CLEARSCREEN($GCLEARSCREEN)
CALL SETTEXTPosition (INT2(1), INT2(5), rc)
oldcolor = SETTEXTCOLOR(INT2(4))
CALL OUTTEXT ('Hello, everyone')
END
```

PACKTIMEQQ

**Portability Subroutine:** Packs time and date values.

**Module:** USE IFPORT
Syntax

CALL PACKTIMEQQ (timedate, iyr, imon, iday, ihr,imin, isec)

*timedate*
(Output) INTEGER(4). Packed time and date information.

*iyr*
(Input) INTEGER(2). Year (xxxx AD).

*imon*
(Input) INTEGER(2). Month (1 – 12).

*iday*
(Input) INTEGER(2). Day (1 – 31).

*ihr*
(Input) INTEGER(2). Hour (0 – 23)

*imin*
(Input) INTEGER(2). Minute (0 – 59)

*isec*
(Input) INTEGER(2). Second (0 – 59)

The packed time is the number of seconds since 00:00:00 Greenwich mean time, January 1, 1970. Because packed time values can be numerically compared, you can use PACKTIMEQQ to work with relative date and time values. Use UNPACKTIMEQQ to unpack time information. SETFILETIMEQQ uses packed time.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS DLL LIB

**See Also:**  "UNPACKTIMEQQ", "SETFILETIMEQQ", "GETFILEINFOQQ", "TIMEF"

**Example**

USE IFPORT

INTEGER(2) year, month, day, hour, minute, second, &
    hund
INTEGER(4) timedate

CALL GETDAT (year, month, day)
CALL GETTIM (hour, minute, second, hund)
CALL PACKTIMEQQ (timedate, year, month, day, hour, &
    minute, second)

END
PASSDIRKEYSQQ

QuickWin Function: Determines the behavior of direction and page keys in a QuickWin application. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

\[ \text{result} = \text{PASSDIRKEYSQQ}(\text{val}) \]

\(\text{val}\) (Input) INTEGER(4) or LOGICAL(4).

A value of .TRUE. causes direction and page keys to be input as normal characters (the PassDirKeys flag is turned on). A value of .FALSE. causes direction and page keys to be used for scrolling.

The following constants, defined in `IFQWIN.F90`, can be used as integer arguments:

- **PASS_DIR_FALSE** – Turns off any special handling of direction keys. They are not passed to the program by GETCHARQQ.
- **PASS_DIR_TRUE** – Turns on special handling of direction keys. That is, they are passed to the program by GETCHARQQ.
- **PASS_DIR_INSDEL** – INSERT and DELETE are also passed to the program by GETCHARQQ.
- **PASS_DIR_CNTRLC** – Only needed for a QuickWin application, but harmless if used with a Standard Graphics application that already passes CTRL+C.
  
  This value allows CTRL+C to be passed to a QuickWin program by GETCHARQQ if the following is true: the program must have removed the File menu EXIT item by using DELETEMENUQQ.

  This value also passes direction keys and INSERT and DELETE.

Results:

The return value indicates the previous setting of the PassDirKeys flag.

The return data type is the same as the data type of \text{val}; that is, either INTEGER(4) or LOGICAL(4).

When the PassDirKeys flag is turned on, the mouse must be used for scrolling since the direction and page keys are treated as normal input characters.

The PASSDIRKEYSQQ function is meant to be used primarily with the GETCHARQQ and INCHARQQ functions. Do not use normal input statements (such as READ) with the PassDirKeys flag turned on, unless your program is prepared to interpret direction and page keys.
Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETCHARQQ”, “INCHARQQ”

Examples

use IFQWIN
logical*4 res
character*1 ch, ch1
Print *,"Type X to exit, S to scroll, D to pass Direction keys"
123 continue
ch = getcharqq()
! check for escapes
! 0x00 0x?? is a function key
! 0xE0 0x?? is a direction key
if (ichar(ch) .eq. 0) then
  ch1 = getcharqq()
  print *,"function key follows escape = ",ichar(ch), " ",ichar(ch1)," ",ch1
goto 123
else if (ichar(ch) .eq. 224) then
  ch1 = getcharqq()
  print *,"direction key follows escape = ",ichar(ch)," ",ichar(ch1)," ",ch1
goto 123
else
  print *,ichar(ch)," ",ch
  if(ch .eq. ’S’) then
    res = passdirkeysqq(.false.)
    print *, "Entering Scroll mode ",res
  endif
  if(ch .eq. ’D’) then
    res = passdirkeysqq(.true.)
    print *, "Entering Direction keys mode ",res
  endif
  if(ch .ne. ’X’) go to 123
endif
end

The following example uses an integer constant as an argument to PASSDIRKEYSQQQ:

c=======================================================================
c
Program to illustrate how to get almost every character from the keyboard in QuickWin or Standard Graphics mode. Comment out the deletemenu line for Standard Graphics mode.

If you are doing a standard graphics application, control C will come in as a Z’03’ without further effort.

In a QuickWin application, The File menu Exit item must be deleted, and PassDirKeysQQ called with PASS_DIR_CNTRLC to get control C.
c===============================================================
10 key1 = getcharqq()
c-----first check for control+c
   if(ichar(key1) .eq. 3) then
      write(*,*) 'Control C Received'
      write(*,*) "Really want to quit?"
      write(*,*) "Type Y <cr> to exit, or any other char <cr> to continue."
      read(*,*) ch1
      if(ch1.eq."y" .or. ch1.eq."Y") goto 30
      goto 10
   endif
   if(ichar(key1).eq.0) then ! function key?
      key2 = getcharqq()
      write(*,15) ichar(key1),ichar(key2),key2
      15 format(1x,2i12,1x,a1,' function key')
   else
      if(ichar(key1).eq.224) then ! direction key?
         key2 = getcharqq()
         write(*,16) ichar(key1),ichar(key2),key2
         16 format(1x,2i12,1x,a1,' direction key')
      else
         write(*,20) key1,ichar(key1)  ! normal key
         20 format(1x,a1,i11)
      endif
   endif
   go to 10
30 stop
end

**PEEKCHARQQ**

**Run-time Function:** 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.

**Module:** USE IFCORE

**Syntax**

```plaintext
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..

Compatibility

See Also: “GETCHARQQ”, “GETSTROQ”, “FGETC”, “GETC”

Example

USE IFCORE
LOGICAL(4) pressed / .FALSE. / 

DO WHILE (.NOT. pressed)
    WRITE(*,*) ' Press any key'
    pressed = PEEKCHARQQ ( )
END DO 

PERROR

Run-Time Subroutine: Sends a message to the standard error stream, preceded by a specified string, for the last detected error.

Module: USE IFCORE

Syntax

CALL PERROR (string)

string
(Input) Character*. Message to precede the standard error message.

The string sent is the same as that given by GERROR.

Compatibility

See Also: “GERROR”, “IERRNO”
Example

USE IFCORE
character*24 errtext
errtext = 'In my opinion, '
  . .
! any error message generated by errtext is
! preceded by 'In my opinion, '
Call PERROR (errtext)

PIE, PIE_W

Graphics Functions: Draw a pie-shaped wedge in the current graphics color. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = PIE (i, x1, y1, x2, y2, x3, y3, x4, y4)
result = PIE_W (i, wx1, wy1, wx2, wy2, wx3, wy3, wx4, wy4)

i
(Input) INTEGER(2). Fill flag. One of the following symbolic constants (defined in IFQWIN.F90):
• SGFILLINTERIOR – Fills the figure using the current color and fill mask.
• SGBORDER – 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.

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.

**Results:**
The result type is INTEGER(2). The result is nonzero if successful; otherwise, 0. If the pie is
cropped 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.

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 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.

**NOTE.** The PIE routine described here is a QuickWin routine. If you are
trying to use the Microsoft® Platform SDK version of the Pie routine by
including the IFWIN module, you need to specify the routine name as
MSFWINSPie. For more information, see "Special Naming Convention for
Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB


Example
! build as Graphics App.
USE IFQWIN
INTEGER(2) status, dummy
INTEGER(2) x1, y1, x2, y2, x3, y3, x4, y4
x1 = 80; y1 = 50
x2 = 180; y2 = 150
x3 = 110; y3 = 80
x4 = 90; y4 = 180
status = SETCOLOR(INT2(4))
dummy = PIE ($GFILLINTERIOR, x1, y1, x2, y2, &
                x3, y3, x4, y4)
END

The following figure shows the coordinates used to define PIE and PIE_W:

![Coordinate Diagram]

**POLYBEZIER, POLYBEZIER_W**

**Graphics Functions:** Draw one or more Bezier curves. These functions are only available on Windows® systems.

**Module:** USE IFQWIN
Syntax

result = POLYBEZIER (ppoints, cpoints)
result = POLYBEZIER_W (wppoints, cpoints)

ppoints
(Input) Derived type xycoord. Array of derived types defining the endpoints and the control points for each Bezier curve. The derived type xycoord is defined in IFQWIN.F90 as follows:

```
TYPE xycoord
  INTEGER(2) xcoord
  INTEGER(2) ycoord
END TYPE xycoord
```

cpoints
(Input) INTEGER(2). Number of points in ppoints or wppoints.

wppoints
(Input) Derived type wxycoord. Array of derived types defining the endpoints and the control points for each Bezier curve. The derived type wxycoord is defined in IFQWIN.F90 as follows:

```
TYPE wxycoord
  REAL(8) wx
  REAL(8) wy
END TYPE wxycoord
```

Results:
The result type is INTEGER(2). The result is nonzero if anything is drawn; otherwise, 0.

A Bezier curve is based on fitting a cubic curve to four points. The first point is the starting point, the next two points are control points, and last point is the ending point. The starting point must be given for the first curve; subsequent curves use the ending point of the previous curve as their starting point. So, cpoints should contain 4 for one curve, 7 for 2 curves, 10 for 3 curves, and so forth.

POLYBEZIER does not use or change the current graphics position.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “POLYBEZIERTO, POLYBEZIERTO_W”

Example
Program Bezier
use IFQWIN
! Shows how to use POLYBEZIER, POLYBEZIER_W,
! POLYBEZIERTO, and POLYBEZIERTO_W,

TYPE(xycoord)  lppoints(31)
TYPE(wxycoord) wlppoints(31)
TYPE(xycoord)  xy
TYPE(wxycoord) wxy

integer(4)  i
integer(2)  istat, orgx, orgy
real(8)    worgx, worgy

i = setcolorrgb(Z'00FFFFFF') ! graphic to black
i = settextcolorrgb(Z'00FFFFFF') ! text to black
i = setbkcolorrgb(Z'00000000') ! background to white
call clearscreen($GCLEARSCREEN)

orgx = 20
orgy = 20

lppoints(1).xcoord = 1+orgx
lppoints(1).ycoord = 1+orgy
lppoints(2).xcoord = 30+orgx
lppoints(2).ycoord = 120+orgy
lppoints(3).xcoord = 150+orgx
lppoints(3).ycoord = 60+orgy
lppoints(4).xcoord = 180+orgx
lppoints(4).ycoord = 180+orgy

istat = PolyBezier(lppoints, 4)

! Show tangent lines
! A bezier curve is tangent to the line
! from the begin point to the first control
! point.  It is also tangent to the line from
! the second control point to the end point.
do i = 1,4,2
call moveto(lppoints(i).xcoord,lppoints(i).ycoord,xy)
  istat = lineto(lppoints(i+1).xcoord,lppoints(i+1).ycoord)
end do
read(*,*)
worgx = 50.0
worgy = 50.0
wlppoints(1).wx = 1.0+worgx
wlppoints(1).wy = 1.0+worgy
wlppoints(2).wx = 30.0+worgx
wlppoints(2).wy = 120.0+worgy
wlppoints(3).wx = 150.0+worgx
wlppoints(3).wy = 60.0+worgy
wlppoints(4).wx = 180.0+worgx
wlppoints(4).wy = 180.0+worgy
i = setcolorrgb(Z'000000FF') ! graphic to red
istat = PolyBezier_W(wlppoints, 4)

! Show tangent lines
! A bezier curve is tangent to the line
! from the begin point to the first control
! point. It is also tangent to the line from
! the second control point to the end point.
do i = 1,4,2
    call moveto_w(wlppoints(i).wx,wlppoints(i).wy,wxy)
    istat = lineto_w(wlppoints(i+1).wx,wlppoints(i+1).wy)
end do
read(*,*)
orgx = 80
orgy = 80

! POLYBEZIERTO uses the current graphics position
! as its initial starting point so we start the
! array with the first first control point.
! lppoints(1).xcoord = 1+orgx ! need to move to this
! lppoints(1).ycoord = 1+orgy
lppoints(1).xcoord = 30+orgx
lppoints(1).ycoord = 120+orgy

lppoints(2).xcoord = 150+orgx
lppoints(2).ycoord = 60+orgy

lppoints(3).xcoord = 180+orgx
lppoints(3).ycoord = 180+orgy

i = setcolorrgb(Z'0000FF00') ! graphic to green
call moveto(1+orgx,1+orgy,xy)

istat = PolyBezierTo(lppoints, 3)

! Show tangent lines
! A bezier curve is tangent to the line
! from the begin point to the first control
! point. It is also tangent to the line from
! the second control point to the end point.
call moveto(1+orgx,1+orgy,xy)

istat = lineto(lppoints(1).xcoord,lppoints(1).ycoord)
call moveto(lppoints(2).xcoord,lppoints(2).ycoord,xy)

istat = lineto(lppoints(3).xcoord,lppoints(3).ycoord)

read(*,*)

worgx = 110.0
worgy = 110.0

! wlppoints(1).wx = 1.0+worgx
! wlppoints(1).wy = 1.0+worgy

wlppoints(1).wx = 30.0+worgx
wlppoints(1).wy = 120.0+worgy

wlppoints(2).wx = 150.0+worgx
wlppoints(2).wy = 60.0+worgy

wlppoints(3).wx = 180.0+worgx
wlppoints(3).wy = 180.0+worgy

call moveto_w(1.0+worgx,1.0+worgy,wxy)
i = setcolorrgb(Z'00FF0000') ! graphic to blue

istat = PolyBezierTo_w(wlppoints, 3)
! Show tangent lines
! A bezier curve is tangent to the line
! from the begin point to the first control
! point. It is also tangent to the line from
! the second control point to the end point.
call moveto_w(1.0+worgx,1.0+worgy,wxy)
  istat = lineto_w(wlppoints(1).wx,wlppoints(1).wy)
call moveto_w(wlppoints(2).wx,wlppoints(2).wy,wxy)
  istat = lineto_w(wlppoints(3).wx,wlppoints(3).wy)
read(*,*)

END PROGRAM Bezier

**POLYBEZIERTO, POLYBEZIERTO_W**

**Graphics Functions:** Draw one or more Bezier curves. These functions are only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

result = POLYBEZIERTO (ppoints, cpoints)
result = POLYBEZIERTO_W (wppoints, cpoints)

**ppoints**

(Input) Derived type xycoord. Array of derived types defining the endpoints and the control points for each Bezier curve. The derived type xycoord is defined in IFQWIN.F90 as follows:

```fortran
TYPE xycoord
  INTEGER(2) xcoord
  INTEGER(2) ycoord
END TYPE xycoord
```

**cpoints**

(Input) INTEGER(2). Number of points in ppoints or wppoints.

**wppoints**

(Input) Derived type wxycoord. Array of derived types defining the endpoints and the control points for each Bezier curve. The derived type wxycoord is defined in IFQWIN.F90 as follows:
TYPE wxycoord
    REAL(8) wx
    REAL(8) wy
END TYPE wxycoord

Results:
The result type is INTEGER(2). The result is nonzero if anything is drawn; otherwise, 0.
A Bezier curve is based on fitting a cubic curve to four points. The first point is the starting point,
the next two points are control points, and last point is the ending point. The starting point is the
current graphics position as set by MOVETO for the first curve; subsequent curves use the ending
point of the previous curve as their starting point. So, cpoints should contain 3 for one curve, 6 for
2 curves, 9 for 3 curves, and so forth.

POLYBEZIERTO moves the current graphics position to the ending point of the last curve drawn.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “POLYBEZIER, POLYBEZIER_W”, “MOVETO, MOVETO_W”

Example
See the example in “POLYBEZIER, POLYBEZIER_W”.

POLYGON, POLYGON_W

Graphics Functions: Draw a polygon using the current graphics color, logical write mode, and
line style. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax
    result = POLYGON (control, ppoints, cpoints)
    result = POLYGON_W (control, wppoints, cpoints)

control
(Input) INTEGER(2). Fill flag. One of the following symbolic constants (defined in
IFQWIN.F90):
• $GFILLINTERIOR – Draws a solid polygon using the current color and fill mask.
• $GBORDER – Draws the border of a polygon using the current color and line style.

ppoints
(Input) Derived type wxycoord. Array of derived types defining the polygon vertices in viewport
coordinates. The derived type wxycoord is defined in IFQWIN.F90 as follows:
TYPE xycoord
    INTEGER(2) xcoord
    INTEGER(2) ycoord
END TYPE xycoord

cpoints
(Input) INTEGER(2). Number of polygon vertices.

wppoints
(Input) Derived type wxycoord. Array of derived types defining the polygon vertices in window coordinates. The derived type wxycoord is defined in IFQWIN.F90 as follows:

TYPE wxycoord
    REAL(8) wx
    REAL(8) wy
END TYPE wxycoord

Results:
The result type is INTEGER(2). The result is nonzero if anything is drawn; otherwise, 0.

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 routine uses the viewport-coordinate system (expressed in xycoord derived types), and the POLYGON_W routine uses real-valued window coordinates (expressed in wxycoord types).

The arguments ppoints and wppoints are arrays whose elements are xycoord or wxycoord derived types. Each element specifies one of the polygon’s vertices. The argument cpoints is the number of elements (the number of vertices) in the ppoints or wppoints 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.
NOTE. The POLYGON routine described here is a QuickWin routine. If you are trying to use the Microsoft® Platform SDK version of the Polygon routine by including the IFWIN module, you need to specify the routine name as MSFWIN$Polygon. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB


Example
! Build as a Graphics App.
!
! Draw a skeletal box
   USE IFQWIN
   INTEGER(2) status
   TYPE (xycoord) poly(12)
   ! Set up box vertices in order they will be drawn, &
   ! repeating some to avoid unwanted lines across box
   poly(1)%xcoord = 50
   poly(1)%ycoord = 80
   poly(2)%xcoord = 85
   poly(2)%ycoord = 35
   poly(3)%xcoord = 185
   poly(3)%ycoord = 35
   poly(4)%xcoord = 150
   poly(4)%ycoord = 80
   poly(5)%xcoord = 50
   poly(5)%ycoord = 80
   poly(6)%xcoord = 50
   poly(6)%ycoord = 180
   poly(7)%xcoord = 150
   poly(7)%ycoord = 180
   poly(8)%xcoord = 185
poly(8)%ycoord = 135
poly(9)%xcoord = 185
poly(9)%ycoord = 35
poly(10)%xcoord = 150
poly(10)%ycoord = 80
poly(11)%xcoord = 150
poly(11)%ycoord = 180
poly(12)%xcoord = 150
poly(12)%ycoord = 80
status = SETCOLORRGB(Z'0000FF')
status = POLYGON($GBORDER, poly, INT2(12))
END

**POLYLINEQQ**

**Graphics Function:** Draws a line between each successive x, y point in a given array. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```
result = POLYLINEQQ (points, cnt)
```

`points`  
(Input) An array of `DF_POINT` objects. The derived type `DF_POINT` is defined in `IFQWIN.F90` as:

```
type DF_POINT
  sequence
    integer(4) x
    integer(4) y
end type DF_POINT
```

`cnt`  
(Input) INTEGER(4). Number of elements in the `points` array.

**Results:**

The result type is INTEGER(4). The result is a nonzero value if successful; otherwise, zero.

POLYLINEQQ uses the viewport-coordinate system.
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`.

The current graphics position is not used or changed as it is in the `LINETO` function.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS

**See Also:** “LINETO, LINETO_W”, “LINETOAREX”, “SETCOLORRGB”, “SETLINESTYLE”, “SETWRITEMODE”

**Example**

```fortran
! Build for QuickWin or Standard Graphics
USE IFQWIN
TYPE(DF_POINT) points(12)
integer(4) result
integer(4) cnt, i
! load the points
do i = 1,12,2
    points(i).x =20*i
    points(i).y =10
    points(i+1).x =20*i
    points(i+1).y =60
end do
! A sawtooth pattern will appear in the upper left corner
result = POLYLINEQQ(points, 12)
end
```

**PUTC**

**Portability Function:** Writes a character to Fortran external unit 6.

**Module:** USE IFPORT

**Syntax**

```fortran
result = PUTC (char)
```

**char**

(Input) Character. Character to be written to external unit 6.

**Results:**

The result type is INTEGER(4). The result is zero if successful; otherwise, an error code.
**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “GETC”. “FPUTC”, WRITE and PRINT in the Language Reference

**Example**

```fortran
use IFPORT
integer(4) i4
character*1 char1
do i = 1,26
   char1 = char(123-i)
   i4 = putc(char1)
   if (i4.ne.0) iflag = 1
enddo
```

**PUTIMAGE, PUTIMAGE_W**

**Graphics Subroutines:** Transfer the image stored in memory to the screen. These subroutines are only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
CALL PUTIMAGE (x, y, image, action)
CALL PUTIMAGE_W (wx, wy, image, action)
```

- **x, y**
  (Input) INTEGER(2). Viewport coordinates for upper-left corner of the image when placed on the screen.

- **wx, wy**
  (Input) REAL(8). Window coordinates for upper-left corner of the image when placed on the screen.

- **image**

- **action**
  (Input) INTEGER(2). Interaction of the stored image with the existing screen image. One of the following symbolic constants (defined in IFQWIN.F90):
• $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 Windows* API BitBlt):
  — $GSRCCOPY (same as $GPSET)
  — $GSRCPAINT (same as $GOR)
  — $GSRCAND (same as $GAND)
  — $GSRCINVERT (same as $GXOR)
  — $GSRCERASE
  — $GNOTSRCCOPY (same as $GPRESET)
  — $GNOTSRCPAINT
  — $GMERGECOPY
  — $GMERGEPAINT
  — $GPATCOPY
  — $GPATPAINT
  — $GPATINVERT
  — $GDSTINVERT
  — $GBLACKNESS
  — $GWHITENESS

PUTIMAGE places the upper-left corner of the image at the viewport coordinates (x, y).
PUTIMAGE_W places the upper-left corner of the image at the window coordinates (wx, wy).
Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: "GETIMAGE, GETIMAGE_W", "GRSTATUS", "IMAGESIZE, IMAGESIZE_W"

Example

! Build as a Graphics App.
USE IFQWIN
INTEGER(1), ALLOCATABLE :: buffer(:)
INTEGER(2) status, x
INTEGER(4) imsize

status = SETCOLOR(INT2(4))
! draw a circle
status = ELLIPSE(\$GFIILLINTERIOR,INT2(40),INT2(55), &
    INT2(70),INT2(85))
imsize = IMAGESIZE (INT2(39),INT2(54),INT2(71), &
    INT2(86))
ALLOCATE (buffer(imsize))
CALL GETIMAGE(INT2(39),INT2(54),INT2(71),INT2(86), &
    buffer)
! copy a row of circles beneath it
DO x = 5 , 395, 35
    CALL PUTIMAGE(x, INT2(90), buffer, \$GPSET)
END DO
DEALLOCATE(buffer)
END

PXFTYPEGET

POSIX Subroutine: Gets the value stored in a component (or field) of a structure.
Module: USE IFPOSIX
Syntax
    CALL PXFTYPEGET (jhandle, compname, value, ierror)
    CALL PXFTYPEGET (jhandle, compname, value, ilen, ierror) !when <TYPE> is STR
<TYPE>
A placeholder for one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Routine Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INTEGER(4)</td>
<td>PXFINTGET</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL(4)</td>
<td>PXFREALGET</td>
</tr>
<tr>
<td>LGCL</td>
<td>LOGICAL(4)</td>
<td>PXFLGCLGET</td>
</tr>
<tr>
<td>STR</td>
<td>CHARACTER*(*)</td>
<td>PXFSTRGET</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHARACTER(1)_</td>
<td>PXFCHARGET</td>
</tr>
<tr>
<td>DBL</td>
<td>REAL(8)</td>
<td>PXFDBLGET</td>
</tr>
<tr>
<td>INT8</td>
<td>INTEGER(8)</td>
<td>PXFINT8GET</td>
</tr>
</tbody>
</table>

**jhandle**
(Input) INTEGER(4). A handle of a structure.

**compname**
(Input) Character. The name of the component (or field) of the structure to retrieve data from.

**value**
(Output) A variable, whose data type depends on the value of <TYPE>. See the table above for the data types for each value; for example, if the value for <TYPE> is INT, the data type is INTEGER(4). Stores the value of the component (or field).

**ilen**
(Output) INTEGER(4). This argument can only be used when <TYPE> is STR (PXFSTRGET). Stores the length of the returned string.

**ierror**
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXF<TYPE>GET subroutines retrieve the value from component (or field) compname of the structure associated with handle jhandle into variable value.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “PXF<TYPE>SET”

**Example**
See the example in “PXFTIMES” (which demonstrates PXFINTGET and PXFINT8GET)
**PXF< TYPE >SET**

**POSIX Subroutine:** Sets the value of a component (or field) of a structure.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXF< TYPE >SET (jhandle, compname, value, ierr)
CALL PXF< TYPE >SET (jhandle, compname, value, ilen, ierr) !when < TYPE > is STR
```

**< TYPE >**

A placeholder for one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Routine Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INTEGER(4)</td>
<td>PXFINTSET</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL(4)</td>
<td>PXFREALSET</td>
</tr>
<tr>
<td>LGCL</td>
<td>LOGICAL(4)</td>
<td>PXFLGCLSET</td>
</tr>
<tr>
<td>STR</td>
<td>CHARACTER(*)</td>
<td>PXFSTRSET</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHARACTER(1)</td>
<td>PXFCHARSET</td>
</tr>
<tr>
<td>DBL</td>
<td>REAL(8)</td>
<td>PXFDBLSET</td>
</tr>
<tr>
<td>INT8</td>
<td>INTEGER(8)</td>
<td>PXFINT8SET</td>
</tr>
</tbody>
</table>

**jhandle**

(Input) INTEGER(4). A handle of a structure.

**compname**

(Input) Character. The name of the component (or field) of the structure to write data to.

**value**

(Input) A variable, whose data type depends on the value of < TYPE >. See the table above for the data types for each value; for example, if the value for < TYPE > is INT, the data type is INTEGER(4). The value for the component (or field).

**ilen**

(Input) INTEGER(4). This argument can only be used when < TYPE > is STR (PXFSTRSET). The length of the string in value.

**ierror**

(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.
The PXF<TYPE>SET subroutines set or modify the value in component (or field) compname of the structure associated with handle jhandle from variable value.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also:  “PXF<TYPE>GET”

Example

See the example in “PXFSTRUCTCREATE” (which demonstrates PXFSTRSET).

**PXFA<TYPE>GET**

**POSIX Subroutine:**  Gets the array values stored in a component (or field) of a structure.

**Module:**  USE IFPOSIX

**Syntax**

CALL PXFA<TYPE>GET (jhandle, compname, value, ialen, ierror)

CALL PXFA<TYPE>GET (jhandle, compname, value, ialen, ilen, ierror)  ! when <TYPE> is STR

A placeholder for one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Routine Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INTEGER(4)</td>
<td>PXAFINTGET</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL(4)</td>
<td>PXFAREALGET</td>
</tr>
<tr>
<td>LGCL</td>
<td>LOGICAL(4)</td>
<td>PXFALGCLGET</td>
</tr>
<tr>
<td>STR</td>
<td>CHARACTER*(*</td>
<td>PXFASTRGET</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHARACTER(1)</td>
<td>PXFACHARGET</td>
</tr>
<tr>
<td>DBL</td>
<td>REAL(8)</td>
<td>PXFADBLGET</td>
</tr>
<tr>
<td>INT8</td>
<td>INTEGER(8)</td>
<td>PXFAINT8GET</td>
</tr>
</tbody>
</table>

**jhandle**

(Input) INTEGER(4).  A handle of a structure.

**compname**

(Input) Character. The name of the component (or field) of the structure to retrieve data from.
value
(Output) An array, whose data type depends on the value of <TYPE>. See the table above for the
data types for each value; for example, if the value for <TYPE> is INT, the data type of the array
is INTEGER(4). Stores the value of the component (or field).

ialen
(Input) INTEGER(4). The size of array value.

ilen
(Output) INTEGER(4). This argument can only be used when <TYPE> is STR (PXFASTRGET).
An array that stores the lengths of elements of array value.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFA<TYPE>GET subroutines are similar to the PXF<TYPE>GET subroutines, but they
should be used when the component (or field) of the structure is an array.
When the PXFA<TYPE>GET subroutines are used, the entire array is accessed (read from the
component or field) as a unit.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "PXFA<TYPE>SET", "PXF<TYPE>GET"

PXFA<TYPE>SET

POSIX Subroutine: Sets the value of an array component (or field) of a structure.

Module: USE IFPOSIX

Syntax

CALL PXFA<TYPE>SET (jhandle, compname, value, ialen, ierror)
CALL PXFA<TYPE>SET (jhandle, compname, value, ialen, ilen, ierror) ! when <TYPE> is STR

A placeholder for one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Routine Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INTEGER(4)</td>
<td>PXAFINTSET</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL(4)</td>
<td>PXFAREALSET</td>
</tr>
<tr>
<td>LGCL</td>
<td>LOGICAL(4)</td>
<td>PXFALGCLSET</td>
</tr>
</tbody>
</table>
**Descriptions of the Library Routines**

**Value** | **Data Type** | **Routine Name**  
---|---|---  
STR | CHARACTER*(*) | PXFASTRSET  
CHAR | CHARACTER(1) | PXFACHARSET  
DBL | REAL(8) | PXFADBLSET  
INT8 | INTEGER(8) | PXFAINT8SET  

**jhandle**  
(Input) INTEGER(4). A handle of a structure.

**compname**  
(Input) Character. The name of the component (or field) of the structure to write data to.

**value**  
(Input) An array, whose data type depends on the value of `<TYPE>`. See the table above for the data types for each value; for example, if the value for `<TYPE>` is INT, the data type of the array is INTEGER(4). The value for the component (or field).

**ialen**  
(Input) INTEGER(4). The size of array `value`.

**ilen**  
(Input) INTEGER(4). This argument can only be used when `<TYPE>` is STR (PXFASTRSET). An array that specifies the lengths of elements of array `value`.

**ierror**  
(Output) INTEGER(4). The error status.  
If successful, `ierror` is set to zero; otherwise, an error code.

The PXFA<TYPE>SET subroutines are similar to the PXF<TYPE>SET subroutines, but they should be used when the component (or field) of the structure is an array.

When the PXFA<TYPE>SET subroutines are used, the entire array is accessed (written to the component or field) as a unit.

**Compatibility**  
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “PXFA<TYPE>GET”, “PXF<TYPE>SET”

**PXFACCESS**

**POSIX Subroutine:** Determines the accessibility of a file.

**Module:** USE IFPOSIX
Syntax

CALL PXFACCESS (path, ilen, iamode, ierror)

path
(Input) Character. The name of the file.

ilen
(Input) INTEGER(4). The length of the path string.

iamode
(Input) INTEGER(4). One or more of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Checks for existence of the file.</td>
</tr>
<tr>
<td>1</td>
<td>Checks for execute permission.</td>
</tr>
<tr>
<td>2</td>
<td>Checks for write access.</td>
</tr>
<tr>
<td>4</td>
<td>Checks for read access.</td>
</tr>
<tr>
<td>6</td>
<td>Checks for read/write access.</td>
</tr>
</tbody>
</table>

1. L'X only

ierror
(Output) INTEGER(4). The error status.

If access is permitted, the result value is zero; otherwise, an error code. Possible error codes are:

- –1: A bad parameter was passed.
- ENOENT: The named directory does not exist.
- EACCES: Access requested was denied.

On Windows* systems, if the name given is a directory name, the function only checks for existence. All directories have read/write access on Windows systems.

Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFALARM

POSIX Subroutine: Schedules an alarm.

Module: USE IFPOSIX

Syntax

CALL PXFALARM (isecseconds, isecleft, ierror)
isec
(Input) INTEGER(4). The number of seconds before the alarm signal should be delivered.

isecleft
(Output) INTEGER(4). The number of seconds remaining until any previously scheduled alarm signal is due to be delivered. It is set to zero if there was no previously scheduled alarm signal.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

The PXFALARM subroutine arranges for a SIGALRM signal to be delivered to the process in seconds isec.

On Linux* systems, SIGALRM is a reserved defined constant that is equal to 14. You can use any other routine to install the signal handler. You can get SIGALRM and other signal values by using PXFCONST or IPXFCONST.

On Windows* systems, the SIGALRM feature is not supported, but the POSIX library has an implementation you can use. You can provide a signal handler for SIGALRM by using PXFSIGACTION.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFCONST”, “IPXFCONST”, “PXFSIGACTION”

PXFCALLSUBHANDLE

POSIX Subroutine: Calls the associated subroutine.
Module: USE IFPOSIX
Syntax
   CALL PXFCALLSUBHANDLE (jhandle2, ival, ierror)

jhandle2
(Input) INTEGER(4). A handle to the subroutine.
ival
(Input) INTEGER(4). The argument to the subroutine.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFCALLSUBHANDLE subroutine, when given a subroutine handle, calls the associated subroutine. PXFGETSUBHANDLE should be used to obtain a subroutine handle.

NOTE. The subroutine cannot be a function, an intrinsic, or an entry point, and must be defined with exactly one integer argument.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “PXFGETSUBHANDLE”

PXFCFGETISPEED

POSIX Subroutine: Returns the input baud rate from a termios structure. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
CALL PXFCFGETISPEED (jtermios, iospeed, ierror)

jtermios
(Input) INTEGER(4). A handle of structure termios.

iospeed
(Output) INTEGER(4). The returned value of the input baud rate from the structure associated with handle jtermios.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

NOTE. To get a handle for an instance of the termios structure, use PXFSTRUCTCREATE with the string 'termios' for the structure name.

See Also: “PXFSTRUCTCREATE”, “PXFCFSETISPEED”
**PXFCFGETOSPEED**

**POSIX Subroutine:** Returns the output baud rate from a `termios` structure. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFCFGETOSPEED (jtermios, iospeed, ierror)
```

- **jtermios** (Input) INTEGER(4). A handle of structure `termios`.
- **iospeed** (Output) INTEGER(4). The returned value of the output baud rate from the structure associated with handle `jtermios`.
- **ierror** (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

**NOTE.** To get a handle for an instance of the `termios` structure, use `PXFSTRUCTCREATE` with the string 'termios' for the structure name.

See Also: “PXFSTRUCTCREATE”, “PXFCFSETOSPEED”

**PXFCFSETISPEED**

**POSIX Subroutine:** Sets the input baud rate in a `termios` structure. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFCFSETISPEED (jtermios, ispeed, ierror)
```

- **jtermios** (Input) INTEGER(4). A handle of structure `termios`.
- **ispeed** (Input) INTEGER(4). The value of the input baud rate for the structure associated with handle `jtermios`.

See Also: “PXFSTRUCTCREATE”, “PXFCFSETOSPEED”
\textit{ierror}  
(Output) INTEGER(4). The error status. 
If successful, \textit{ierror} is set to zero; otherwise, an error code.

\textbf{NOTE.} To get a handle for an instance of the \texttt{termios} structure, use \texttt{PXFSTRUCTCREATE} with the string 'termios' for the structure name.

\textbf{See Also:} \texttt{"PXFSTRUCTCREATE", \"PXFCFGETOSPEED\"}

\textbf{PXFCFSETOSPEED}

\textbf{POSIX Subroutine:} Sets the output baud rate in a \texttt{termios} structure. This subroutine is only available on Linux* systems.  
\textbf{Module:} USE IFPOSIX  
\textbf{Syntax}  
CALL PXFCFSETOSPEED \texttt{(jtermios, ispeed, ierror)}  
\textit{jtermios}  
(Input) INTEGER(4). A handle of structure \texttt{termios}.  
\textit{ispeed}  
(Input) INTEGER(4). The value of the output baud rate for the structure associated with handle \textit{jtermios}.  
\textit{ierror}  
(Output) INTEGER(4). The error status. 
If successful, \textit{ierror} is set to zero; otherwise, an error code.

\textbf{NOTE.} To get a handle for an instance of the \texttt{termios} structure, use \texttt{PXFSTRUCTCREATE} with the string 'termios' for the structure name.

\textbf{See Also:} \texttt{"PXFSTRUCTCREATE", \"PXFCFGETOSPEED\"}

\textbf{PXFCHDIR}

\textbf{POSIX Subroutine:} Changes the current working directory.
Module: USE IFPOSIX
Syntax
   CALL PXFCHDIR (path, ilen, ierror)
path   (Input) Character. The directory to be changed to.
ilen   (Input) INTEGER(4). The length of the path string.
ierror (Output) INTEGER(4). The error status.
   If successful, ierror is set to zero; otherwise, an error code.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “PXFMKDIR”

PXFCHMOD
POSIX Subroutine: Changes the ownership mode of the file.
Module: USE IFPOSIX
Syntax
   CALL PXFCHMOD (path, ilen, imode, ierror)
path   (Input) Character. The path to the file.
ilen   (Input) INTEGER(4). The length of the path string.
imode  (Input) INTEGER(4). The ownership mode of the file. On Windows* 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 mode.
ierror (Output) INTEGER(4). The error status.
   If successful, ierror is set to zero; otherwise, an error code.
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.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFCCHOWN

POSIX Subroutine: Changes the owner and group of a file. This subroutine is only available on Linux* systems.
Module: USE IFPOSIX
Syntax
   CALL PXFCCHOWN (path, ilen, iowner,igroup, ierror)
path
(Input) Character. The file or directory name.
ilen
(Input) INTEGER(4). The length of the path string.
iowner
(Input) INTEGER(4). The owner UID.
igroup
(Input) INTEGER(4). The group GID.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

PXFCLEARENV

POSIX Subroutine: Clears the process environment.
Module: USE IFPOSIX
Syntax
   CALL PXFCLEARENV (ierror)
**PXFCLOSE**

**POSIX Subroutine:** Closes the file associated with the descriptor.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFCLOSE (fd, ierror)
```

- **fd** (Input) INTEGER(4). A file descriptor.
- **ierror** (Output) INTEGER(4). The error status.
  If successful, **ierror** is set to zero; otherwise, an error code.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFOPEN”

**PXFCLOSEDIR**

**POSIX Subroutine:** Closes the directory stream.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFCLOSEDIR (idirid, ierror)
```

- **idirid** (Input) INTEGER(4). The directory ID obtained from PXFOPENDIR.
- **ierror** (Output) INTEGER(4). The error status.
  If successful, **ierror** is set to zero; otherwise, an error code.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
The PXFCLOSEDIR subroutine closes the directory associated with idirid.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFOPENDIR”

PXFCNTL

**POSIX Subroutine:** Manipulates an open file descriptor. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFCNTL (ifildes, icmd, iargin, iargout, ierror)
```

- **ifildes** (Input) INTEGER(4). A file descriptor.
- **icmd** (Input) INTEGER(4). Defines an action for the file descriptor.
- **iargin** (Input; output) INTEGER(4). Interpretation of this argument depends on the value of icmd.
- **iargout** (Output) INTEGER(4). Interpretation of this argument depends on the value of icmd.
- **ierror** (Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

PXFFCNCTL is a multi-purpose subroutine that causes an action to be performed on a file descriptor. The action, defined in icmd, can be obtained by using the values of predefined macros in C header fcntl.h, or by using PXFCONST or IPXFCONST with one of the following constant names:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_DUPFD</td>
<td>Returns into iargout the lowest available unopened file descriptor greater than or equal to iargin. The new file descriptor refers to the same open file as ifildes and shares any locks. The system flag FD_CLOEXEC for the new file descriptor is cleared so the new descriptor will not be closed on a call to PXFEXEC subroutine.</td>
</tr>
</tbody>
</table>
Descriptions of the Library Routines

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_GETFD</td>
<td>Returns into iargout the value of system flag FD_CLOEXEC associated with ifildes. In this case, iargin is ignored.</td>
</tr>
<tr>
<td>F_SETFD</td>
<td>Sets or clears the system flag FD_CLOEXEC for file descriptor ifildes. The PXFEXEC family of functions will close all file descriptors with the FD_CLOEXEC flag set. The value for FD_CLOEXEC is obtained from argument iargin.</td>
</tr>
<tr>
<td>F_GETFL</td>
<td>Returns the file status flags for file descriptor ifildes. Unlike F_GETFD, these flags are associated with the file and shared by all descriptors. A combination of the following flags, which are symbolic names for PXFCONST or IPXFCONST, can be returned:</td>
</tr>
<tr>
<td></td>
<td>• O_APPEND – Specifies the file is opened in append mode.</td>
</tr>
<tr>
<td></td>
<td>• O_NONBLOCK – Specifies when the file is opened, it does not block waiting for data to become available.</td>
</tr>
<tr>
<td></td>
<td>• O_RDONLY – Specifies the file is opened for reading only.</td>
</tr>
<tr>
<td></td>
<td>• O_RDWR – Specifies the file is opened for both reading and writing.</td>
</tr>
<tr>
<td></td>
<td>• O_WRONLY – Specifies the file is opened for writing only.</td>
</tr>
<tr>
<td>F_SETFL</td>
<td>Sets the file status flags from iargin for file descriptor ifildes. Only O_APPEND or O_NONBLOCK flags can be modified. In this case, iargout is ignored.</td>
</tr>
<tr>
<td>F_GETLK</td>
<td>Gets information about a lock. Argument iargin must be a handle of structure flock. This structure is taken as the description of a lock for the file. If there is a lock already in place that would prevent this lock from being locked, it is returned to the structure associated with handle iargin. If there are no locks in place that would prevent the lock from being locked, field l_type in the structure is set to the value of the constant with symbolic name F_UNLCK.</td>
</tr>
<tr>
<td>F_SETLK</td>
<td>Sets or clears a lock. Argument iargin must be a handle of structure flock. The lock is set or cleared according to the value of structure field l_type. If the lock is busy, an error is returned.</td>
</tr>
<tr>
<td>F_SETLKW</td>
<td>Sets or clears a lock, but causes the process to wait if the lock is busy. Argument iargin must be a handle of structure flock. The lock is set or cleared according to the value of structure field l_type. If the lock is busy, PXFCNTL waits for an unlock.</td>
</tr>
</tbody>
</table>

NOTE. To get a handle for an instance of the flock structure, use PXFSTRUCTCREATE with the string 'flock' for the structure name.

See Also: “PXFSTRUCTCREATE”, “IPXFCONST”, “PXFCONST”
PXFCONST

POSIX Subroutine: Returns the value associated with a constant.
Module: USE IFPOSIX
Syntax
   CALL PXFCONST (constname, ival, ierror)

constname
(Input) Character. The name of one of the following constants:
• STDIN_UNIT
• STDOUT_UNIT
• STDERR_UNIT
• EINV AL
• ENONAME
• ENOHANDLE
• EARRAYLEN
The constants beginning with E signify various error values for the system variable errno.
ival
(Output) INTEGER(4). The returned value of the constant.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
For more information on these constants, see your Microsoft Visual C++ documentation
(Windows® systems) or the errno.h file (Linux® systems).

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
See Also: “PXFISCONST”

PXFCREAT

POSIX Subroutine: Creates a new file or rewrites an existing file.
Module: USE IFPOSIX
Syntax
   CALL PXFCREAT (path, ilen, imode, ifildes, ierror)
path
(Input) Character. The pathname of the file.

ilen
(Input) INTEGER(4). The length of path string.

imode
(Input) INTEGER(4). The mode of the newly created file. On Windows* systems, see your Microsoft* Visual C++ documentation for permitted mode values. On Linux* systems, use octal file-access mode.

ifildes
(Output) INTEGER(4). The file descriptor.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFCTERMID

POSIX Subroutine: Generates a terminal pathname. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
CALL PXFCTERMID (s, ilen, ierror)

s
(Output) Character. The returned pathname of the terminal.

ilen
(Output) INTEGER(4). The length of the returned value in the s string.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
This subroutine returns a string that refers to the current controlling terminal for the current process.
PXFDUP, PXFDUP2

**POSIX Subroutine:** Duplicates an existing file descriptor.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFDUP (ifildes, ifid, ierror)
CALL PXFDUP2 (ifildes, ifildes2, ierror)
```

- **ifildes**
  - (Input) INTEGER(4). The file descriptor to duplicate.

- **ifid**
  - (Output) INTEGER(4). The returned new duplicated file descriptor.

- **ifildes2**
  - (Output) INTEGER(4). The number for the new file descriptor.

- **ierror**
  - (Output) INTEGER(4). The error status.

If successful, **ierror** is set to zero; otherwise, an error code.

The PXFDUP subroutine creates a second file descriptor for an opened file.

The PXFDUP2 subroutine copies the file descriptor associated with **ifildes**. Integer number **ifildes2** becomes associated with this new file descriptor, but the value of **ifildes2** is not changed.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFE<TYPE>GET

**POSIX Subroutine:** Gets the value stored in an array element component (or field) of a structure.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFE<TYPE>GET (jhandle, compname, index, value, ierror)
CALL PXFE<TYPE>GET (jhandle, compname, index, value, ilen, ierror)
```

- **<TYPE>**
  - A placeholder for one of the following values:

A placeholder for one of the following values:
**Value** | **Data Type** | **Routine Name**  
---|---|---  
INT | INTEGER(4) | PXEFINTGET  
REAL | REAL(4) | PXFEREALGET  
LGCL | LOGICAL(4) | PXFELGCLGET  
STR | CHARACTER*(*) | PXFESTRGET  
CHAR | CHARACTER(1) | PXFECHARGET  
DBL | REAL(8) | PXFEDBLGET  
INT8 | INTEGER(8) | PXFEINT8GET  

**jhandle**
(Input) INTEGER(4). A handle of a structure.

**compname**
(Input) Character. The name of the component (or field) of the structure to retrieve data from.

**index**
(Input) INTEGER(4). The index of the array element to get data for.

**value**
(Output) A variable, whose data type depends on the value of `<TYPE>`. See the table above for the data types for each value; for example, if the value for `<TYPE>` is INT, the data type is INTEGER(4). Stores the value of the component (or field).

**ilen**
(Output) INTEGER(4). This argument can only be used when `<TYPE>` is STR (PXFESTRGET). Stores the length of the returned string.

**ierror**
(Output) INTEGER(4). The error status.
If successful, `ierror` is set to zero; otherwise, an error code.

The PXFE<TYPE>GET subroutines are similar to the PXF<TYPE>GET subroutines, but they should be used when the component (or field) of the structure is an array.

When the PXFE<TYPE>GET subroutines are used, the array element with index `index` is accessed (read from the component or field).

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "PXFE<TYPE>SET", "PXF<TYPE>GET"
**PXFE<TYPE>SET**

**POSIX Subroutine:** Sets the value of an array element component (or field) of a structure.

**Module:** USE IFPOSIX

**Syntax**

CALL PXFE<TYPE>SET (jhandle, compname, index, value, ierror)

CALL PXFE<TYPE>SET (jhandle, compname, index, value, ilen, ierror) ! when <TYPE> is STR

A placeholder for one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Routine Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INTEGER(4)</td>
<td>PXEFINTSET</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL(4)</td>
<td>PXFEREALSET</td>
</tr>
<tr>
<td>LGCL</td>
<td>LOGICAL(4)</td>
<td>PXFELGCLSET</td>
</tr>
<tr>
<td>STR</td>
<td>CHARACTER(*)</td>
<td>PXFESTRSET</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHARACTER(1)</td>
<td>PXFECHARSET</td>
</tr>
<tr>
<td>DBL</td>
<td>REAL(8)</td>
<td>PXFEDBLSET</td>
</tr>
<tr>
<td>INT8</td>
<td>INTEGER(8)</td>
<td>PXFEINT8SET</td>
</tr>
</tbody>
</table>

**jhandle**

(Input) INTEGER(4). A handle of a structure.

**compname**

(Input) Character. The name of the component (or field) of the structure to write data to.

**index**

(Input) INTEGER(4). The index of the array element to write data to.

**value**

(Input) A variable, whose data type depends on the value of <TYPE>. See the table above for the data types for each value; for example, if the value for <TYPE> is INT, the data type is INTEGER(4). The value for the component (or field).

**ilen**

(Input) INTEGER(4). This argument can only be used when <TYPE> is STR (PXFESTRSET). The length of the string value.

**ierror**

(Output) INTEGER(4). The error status.
If successful, \textit{ierror} is set to zero; otherwise, an error code.

The PXFE\textless TYPE\textgreater SET subroutines are similar to the PXF\textless TYPE\textgreater SET subroutines, but they should be used when the component (or field) of the structure is an array.

When the PXFE\textless TYPE\textgreater SET subroutines are used, the array element with index \textit{index} is accessed (written to the component or field).

\textbf{Compatibility}

\begin{verbatim}
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
\end{verbatim}

\textbf{See Also:} “PXFE\textless TYPE\textgreater GET”, “PXF\textless TYPE\textgreater SET”

\section*{PXFEEXECV}

\textbf{POSIX Subroutine:} Executes a new process by passing command-line arguments.

\textbf{Module:} \texttt{USE IFPOSIX}

\textbf{Syntax}

\begin{verbatim}
CALL PXFEEXECV (path, lenpath, argv, lenargv, iargc, ierror)
\end{verbatim}

\textit{path}

(Input) Character. The path to the new executable process.

\textit{lenpath}

(Input) INTEGER(4). The length of \textit{path} string.

\textit{argv}

(Input) An array of character strings. Contains the command-line arguments to be passed to the new process.

\textit{lenargv}

(Input) INTEGER(4). An array that contains the lengths for each corresponding character string in \textit{argv}.

\textit{iargc}

(Input) INTEGER(4). The number of command-line arguments.

\textit{ierror}

(Output) INTEGER(4). The error status.

If successful, \textit{ierror} is set to zero; otherwise, an error code.

The PXFEEXECV subroutine executes a new executable process (file) by passing command-line arguments specified in the \textit{argv} array. If execution is successful, no return is made to the calling process.
Compatibility
CONSOLE   STANDARD GRAPHICS   QUICKWIN GRAPHICS   WINDOWS   DLL   LIB

See Also: “PXFEEXECVE”, “PXFEEXECVP”

**PXFEEXECVE**

**POSIX Subroutine:** Executes a new process by passing command-line arguments.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFEEXECVE (path, lenpath, argv, lenargv, iargc, env, lenenv, ienvc, ierror)
```

`path`
(Input) Character. The path to the new executable process.

`lenpath`
(Input) INTEGER(4). The length of `path` string.

`argv`
(Input) An array of character strings. Contains the command-line arguments to be passed to the new process.

`lenargv`
(Input) INTEGER(4). An array that contains the lengths for each corresponding character string in `argv`.

`iargc`
(Input) INTEGER(4). The number of command-line arguments.

`env`
(Input) An array of character strings. Contains the environment settings for the new process.

`lenenv`
(Input) INTEGER(4). An array that contains the lengths for each corresponding character string in `env`.

`ienvc`
(Input) INTEGER(4). The number of environment settings in `env`.

`ierror`
(Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.
The PXFEXECVE subroutine executes a new executable process (file) by passing command-line arguments specified in the \texttt{argv} array and environment settings specified in the \texttt{env} array.

**Compatibility**

\texttt{CONSOLE}  \texttt{STANDARD GRAPHICS}  \texttt{QUICKWIN GRAPHICS}  \texttt{WINDOWS}  \texttt{DLL}  \texttt{LIB}

**See Also:** “\texttt{PXFEXECV}”, “\texttt{PXFEXECVP}”

**PXFEXECVP**

**POSIX Subroutine:** Executes a new process by passing command-line arguments.

**Module:** USE IFPOSIX

**Syntax**

\begin{verbatim}
CALL PXFEXECVP (file, lenfile, argv, lenargv, iargc, ierror)
\end{verbatim}

- **file**
  (Input) Character. The filename of the new executable process.

- **lenfile**
  (Input) INTEGER(4). The length of \texttt{file} string.

- **argv**
  (Input) An array of character strings. Contains the command-line arguments to be passed to the new process.

- **lenargv**
  (Input) INTEGER(4). An array that contains the lengths for each corresponding character string in \texttt{argv}.

- **iargc**
  (Input) INTEGER(4). The number of command-line arguments.

- **ierror**
  (Output) INTEGER(4). The error status.

If successful, \texttt{ierror} is set to zero; otherwise, an error code.

The PXFEXECVP subroutine executes a new executable process (file) by passing command-line arguments specified in the \texttt{argv} array. It uses the \texttt{PATH} environment variable to find the file to execute.

**Compatibility**

\texttt{CONSOLE}  \texttt{STANDARD GRAPHICS}  \texttt{QUICKWIN GRAPHICS}  \texttt{WINDOWS}  \texttt{DLL}  \texttt{LIB}

**See Also:** “\texttt{PXFEXECV}”, “\texttt{PXFEXECVE}”
PXFEXIT, PXFFASTEXIT

POSIX Subroutine: Exits from a process.

Module: USE IFPOSIX

Syntax

CALL PXFEXIT (istatus)
CALL PXFFASTEXIT (istatus)

istatus

(Input) INTEGER(4). The exit value.

The PXFEXIT subroutine terminates the calling process. It calls, in last-in-first-out (LIFO) order, the functions registered by C runtime functions `atexit` and `onexit`, and flushes all file buffers before terminating the process. The `istatus` value is typically set to zero to indicate a normal exit and some other value to indicate an error.

The PXFFASTEXIT subroutine terminates the calling process without processing `atexit` or `onexit`, and without flushing stream buffers.

Compatibility

CONSOLE  STANDARD  GRAPHICS  QUICKWIN  GRAPHICS  WINDOWS  DLL  LIB

Example

program t1
use ifposix

integer(4) ipid, istat, ierror, ipid_ret, istat_ret
print *," the child process will be born"
call PXFFORK(IPID, IERROR)
call PXFGETPID(IPID_RET,IERROR)
if(IPID.EQ.0) then
  print *," I am a child process"
  print *," My child's pid is", IPID_RET
call PXFGETPPID(IPID_RET,IERROR)
  print *," The pid of my parent is",IPID_RET
  print *," Now I have exited with code 0xABCD"
call PXFEXIT(Z'ABCD')
else
  print *," I am a parent process"
  print *," My parent pid is ", IPID_RET
  print *," I am creating the process with pid", IPID
  print *," Now I am waiting for the end of the child process"
call PXFWAIT(ISTAT, IPID_RET, IERROR)
print *," The child with pid ", IPID_RET," has exited"
if( PXFWIFEXITED(ISTAT) ) then
  print *, " The child exited normally"
  istat_ret = IPXFWEXITSTATUS(ISTAT)
  print 10," The low byte of the child exit code is", istat_ret
end if
end if
10 FORMAT (A,Z)
end program

PXFFDOPEN

POSIX Subroutine: Opens an external unit.
Module: USE IFPOSIX
Syntax
  CALL PXFFDOPEN (ifildes, iunit, access, ierror)
ifildes
(Input) INTEGER(4). The file descriptor of the opened file.

iunit
(Input) INTEGER(4). The Fortran logical unit to connect to file descriptor ifildes.

access
(Input) Character. A character string that specifies the attributes for the Fortran unit. The string must consist of one or more of the following keyword/value pairs. Keyword/value pairs should be separated by a comma, and blanks are ignored.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Possible Values</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>'NEWLINE'</td>
<td>'YES' or 'NO'</td>
<td>I/O type</td>
<td>'YES'</td>
</tr>
<tr>
<td>'BLANK'</td>
<td>'NULL' or 'ZERO'</td>
<td>Interpretation of blanks</td>
<td>'NULL'</td>
</tr>
<tr>
<td>'STATUS'</td>
<td>'OLD', 'SCRATCH', or 'UNKNOWN'</td>
<td>File status at open</td>
<td>'UNKNOWN'</td>
</tr>
<tr>
<td>'FORM'</td>
<td>'FORMATTED' or 'UNFORMATTED'</td>
<td>Format type</td>
<td>'FORMATTED'</td>
</tr>
</tbody>
</table>

Keywords should be separated from their values by the equals (‘=’) character; for example:

call PXFDOPEN (IFILDES, IUNIT, 'BLANK=NULL, STATUS=UNKNOWN', IERROR)
ierror
(Output) INTEGER(4). The error status.

The PXFFDOPEN subroutine connects an external unit identified by iunit to a file descriptor ifildes. If the unit is already connected to a file, the file should be closed before using PXFFDOPEN.

NOTE. On Windows® systems, the default value of the POSIX/IO flag is 0, which causes PXFFDOPEN to return an error. To prevent this, call subroutine PXFPOSIXIO and set the value of the POSIX/IO flag to 1.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFFFLUSH

POSIX Subroutine: Flushes a file directly to disk.
Module: USE IFPOSIX
Syntax
CALL PXFFFLUSH (lunit, ierror)
lunit
(Input) INTEGER(4). A Fortran logical unit.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFFFLUSH subroutine writes any buffered output to the file connected to unit lunit.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFFGETC

POSIX Subroutine: Reads a character from a file.
Module: USE IFPOSIX
Syntax
   CALL PXFFGETC (lunit, char, ierror)

lunit
(Input) INTEGER(4). A Fortran logical unit.

char
(Input) Character. The character to be read.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFFGETC subroutine reads a character from a file connected to unit lunit.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFFPUTC”

PXFFILENO

POSIX Subroutine: Returns the file descriptor associated with a specified unit.

Module: USE IFPOSIX

Syntax
   CALL PXFFILENO (lunit, fd, ierror)

lunit
(Input) INTEGER(4). A Fortran logical unit.

fd
(Output) INTEGER(4). The returned file descriptor.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code. Possible error codes are:

• EINVAL: lunit is not an open unit.
• EBADF: lunit is not connected with a file descriptor.

The PXFFILENO subroutine returns in fd the file descriptor associated with lunit.
NOTE. On Windows* systems, the default value of the POSIX/IO flag is 0, which prevents OPEN from connecting a unit to a file descriptor and causes PXFFILENO to return an error. To prevent this, call subroutine PXFPOSIXIO and set the value of the POSIX/IO flag to 1. This setting allows a connection to a file descriptor.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFFORK

POSIX Subroutine: Creates a child process that differs from the parent process only in its PID. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
CALL PXFFORK (ipid, ierror)

ipid
(Output) INTEGER(4). The returned PID of the new child process.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFFORK subroutine creates a child process that differs from the parent process only in its PID. If successful, the PID of the child process is returned in the parent’s thread of execution, and a zero is returned in the child’s thread of execution. Otherwise, a –1 is returned in the parent’s context and no child process is created.

See Also: “IPXFEXITSTATUS"

Example

program t1
use ifposix
integer(4) ipid, istat, ierror, ipid_ret, istat_ret
print *," the child process will be born"
call PXFFORK(IPID, IERROR)
call PXFGETPID(IPID_RET, IERROR)
if(IPID.EQ.0) then
  print *," I am a child process"
  print *," My child's pid is", IPID_RET
  call PXFGETPPID(IPID_RET,IERROR)
  print *," The pid of my parent is",IPID_RET
  print *," Now I have exited with code 0xABCD"
  call PXFEXIT(Z'ABCD')
else
  print *," I am a parent process"
  print *," My parent pid is ", IPID_RET
  print *," I am creating the process with pid", IPID
  print *," Now I am waiting for the end of the child process"
  call PXFWAIT(ISTAT, IPID_RET, IERROR)
  print *," The child with pid ", IPID_RET," has exited"
  if( PXFWIFEXITED(ISTAT) ) then
    print *, " The child exited normally"
    istat_ret = PXFWEXITSTATUS(ISTAT)
    print 10," The low byte of the child exit code is", istat_ret
  end if
end if
10 FORMAT (A,Z)
end program

PXFFPATHCONF

POSIX Subroutine:  Gets the value for a configuration option of an opened file.
Module:  USE IFPOSIX
Syntax
   CALL PXFFPATHCONF (ifildes, name, ival, ierror)
ifildes
(Input) INTEGER(4). The file descriptor of the opened file.
name
(Input) INTEGER(4). The configurable option.
ival
(Output) INTEGER(4). The value of the configurable option.
**ierror**

(Output) INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

The PXFSPATHCONF subroutine gets a value for the configuration option named for the opened file with descriptor *ifildes*.

The configuration option, defined in *name*, can be obtained by using PXFCNST or IPXFCNST with one of the following constant names:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PC_LINK_MAX</td>
<td>Returns the maximum number of links to the file. If <em>ifildes</em> refers to a directory, then the value applies to the whole directory.</td>
</tr>
<tr>
<td>_PC_MAX_CANON1</td>
<td>Returns the maximum length of a formatted input line; the file descriptor <em>ifildes</em> must refer to a terminal.</td>
</tr>
<tr>
<td>_PC_MAX_INPUT1</td>
<td>Returns the maximum length of an input line; the file descriptor <em>ifildes</em> must refer to a terminal.</td>
</tr>
<tr>
<td>_PC_NAME_MAX</td>
<td>Returns the maximum length of a filename in <em>ifildes</em> that the process is allowed to create.</td>
</tr>
<tr>
<td>_PC_PATH_MAX</td>
<td>Returns the maximum length of a relative pathname when <em>ifildes</em> is the current working directory.</td>
</tr>
<tr>
<td>_PCPIPE_BUF</td>
<td>Returns the size of the pipe buffer; the file descriptor <em>ifildes</em> must refer to a pipe or FIFO.</td>
</tr>
<tr>
<td>_PC_CHOWN_RESTRICTED1</td>
<td>Returns nonzero if PXFCHOWN may not be used on this file. If <em>ifildes</em> refers to a directory, then this applies to all files in that directory.</td>
</tr>
<tr>
<td>_PC_NO_TRUNC1</td>
<td>Returns nonzero if accessing filenames longer than _POSIX_NAME_MAX will generate an error.</td>
</tr>
<tr>
<td>_PC_VDISABLE1</td>
<td>Returns nonzero if special character processing can be disabled; the file descriptor <em>ifildes</em> must refer to a terminal.</td>
</tr>
</tbody>
</table>

1. L*X only

On Linux* systems, the corresponding macros are defined in `<unistd.h>`. The values for *name* can be obtained by using PXFCNST or IPXFCNST when passing the string names of predefined macros in `<unistd.h>`. The following table shows the corresponding macro names for the above constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Corresponding Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PC_LINK_MAX</td>
<td>_POSIX_LINK_MAX</td>
</tr>
<tr>
<td>_PC_MAX_CANON</td>
<td>_POSIX_MAX_CANON</td>
</tr>
</tbody>
</table>
PXFFPUTC

**POSIX Subroutine:** Writes a character to a file.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFFPUTC (lunit, char, ierr)
```

- **lunit**
  - (Input) INTEGER(4). A Fortran logical unit.
- **char**
  - (Input) Character. The character to be written.
- **ierr**
  - (Output) INTEGER(4). The error status.

If successful, **ierr** is set to zero; otherwise, an error code. A possible error code is EEND if the end of the file has been reached.

The PXFFPUTC subroutine writes a character to the file connected to unit **lunit**.

**Compatibility**

| CONSOLE | STANDARD GRAPHICS | QUICKWIN GRAPHICS | WINDOWS | DLL | LIB |

**See Also:** “IPXFCONST”, “PXCONST”, “PXFPATHCONF”

PXFFSEEK

**POSIX Subroutine:** Modifies a file position.

**Module:** USE IFPOSIX
CALL PXFFSEEK (lunit, ioffset, iwhence, ierr)

lunit (Input) INTEGER(4). A Fortran logical unit.

ioffset (Input) INTEGER(4). The number of bytes away from iwhence to place the pointer.

iwhence (Input) INTEGER(4). The position within the file. The value must be one of the following constants (defined in stdio.h):

• SEEK_SET = 0
  Offset from the beginning of the file.
• SEEK_CUR = 1
  Offset from the current position of the file pointer.
• SEEK_END = 2
  Offset from the end of the file.

ierror (Output) INTEGER(4). The error status.

If successful, ierr is set to zero; otherwise, an error code. Possible error codes are:

• EINVAL: No file is connected to lunit, iwhence is not a proper value, or the resulting offset is invalid.
• ESPIPE: lunit is a pipe or FIFO.
• EEND: The end of the file has been reached.

The PXFFSEEK subroutine modifies the position of the file connected to unit lunit.

Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFFSTAT

POSIX Subroutine: Gets a file's status information.

Module: USE IFPOSIX

Syntax

CALL PXFFSTAT (ifildes, jstat, ierr)
ifildes
(Input) INTEGER(4). The file descriptor for an opened file.

jstat
(Input) INTEGER(4). A handle of structure stat.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFFSTAT subroutine puts the status information for the file associated with ifildes into the structure associated with handle jstat.

NOTE. To get a handle for an instance of the stat structure, use PXFSTRUCTCREATE with the string 'stat' for the structure name.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: "PXFSTRUCTCREATE"

PXFFTELL

POSIX Subroutine: Returns the relative position in bytes from the beginning of the file.
Module: USE IFPOSIX
Syntax
CALL PXFFTELL (lunit, ioffset, ierror)
lunit
(Input) INTEGER(4). A Fortran logical unit.
ioffset
(Output) INTEGER(4). The returned relative position in bytes from the beginning of the file.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
PXFGETARG

POSIX Subroutine: Gets the specified command-line argument.
Module: USE IFPOSIX
Syntax
   CALL PXFGETARG (argnum, str, istr, ierror)

argnum
(Input) INTEGER(4). The number of the command-line argument.
str
(Output) Character. The returned string value.
istr
(Output) INTEGER(4). The length of the returned string; it is zero if an error occurs.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFGETARG subroutine places the command-line argument with number argnum into character string str. If argnum is equal to zero, the value of the argument returned is the command name of the executable file.

Compatibility
   CONSOLE  STANDARD  GRAPHICS  QUICKWIN  GRAPHICS  WINDOWS  DLL  LIB
See Also: “IPXFARGC”

PXFGETATTY

POSIX Subroutine: Tests whether a file descriptor is connected to a terminal.
Module: USE IFPOSIX
Syntax
   CALL PXFGETATTY (ifildes, isatty, ierror)

ifildes
(Input) INTEGER(4). The file descriptor.
isatty
(Output) LOGICAL(4). The returned value.
Descriptions of the Library Routines

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
If file descriptor ifildes is open and connected to a terminal, isatty returns .TRUE.; otherwise, .FALSE..

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFGETC

POSIX Subroutine: Reads a character from standard input unit 5.
Module: USE IFPOSIX
Syntax
   CALL PXFGETC (nextcar, ierror)
nextcar
(Output) Character. The returned character that was read.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
See Also: “PXFPUTC”

PXFGETCWD

POSIX Subroutine: Returns the path of the current working directory.
Module: USE IFPOSIX
Syntax
   CALL PXFGETCWD (buf, ilen, ierror)
buf
(Output) Character. The returned pathname of the current working directory.
ilen
(Output) INTEGER(4). The length of the returned pathname.
$ierror$

(Output) INTEGER(4). The error status.

If successful, $ierror$ is set to zero; otherwise, an error code. A possible error code is EINVAL if the size of $buf$ is insufficient.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**PXFGETEGID**

**POSIX Subroutine**: Gets the effective group ID of the current process. This subroutine is only available on Linux* systems.

**Module**: USE IFPOSIX

**Syntax**

```
CALL PXFGETEGID (iegid, ierror)
```

$iegid$

(Output) INTEGER(4). The returned effective group ID.

$ierror$

(Output) INTEGER(4). The error status.

If successful, $ierror$ is set to zero; otherwise, an error code.

The effective ID corresponds to the set ID bit on the file being executed.

**PXFGETENV**

**POSIX Subroutine**: Gets the setting of an environment variable.

**Module**: USE IFPOSIX

**Syntax**

```
CALL PXFGETENV (name, lenname, value, lenvalue, ierror)
```

$name$

(Input) Character. The name of the environment variable.

$lenname$

(Input) INTEGER(4). The length of $name$.

$value$

(Output) Character. The returned value of the environment variable.
**lenvalue**
(Output) INTEGER(4). The returned length of value. If an error occurs, it returns zero.

**ierror**
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "PXFSETENV"

**PXFGTEUOID**

**POSIX Subroutine:** Gets the effective user ID of the current process. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFGTEUOID (ieuid, ierror)
```

**ieuid**
(Output) INTEGER(4). The returned effective user ID.

**ierror**
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The effective ID corresponds to the set ID bit on the file being executed.

**PXFGGETGID**

**POSIX Subroutine:** Gets the real group ID of the current process. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFGGETGID (igid, ierror)
```

**igid**
(Output) INTEGER(4). The returned real group ID.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The real ID corresponds to the ID of the calling process.
See Also: “PXFSETGID”
Example
See the example in “PXFGETGROUPS”.

PXFGETGRGID

POSIX Subroutine: Gets group information for the specified GID. This subroutine is only available on Linux* systems.
Module: USE IFPOSIX
Syntax
CALL PXFGETGRGID (jgid, jgroup, ierror)
jgid
(Input) INTEGER(4). The group ID to retrieve information about.
jgroup
(Input) INTEGER(4). A handle of structure group.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is not changed; otherwise, an error code.
The PXFGETGRGID subroutine stores the group information from /etc/group for the entry that matches the group GID jgid in the structure associated with handle jgroup.

NOTE. To get a handle for an instance of the group structure, use PXFSTRUCTCREATE with the string 'group' for the structure name.

See Also: “PXFSTRUCTCREATE”
Example
See the example in “PXFGETGROUPS”.

2-298
Descriptions of the Library Routines

**PXFGETGRNAM**

**POSIX Subroutine:** Gets group information for the named group. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFGETGRNAM (name, ilen, jgroup, ierror)
```

- `name` (Input) Character. The name of the group to retrieve information about.
- `ilen` (Input) INTEGER(4). The length of the `name` string.
- `jgroup` (Input) INTEGER(4). A handle of structure `group`.
- `ierror` (Output) INTEGER(4). The error status.

If successful, `ierror` is not changed; otherwise, an error code.

The PXFGETGRNAM subroutine stores the group information from `/etc/group` for the entry that matches the group name `name` in the structure associated with handle `jgroup`.

**NOTE.** To get a handle for an instance of the `group` structure, use `PXFSTRUCTCREATE` with the string 'group' for the structure name.

See Also: “PXFSTRUCTCREATE”

**PXFGETGROUPS**

**POSIX Subroutine:** Gets supplementary group IDs. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFGETGROUPS (igidsetsize, igrouplist, ngroups, ierror)
```

- `igidsetsize` (Input) INTEGER(4). The number of elements in the `igrouplist` array.

The PXFGETGROUPS subroutine retrieves supplementary group IDs that are not associated with the user's primary group. The subroutine can be used to get the supplementary group IDs for a given user on Linux* systems.

**NOTE:** The `igidsetsize` parameter specifies the number of supplementary group IDs to be returned. The `igrouplist` array contains the group IDs obtained from the `/etc/group` file.
igrouplist
(Output) INTEGER(4). The array that has the returned supplementary group IDs.

ngroups
(Output) INTEGER(4). The total number of supplementary group IDs for the process.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFGETGROUPS subroutine returns, up to sizeigidsetsize, the supplementary group IDs in array igrouplist. It is unspecified whether the effective group ID of the calling process is included in the returned list. If the size is zero, the list is not modified, but the total number of supplementary group IDs for the process is returned.

Example

program test5
  use ifposix
  implicit none
  integer(4) number_of_groups, ierror, isize, i,igid
  integer(4), allocatable, dimension(:):: igrouplist
  integer(JHANDLE_SIZE) jgroup

  ! Get total number of groups in system
  ! call PXFGETGROUPS with 0
  call PXFGETGROUPS(0, igrouplist, number_of_groups, ierror)
  if(ierror.NE.0) STOP 'Error: first call of PXFGETGROUPS fails'
  print *, "The number of groups in system ", number_of_groups

  ! Get Group IDs
  isize = number_of_groups
  ALLOCATE( igrouplist(isize))
  call PXFGETGROUPS(isize, igrouplist, number_of_groups, ierror)
  if(ierror.NE.0) then
    DEALLOCATE(igrouplist)
    STOP 'Error: first call of PXFGETGROUPS fails'
  end if

  print *, "Create an instance for structure 'group'"
  call PXFSTRUCTCREATE("group", jgroup, ierror)
  if(ierror.NE.0) then
DEALLOCATE(igrouplist)
STOP 'Error: PXFSTRUCTCREATE failed to create an instance of group'
end if

do i=1, number_of_groups
call PXFGETGRGID( igrouplist(i), jgroup, ierror)
if(ierror.NE.0) then
    DEALLOCATE(igrouplist)
call PXFSTRUCTFREE(jgroup, ierror)
    print *, 'Error: PXFGETGRGID failed for i=',i," gid=", igrouplist(i)
    STOP 'Abnormal termination'
end if
    call PRINT_GROUP_INFO(jgroup)
end do

call PXFGETGID(igid,ierror)
if(ierr .NE. 0) then
    DEALLOCATE(igrouplist)
call PXFSTRUCTFREE(jgroup, ierror)
    print *, 'Error: PXFGETGID failed'
    STOP 'Abnormal termination'
end if

call PXFGETGRGID(igid, jgroup, ierror)
if(ierr .NE. 0) then
    DEALLOCATE(igrouplist)
call PXFSTRUCTFREE(jgroup, ierror)
    print *, 'Error: PXFGETGRGID failed for gid=',igid
    STOP 'Abnormal termination'
end if

call PRINT_GROUP_INFO(jgroup)
DEALLOCATE(igrouplist)
call PXFSTRUCTFREE(jgroup, ierror)
print *, " Program will normal terminated"
call PXFEXIT(0)
end
PXFGETLOGIN

**POSIX Subroutine**: Gets the name of the user.

**Module**: USE IFPOSIX

**Syntax**

```fortran
CALL PXFGETLOGIN (s, ilen, ierror)
```

- `s` (Output) Character. The returned user name.
- `ilen` (Output) INTEGER(4). The length of the string stored in `s`.
- `ierror` (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

**Compatibility**

| CONSOLE | STANDARD GRAPHICS | QUICKWIN GRAPHICS | WINDOWS | DLL | LIB |

PXFGETPGRP

**POSIX Subroutine**: Gets the process group ID of the calling process. This subroutine is only available on Linux* systems.

**Module**: USE IFPOSIX

**Syntax**

```fortran
CALL PXFGETPGRP (ipgrp, ierror)
```

- `ipgrp` (Output) INTEGER(4). The returned process group ID.
- `ierror` (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

Each process group is a member of a session and each process is a member of the session in which its process group is a member.

PXFGETPID

**POSIX Subroutine**: Gets the process ID of the calling process.
Module: USE IFPOSIX

Syntax
CALL PXFGETPID (ipid, ierror)

ipid
(Output) INTEGER(4). The returned process ID.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: the example in “PXFWAIT”

PXFGETPPID

POSIX Subroutine: Gets the process ID of the parent of the calling process.
Module: USE IFPOSIX

Syntax
CALL PXFGETPPID (ippid, ierror)

ippid
(Output) INTEGER(4). The returned process ID.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: the example in “PXFWAIT”

PXFGETPWNAM

POSIX Subroutine: Gets password information for a specified name. This subroutine is only available on Linux® systems.
Module: USE IFPOSIX
Syntax

CALL PXFGETPWNAM (name, ilen, jpasswd, ierror)

name
(Input) Character. The login name of the user to retrieve information about. For example, a login name might be "jsmith", while the actual name is "John Smith".

ilen
(Input) INTEGER(4). The length of the name string.

jpasswd
(Input) INTEGER(4). A handle of structure compnam.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

The PXFGETPWNAM subroutine stores the user information from /etc/passwd for the entry that matches the user name name in the structure associated with handle jpasswd.

NOTE. To get a handle for an instance of the compnam structure, use PXFSTRUCTCREATE with the string 'compnam' for the structure name.

See Also: “PXFSTRUCTCREATE”

PXFGETPWUID

POSIX Subroutine: Gets password information for a specified UID. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

CALL PXFGETPWUID (iuid, jpasswd, ierror)

iuid
(Input) INTEGER(4). The user ID to retrieve information about.

jpasswd
(Input) INTEGER(4). A handle of structure compnam.
**ierror**

(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFGETPWUID subroutine stores the user information from `/etc/passwd` for the entry that matches the user ID iuid in the structure associated with handle jpasswd.

---

**NOTE.** To get a handle for an instance of the `compnam` structure, use PXFSTRUCTCREATE with the string 'compnam' for the structure name.

---

See Also: "PXFSTRUCTCREATE"

---

### PXFGETSUBHANDLE

**POSIX Subroutine:** Returns a handle for a subroutine.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFGETSUBHANDLE (sub, jhandle1, ierror)
```

*sub*  
(Input) The Fortran subroutine to get a handle for.

*jhandle1*  
(Output) INTEGER(4). The returned handle for the subroutine.

*ierror*  
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

---

**NOTE.** The argument "sub" cannot be a function, an intrinsic, or an entry point, and must be defined with exactly one integer argument.

---

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

---

2-305
PXGETUID

**POSIX Subroutine:** Gets the real user ID of the current process. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXGETUID (iuid, ierror)
```

- **iuid**
  (Output) INTEGER(4). The returned real user ID.
- **ierror**
  (Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The real ID corresponds to the ID of the calling process.

**See Also:** “[PXFSETUID](#)”

PXISBLK

**POSIX Function:** Tests for a block special file.

**Module:** USE IFPOSIX

**Syntax**

```fortran
result = PXISBLK (m)
```

- **m**
  (Input) INTEGER(4). The value of the st_mode component (field) in the structure stat.

**Results:**

The result type is logical. If the file is a block special file, the result value is .TRUE.; otherwise, .FALSE..

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

**See Also:** “[PXFISCHR](#)”

PXISCHR

**POSIX Function:** Tests for a character file.

**Module:** USE IFPOSIX
Syntax
result = PXFISCHR (m)

m
(Input) INTEGER(4). The value of the st_mode component (field) in the structure stat.

Results:
The result type is logical. If the file is a character file, the result value is .TRUE.; otherwise, .FALSE..

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFISBLK”

PXFISCONST

POSIX Function: Tests whether a string is a valid constant name.
Module: USE IFPOSIX
Syntax
result = PXFISCONST (s)

s
(Input) Character. The name of the constant to test.

Results:
The result type is logical. The PXFISCONST function confirms whether the argument is a valid constant name that can be passed to functions PXFCONST and IPXFCONST. It returns .TRUE. only if IPXFCONST will return a valid value for name s.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “IPXFCONST”, “PXFCONST”

PXFISDIR

POSIX Function: Tests whether a file is a directory.
Module: USE IFPOSIX
Syntax
result = PXFISDIR (m)
2 Intel Fortran Libraries Reference

\( m \)

(Input) INTEGER(4). The value of the \textit{st\_mode} component (field) in the structure \textit{stat}.

Results:
The result type is logical. If the file is a directory, the result value is .TRUE.; otherwise, .FALSE..

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

\textbf{PXFISFIFO}

\textbf{POSIX Function}: Tests whether a file is a special FIFO file.
Module: USE IFPOSIX
Syntax
\begin{verbatim}
result = PXFISFIFO (m)
\end{verbatim}
\( m \)

(Input) INTEGER(4). The value of the \textit{st\_mode} component (field) in the structure \textit{stat}.

Results:
The result type is logical.
The \textit{PXFISFIFO} function tests whether the file is a special FIFO file created by \textit{PXFMKFIFO}. If the file is a special FIFO file, the result value is .TRUE.; otherwise, .FALSE..

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “\textit{PXFISREG}”, “\textit{PXFMKFIFO}”

\textbf{PXFISREG}

\textbf{POSIX Function}: Tests whether a file is a regular file.
Module: USE IFPOSIX
Syntax
\begin{verbatim}
result = PXFISREG (m)
\end{verbatim}
\( m \)

(Input) INTEGER(4). The value of the \textit{st\_mode} component (field) in the structure \textit{stat}.
Results:
The result type is logical. If the file is a regular file, the result value is .TRUE.; otherwise, .FALSE..

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
See Also: “PXFISFIFO”, “PXFMKFIFO”

PXFKILL

POSIX Subroutine: Sends a signal to a specified process.
Module: USE IFPOSIX
Syntax
   CALL PXFKILL (ipid, isig, ierror)

ipid
(Input) INTEGER(4). The process to kill. It is determined by one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>Kills the specific process.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Kills all processes in the group.</td>
</tr>
<tr>
<td>== 0</td>
<td>Kills all processes in the group except special processes.</td>
</tr>
<tr>
<td>== pid_t-1</td>
<td>Kills all processes.</td>
</tr>
</tbody>
</table>

isig
(Input) INTEGER(4). The value of the signal to be sent.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFKILL subroutine sends a signal with value isig to a specified process. On Windows* systems, only the ipid for the current process can be used.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFLINK

POSIX Subroutine: Creates a link to a file or directory.
Module:  USE IFPOSIX

Syntax

CALL PXFLINK (existing, lenexist, new, lennew, ierror)

existing
(Input) Character. The path to the file or directory you want to link to.

lenexist
(Input) INTEGER(4). The length of the existing string.

new
(Input) Character. The name of the new link file.

lennew
(Input) INTEGER(4). The length of the new string.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFLINK subroutine creates a new link (also known as a hard link) to an existing file. This new name can be used exactly as the old one for any operation. Both names refer to the same file (so they have the same permissions and ownership) and it is impossible to tell which name was the "original".

NOTE. On Windows* systems, this subroutine is only valid for NTFS file systems; for FAT systems, it returns an error.

Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFLOCALTIME

POSIX Subroutine: Converts a given elapsed time in seconds to local time.

Module:  USE IFPOSIX

Syntax

CALL PXFLOCALTIME (isecnds, iatime, ierror)
**iseconds**

(Input) INTEGER(4). The elapsed time in seconds since 00:00:00 Greenwich Mean Time, January 1, 1970.

**iatime**

(Output) INTEGER(4). One-dimensional array with 9 elements used to contain numeric time data. The elements of *iatime* are returned as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>iatime(1)</td>
<td>Seconds (0-59)</td>
</tr>
<tr>
<td>iatime(2)</td>
<td>Minutes (0-59)</td>
</tr>
<tr>
<td>iatime(3)</td>
<td>Hours (0-23)</td>
</tr>
<tr>
<td>iatime(4)</td>
<td>Day of month (1-31)</td>
</tr>
<tr>
<td>iatime(5)</td>
<td>Month (1-12)</td>
</tr>
<tr>
<td>iatime(6)</td>
<td>Gregorian year (for example, 1990)</td>
</tr>
<tr>
<td>iatime(7)</td>
<td>Day of week (0-6, where 0 is Sunday)</td>
</tr>
<tr>
<td>iatime(8)</td>
<td>Day of year (1-366)</td>
</tr>
<tr>
<td>iatime(9)</td>
<td>Daylight savings flag (1 if daylight savings time is in effect; otherwise, 0)</td>
</tr>
</tbody>
</table>

**ierror**

(Output) INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

The PXFLOCALTIME subroutine converts the time (in seconds since epoch) in the *iseconds* argument to the local date and time as described by the array *iatime* above.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

**PXFLSEEK**

**POSIX Subroutine:** Positions a file a specified distance in bytes.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFLSEEK (ifildes, ioffset, iwhence, iposition, ierror)
```

**ifildes**

(Input) INTEGER(4). A file descriptor.
ioffset
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. The number of bytes to move.

iwhence
(Input) INTEGER(4). The starting position. The value must be one of the following:
- SEEK_SET = 0
  Sets the offset to $ioffset$ bytes.
- SEEK_CUR = 1
  Sets the offset to its current location plus $ioffset$ bytes.
- SEEK_END = 2
  Sets the offset to the size of the file plus $ioffset$ bytes.

iposition
(Output) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. The ending position; the resulting offset location as measured in bytes from the beginning of the file.

ierror
(Output) INTEGER(4). The error status.
If successful, $ierror$ is set to zero; otherwise, an error code.
The PXFLSEEK subroutine repositions the offset of file descriptor $ifildes$ to the argument $ioffset$ according to the value of argument $iwhence$.
PXFLSEEK allows the file offset to be set beyond the end of the existing end-of-file. If data is later written at this point, subsequent reads of the data in the gap return bytes of zeros (until data is actually written into the gap).

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFMKDIR

POSIX Subroutine: Creates a new directory.
Module: USE IFPOSIX
Syntax
   CALL PXFMKDIR (path, ilen, imode, ierror)
path
(Input) Character. The path for the new directory.
ilen
(Input) INTEGER(4). The length of path string.

imode (L*X only)
(Input) INTEGER(4). The mode mask. Octal file-access mode.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFRMDIR”, “PXFCHDIR”

PXFMKFIFO

POSIX Subroutine: Creates a new FIFO. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
CALL PXFMKFIFO (path, ilen, imode, ierror)

path
(Input) Character. The path for the new FIFO.

ilen
(Input) INTEGER(4). The length of path string.

imode
(Input) INTEGER(4). The mode mask; specifies the FIFO's permissions. Octal file-access mode.

ierror
(Output) INTEGER(4). The error status.
If successful ierror is set to zero; otherwise, an error code.

The PXFMKFIFO subroutine creates a FIFO special file with name path. A FIFO special file is similar to a pipe, except that it is created in a different way. Once a FIFO special file is created, any process can open it for reading or writing in the same way as an ordinary file. However, the FIFO file has to be open at both ends simultaneously before you can proceed to do any input or output operations on it. Opening a FIFO for reading normally blocks it until some other process opens the same FIFO for writing, and vice versa.

See Also: “PXFISFIFO”
**PXFOPEN**

**POSIX Subroutine:** Opens or creates a file.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFOPEN (path, ilen, iopenflag, imode, ifildes, ierror)
```

- **path**
  (Input) Character. The path of the file to be opened or created.

- **ilen**
  (Input) INTEGER(4). The length of `path` string.

- **iopenflag**
  (Input) INTEGER(4). The flags for the file. (For possible constant names that can be passed to PXFCONST or IPXFCONST, see below.)

- **imode**
  (Input) INTEGER(4). The permissions for a new file. This argument should always be specified when `iopenflag` = O_CREAT; otherwise, it is ignored. (For possible permissions, see below.)

- **ifildes**
  (Output) INTEGER(4). The returned file descriptor for the opened or created file.

- **ierror**
  (Output) INTEGER(4). The error status.

  If successful, `ierror` is set to zero; otherwise, an error code.

For `iopenflag`, you should specify one of the following constant values:

- O_RDONLY (read only)
- O_WRONLY (write only)
- O_RDWR (read and write)

In addition, you can also specify one of the following constant values by using a bitwise inclusive OR (IOR):

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_CREAT</td>
<td>Creates and opens a file if the file does not exist.</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>When used with O_CREAT, it causes the open to fail if the file already exists. In this case, a symbolic link exists, regardless of where it points to.</td>
</tr>
<tr>
<td>O_NOCTTY</td>
<td>If <code>path</code> refers to a terminal device, it prevents it from becoming the process's controlling terminal even if the process does not have one.</td>
</tr>
</tbody>
</table>
Argument *imode* specifies the permissions to use if a new file is created. The permissions only apply to future accesses of the newly created file. The value for *imode* can be any of the following constant values (which can be obtained by using PXFCONST or IPXFCONST):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_TRUNC</td>
<td>If the file already exists, it is a regular file, and <em>imode</em> allows writing (its value is O_RDWR or O_WRONLY), it causes the file to be truncated to length 0.</td>
</tr>
<tr>
<td>O_APPEND</td>
<td>Opens the file in append mode. Before each write, the file pointer is positioned at the end of the file, as if with PXFSEEK.</td>
</tr>
<tr>
<td>O_NONBLOCK (or O_NDELAY)</td>
<td>When possible, opens the file in non-blocking mode. Neither the open nor any subsequent operations on the file descriptor that is returned will cause the calling process to wait. This mode need not have any effect on files other than FIFOs.</td>
</tr>
<tr>
<td>O_SYNC</td>
<td>Opens the file for synchronous I/O. Any writes on the resulting file descriptor will block the calling process until the data has been physically written to the underlying hardware.</td>
</tr>
<tr>
<td>O_NOFOLLOW</td>
<td>If path is a symbolic link, it causes the open to fail.</td>
</tr>
<tr>
<td>O_DIRECTORY</td>
<td>If path is not a directory, it causes the open to fail.</td>
</tr>
<tr>
<td>O_LARGEFILE</td>
<td>On 32-bit systems that support the Large Files System, it allows files whose sizes cannot be represented in 31 bits to be opened.</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>Opens the file in binary (untranslated) mode.</td>
</tr>
<tr>
<td>O_SHORT_LIVED</td>
<td>Creates the file as temporary. If possible, it does not flush to the disk.</td>
</tr>
<tr>
<td>O_TEMPORARY</td>
<td>Creates the file as temporary. The file is deleted when last file handle is closed.</td>
</tr>
<tr>
<td>O_RANDOM</td>
<td>Specifies primarily random access from the disk.</td>
</tr>
<tr>
<td>O_SEQUENTIAL</td>
<td>Specifies primarily sequential access from the disk.</td>
</tr>
<tr>
<td>O_TEXT</td>
<td>Opens the file in text (translated) mode.</td>
</tr>
</tbody>
</table>

1. L*X only
2. W*32, W*64
3. For more information, see "Text and Binary Modes" in the Visual C++ programmer's guide.
## Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: "PXFCLOSE", "IPXFCONST", "PXFCONST"

### Example

```fortran
call PXFOPEN( "OPEN.OUT", &
            8, &
            IOR( IPXFCONST(O_WRONLY), IPXFCONST(O_CREAT) ), &
            IOR( IPXFCONST(S_IREAD), IPXFCONST(S_IWRITE) ) )
```

## PXFOPENDIR

**POSIX Subroutine:** Opens a directory and associates a stream with it.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFOPENDIR (dirname, lendirname, opendirid, ierror)
```

- **dirname**
  - (Input) Character. The directory name.

- **lendirname**
  - (Input) INTEGER(4). The length of `dirname` string.

- **opendirid**
  - (Output) INTEGER(4). The returned ID for the directory.

- **ierror**
  - (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

### Value Description

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IWGRP</td>
<td>00020 group has write permission.</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>00010 group has execute permission.</td>
</tr>
<tr>
<td>S_IXWXO</td>
<td>00007 others have read, write and execute permission.</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>00004 others have read permission.</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>00002 others have write permission.</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>00001 others have execute permission.</td>
</tr>
</tbody>
</table>

1. L’X only

Value Description

1. L’X only

S_IWGRP 1 00020 group has write permission.

S_IXGRP 1 00010 group has execute permission.

S_IXWXO 1 00007 others have read, write and execute permission.

S_IROTH 1 00004 others have read permission.

S_IWOTH 1 00002 others have write permission.

S_IXOTH 1 00001 others have execute permission.
The PXFOPENDIR subroutine opens a directory pointed to by the `dirname` argument and returns the ID of the directory into `opendirid`. After the call, this ID can be used by functions PXFREADDR, PXFREWINDDIR, PXFCLOSEDIR.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL LIB

See Also: “PXFCLOSEDIR”, “PXFREADDR”, “PXFREWINDDIR”

### PXFPATHCONF

**POSIX Subroutine:** Gets the value for a configuration option of an opened file.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFPATHCONF (path, ilen, name, ival, ierror)
```

- **path**
  - (Input) Character. The path to the opened file.
- **ilen**
  - (Input) INTEGER(4). The length of `path`.
- **name**
  - (Input) INTEGER(4). The configurable option.
- **ival**
  - (Input) INTEGER(4). The value of the configurable option.
- **ierror**
  - (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

The PXFPATHCONF subroutine gets a value for the configuration option named for the opened file with path `path`.

The configuration option, defined in `name`, can be obtained by using PXFCONST or IPXFCONST with one of the following constant names:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PC_LINK_MAX</td>
<td>Returns the maximum number of links to the file. If <code>path</code> refers to a directory, then the value applies to the whole directory.</td>
</tr>
<tr>
<td>_PC_MAX_CANON1</td>
<td>Returns the maximum length of a formatted input line; the <code>path</code> must refer to a terminal</td>
</tr>
</tbody>
</table>
On Linux* systems, the corresponding macros are defined in `<unistd.h>`. The values for `name` can be obtained by using `PXFCCONST` or `IPXFCCONST` when passing the string names of predefined macros in `<unistd.h>`. The following table shows the corresponding macro names for the above constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PC_MAX_INPUT¹</td>
<td>Returns the maximum length of an input line; the <code>path</code> must refer to a terminal.</td>
</tr>
<tr>
<td>_PC_NAME_MAX</td>
<td>Returns the maximum length of a filename in <code>path</code> that the process is allowed to create.</td>
</tr>
<tr>
<td>_PC_PATH_MAX</td>
<td>Returns the maximum length of a relative pathname when <code>path</code> is the current working directory.</td>
</tr>
<tr>
<td>_PC_PIPE_BUF</td>
<td>Returns the size of the pipe buffer; the <code>path</code> must refer to a FIFO.</td>
</tr>
<tr>
<td>_PC_CHOWN_RESTRICTED¹</td>
<td>Returns nonzero if PXFCCHOWN may not be used on this file. If <code>path</code> refers to a directory, then this applies to all files in that directory.</td>
</tr>
<tr>
<td>_PC_NO_TRUNC¹</td>
<td>Returns nonzero if accessing filenames longer than _POSIX_NAME_MAX will generate an error.</td>
</tr>
<tr>
<td>_PC_VDISABLE¹</td>
<td>Returns nonzero if special character processing can be disabled; the <code>path</code> must refer to a terminal.</td>
</tr>
</tbody>
</table>

¹. L*X only

See Also: “[PXFCCONST](#), [PXFCCONST](#), [PXFFPATHCONF](#)

**PXFPAUSE**

**POSIX Subroutine:** Suspends process execution.
Module: USE IFPOSIX

Syntax

CALL PXFPAUSE (ierror)

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFPAUSE subroutine causes the invoking process (or thread) to sleep until a signal is received that either terminates it or causes it to call a signal-catching function.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFPIPE

POSIX Subroutine: Creates a communications pipe between two processes.
Module: USE IFPOSIX

Syntax

CALL PXFPIPE (ireadfd, iwritefd, ierror)

ireadfd
(Output) INTEGER(4). The file descriptor for reading.
iwritefd
(Output) INTEGER(4). The file descriptor for writing.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code. The PXFPIPE subroutine returns a pair of file descriptors, pointing to a pipe inode, and places them into ireadfd for reading and into iwritefd for writing.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFPOSIXIO

POSIX Subroutine: Sets the current value of the POSIX I/O flag.
Module: USE IFPOSIX
Syntax

CALL PXFPOSIXIO (new, old, ierror)

new
(Input) INTEGER(4). The new value for the POSIX I/O flag.

old
(Output) INTEGER(4). The previous value of the POSIX I/O flag.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

This subroutine sets the current value of the Fortran POSIX I/O flag and returns the previous value of the flag. The initial state of the POSIX I/O flag is unspecified.

If a file is opened with a Fortran OPEN statement when the value of the POSIX I/O flag is 1, the unit is accessed as if the records are newline delimited, even if the file does not contain records that are delimited by a new line character.

If a file is opened with a Fortran OPEN statement when the value of the POSIX I/O flag is zero, a connection to a file descriptor is not assumed and the records in the file are not required to be accessed as if they are newline delimited.

Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFPUTC

POSIX Subroutine: Outputs a character to logical unit 6 (stdout).

Module: USE IFPOSIX

Syntax

CALL PXFPUTC (ch, ierror)

ch
(Input) Character. The character to be written.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code. A possible error code is EEND if the end of the file has been reached.
Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
See Also: “PXFGETC”

PXFREAD

POSIX Subroutine: Reads from a file.
Module: USE IFPOSIX
Syntax
CALL PXFREAD (ifildes, buf, nbyte, nread, ierror)

ifildes
(Input) INTEGER(4). The file descriptor of the file to be read from.
buf
(Output) Character. The buffer that stores the data read from the file.
nbyte
(Input) INTEGER(4). The number of bytes to read.
nread
(Output) INTEGER(4). The number of bytes that were read.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFREAD subroutine reads nbyte bytes from the file specified by ifildes into memory in buf.
The subroutine returns the total number of bytes read into nread. If no error occurs, the value of nread will equal the value of nbyte.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
See Also: “PXFWRITE”

PXFREADDR

POSIX Subroutine: Reads the current directory entry.
Module: USE IFPOSIX
**Intel Fortran Libraries Reference**

**Syntax**

```fortran
CALL PXFREADDR (idirid, jdirent, ierror)
```

- `idirid` (Input) INTEGER(4). The ID of a directory obtained from PXFOPENDIR.
- `jdirent` (Output) INTEGER(4). A handle of structure `dirent`.
- `ierror` (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

The PXFREADDR subroutine reads the entry of the directory associated with `idirid` into the structure associated with handle `jdirent`.

---

**NOTE.** To get a handle for an instance of the `dirent` structure, use PXFSTRUCTCREATE with the string 'dirent' for the structure name.

---

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFOPENDIR”, “PXFREWINDDIR”

### PXFRENAME

**POSIX Subroutine**: Changes the name of a file.

**Module**: USE IFPOSIX

**Syntax**

```fortran
CALL PXFRENAME (old, lenold, new, lennew, ierror)
```

- `old` (Input) Character. The name of the file to be renamed.
- `lenold` (Input) INTEGER(4). The length of `old` string.
- `new` (Input) Character. The new file name.

**NOTE.** To get a handle for an instance of the `dirent` structure, use PXFSTRUCTCREATE with the string 'dirent' for the structure name.
lennew
(Input) INTEGER(4). The length of new string.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFREWINDDIR

POSIX Subroutine: Resets the position of the stream to the beginning of the directory.
Module: USE IFPOSIX

Syntax
CALL PXFREWINDDIR (idirid, ierror)

idirid
(Input) INTEGER(4). The ID of a directory obtained from PXFOPENDIR.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

PXFRMDIR

POSIX Subroutine: Removes a directory.
Module: USE IFPOSIX

Syntax
CALL PXFRMDIR (path, ilen, ierror)

path
(Input) Character. The directory to be removed. It must be empty.

ilen
(Input) INTEGER(4). The length of path string.
**ierror**

(Output) INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "PXFMKDIR", "PXFCHDIR"

---

**PXFSSETENV**

**POSIX Subroutine:** Adds a new environment variable or sets the value of an environment variable.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSSETENV (name, lenname, new, lennew, ioverwrite, ierror)
```

**name**

(Input) Character. The name of the environment variable.

**lenname**

(Input) INTEGER(4). The length of *name*.

**new**

(Input) Character. The value of the environment variable.

**lennew**

(Input) INTEGER(4). The length of *new*.

**ioverwrite**

(Input) INTEGER(4). A flag indicating whether to change the value of the environment variable if it exists.

**ierror**

(Output) INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

If *name* does not exist, PXFSSETENV adds it with value *new*.

If *name* exists, PXFSSETENV sets its value to *new* if *ioverwrite* is a nonzero number. If *ioverwrite* is zero, the value of *name* is not changed.

If *lennew* is equal to zero, PXFSSETENV sets the value of the environment variable to a string equal to *new* after removing any leading or trailing blanks.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “PXFGETENV”

Example
program test2
use ifposix
character*10 name, new
integer lenname, lennew, ioverwrite, ierror
name = "FOR_NEW"
lenname = 7
new = "ON"
lennew = 2
ioverwrite = 1

CALL PXFSETENV (name, lenname, new, lennew, ioverwrite, ierror)
print *, "name= ", name
print *, "lenname= ", lenname
print *, "new= ", lenname
print *, "lennew= ", lenname
print *, "ierror= ", ierror
end

PXFSETGID

POSIX Subroutine: Sets the effective group ID of the current process. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
   CALL PXFSETGID (igid, ierror)

igid
(Input) INTEGER(4). The group ID.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
If the caller is the superuser, the real and saved group ID’s are also set. This feature allows a program other than root to drop all of its group privileges, do some un-privileged work, and then re-engage the original effective group ID in a secure manner.

**CAUTION.** *If the user is root then special care must be taken.* PXFSETGID checks the effective gid of the caller. If it is the superuser, all process-related group ID’s are set to gid. After this has occurred, it is impossible for the program to regain root privileges.

See Also: “PXFGETGID”

**PXFSETPGID**

**POSIX Subroutine:** Sets the process group ID. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFSETPGID (ipid, ipgid, ierror)
```

*ipid* *(Input)* INTEGER(4). The process group ID to change.

*ipgid* *(Input)* INTEGER(4). The new process group ID.

*ierror* *(Output)* INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

The PXFSETPGID subroutine sets the process group ID of the process specified by *ipid* to *ipgid*. If *ipid* is zero, the process ID of the current process is used. If *ipgid* is zero, the process ID of the process specified by *ipid* is used.

PXFSETPGID can be used to move a process from one process group to another, but both process groups must be part of the same session. In this case, *ipgid* specifies an existing process group to be joined and the session ID of that group must match the session ID of the joining process.
PXFSETSID

**POSIX Subroutine:** Creates a session and sets the process group ID. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```plaintext
CALL PXFSETSID (isid, ierror)
```

- `isid` (Output) INTEGER(4). The session ID.
- `ierror` (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

The PXFSETSID subroutine creates a new session if the calling process is not a process group leader.

The calling process is the leader of the new session and the process group leader for the new process group. The calling process has no controlling terminal.

The process group ID and session ID of the calling process are set to the PID of the calling process. The calling process will be the only process in this new process group and in this new session.

PXFSETUID

**POSIX Subroutine:** Sets the effective user ID of the current process. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```plaintext
CALL PXFSETUID (iuid, ierror)
```

- `iuid` (Output) INTEGER(4). The session ID.
- `ierror` (Output) INTEGER(4). The user status.

If successful, `ierror` is set to zero; otherwise, an error code.
If the effective user ID of the caller is root, the real and saved user ID’s are also set. This feature allows a program other than root to drop all of its user privileges, do some un-privileged work, and then re-engage the original effective user ID in a secure manner.

CAUTION. If the user is root then special care must be taken. PXFSETUID checks the effective uid of the caller. If it is the superuser, all process-related user ID's are set to uid. After this has occurred, it is impossible for the program to regain root privileges.

See Also: “PXFGETUID”

PXFSIGACTION

POSIX Subroutine: Changes the action associated with a specific signal. It can also be used to examine the action of a signal.

Module: USE IFPOSIX

Syntax

CALL PXFSIGACTION (isig, jsigact, josigact, ierror)

isig
(Input) INTEGER(4). The signal number whose action should be changed.

jsigact
(Input) INTEGER(4). A handle of structure sigaction. Specifies the new action for signal isig.

josigact
(Output) INTEGER(4). A handle of structure sigaction. Stores the previous action for signal isig.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The signal specified in isig can be any valid signal except SIGKILL and SIGSTOP.

If jsigact is nonzero, the new action for signal isig is installed from the structure associated with handle jsigact. If josigact is nonzero, the previous action of the specified signal is saved in the structure associated with handle josigact where it can be examined.

On Windows* systems, PXFSIGACTION ignores the fields sa_mask and sa_flags in structure sigaction.
NOTE. To get a handle for an instance of the sigaction structure, use PXFSTRUCTCREATE with the string 'sigaction' for the structure name.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFSTRUCTCREATE”

PXFSIGADDSET

POSIX Subroutine: Adds a signal to the signal set. This subroutine is only available on Linux* systems.
Module: USE IFPOSIX
Syntax
CALL PXFSIGADDSET (jsigset, isigno, ierror)

jsigset (Input) INTEGER(4). A handle of structure sigset. This is the set to add the signal to.
isigno (Input) INTEGER(4). The signal number to add to the set.
ierror (Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.
The PXFSIGADDSET subroutine adds signal number isigno to the set of signals associated with handle jsigset. This set of signals is used by PXFSIGACTION as field sa_mask in structure sigaction. It defines the set of signals that will be blocked during execution of the signal handler function (the field sa_handler in structure sigaction).

On Windows* systems, PXFSIGACTION ignores the field sa_mask in structure sigaction.

NOTE. To get a handle for an instance of the sigset structure, use PXFSTRUCTCREATE with the string 'sigset' for the structure name.

See Also: “PXFSTRUCTCREATE”, “PXFSIGDELSET”, “PXFSIGACTION”
**PXFSIGDELSASET**

**POSIX Subroutine:** Deletes a signal from the signal set. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSIGDELSASET (jsigset, isigno, ierror)
```

- `jsigset` *(Input)* INTEGER(4). A handle of structure `sigset`. This is the set to delete the signal from.
- `isigno` *(Input)* INTEGER(4). The signal number to delete from the set.
- `ierror` *(Output)* INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

The PXFSIGDELSASET subroutine removes signal number `isigno` from the set of signals associated with handle `jsigset`. This set of signals is used by PXFSIGACTION as field `sa_mask` in structure `sigaction`. It defines the set of signals that will be blocked during execution of the signal handler function (the field `sa_handler` in structure `sigaction`).

On Windows* systems, PXFSIGACTION ignores the field `sa_mask` in structure `sigaction`.

---

**NOTE.** To get a handle for an instance of the `sigset` structure, use PXFSTRUCTCREATE with the string `sigset` for the structure name.

**See Also:** “PXFSTRUCTCREATE”, “PXFSIGADDSET”, “PXFSIGACTION”

---

**PXFSIGEMPTYSET**

**POSIX Subroutine:** Empties a signal set. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSIGEMPTYSET (jsigset, ierror)
```

- `jsigset` *(Input)* INTEGER(4). A handle of structure `sigset`. This is the set to empty.
ierror
(Output) INTEGER(4). The error status.
If successful, *ierror* is set to zero; otherwise, nonzero.
The PXFSIGEMPTYSET subroutine initializes the signal set associated with handle *jsigset* to empty; all signals are excluded from the set. This set of signals is used by PXFSIGACTION as field *sa_mask* in structure *sigaction*. It defines the set of signals that will be blocked during execution of the signal handler function (the field *sa_handler* in structure *sigaction*).
On Windows* systems, PXFSIGACTION ignores the field *sa_mask* in structure *sigaction*.

NOTE. To get a handle for an instance of the *sigset* structure, use PXFSTRUCTCREATE with the string 'sigset' for the structure name.

See Also: “PXFSTRUCTCREATE”, “PXFSIGFILLSET”, “PXFSIGACTION”

PXFSIGFILLSET

**POSIX Subroutine:** Fills a signal set. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

CALL PXFSIGFILLSET (*jsigset*, *ierror*)

*jsigset*
(Input) INTEGER(4). A handle of structure *sigset*. This is the set to fill.

*ierror*
(Output) INTEGER(4). The error status.
If successful, *ierror* is set to zero; otherwise, an error code.

The PXFSIGFILLSET subroutine initializes the signal set associated with handle *jsigset* to full; all signals are included into the set. This set of signals is used by PXFSIGACTION as field *sa_mask* in structure *sigaction*. It defines the set of signals that will be blocked during execution of the signal handler function (the field *sa_handler* in structure *sigaction*).
On Windows* systems, PXFSIGACTION ignores the field *sa_mask* in structure *sigaction*.
POSIX Subroutine: Tests whether a signal is a member of a signal set. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

CALL PXFSIGISMEMBER (jsigset, isigno, ismember, ierror)

jsigset
(Input) INTEGER(4). A handle of structure sigset. This is the set the signal will be tested in.

isigno
(Input) INTEGER(4). The signal number to test for membership.

ismember
(Output) Logical. The returned result.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFSIGISMEMBER subroutine tests whether isigno is a member of the set associated with handle jsigset. If the signal is a member of the set, ismember is set to .TRUE.; otherwise, .FALSE.. This set of signals is used by PXFSIGACTION as field sa_mask in structure sigaction. It defines the set of signals that will be blocked during execution of the signal handler function (the field sa_handler in structure sigaction).

On Windows* systems, PXFSIGACTION ignores the field sa_mask in structure sigaction.

NOTE. To get a handle for an instance of the sigset structure, use PXFSTRUCTCREATE with the string 'sigset' for the structure name.
See Also: “PXFSTRUCTCREATE”, “PXFSIGACTION”

**PXFSIGPENDING**

**POSIX Subroutine:** Examines pending signals. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSIGPENDING (jsigset, ierror)
```

- `jsigset`  
  (Input) INTEGER(4). A handle of structure `sigaction`. The signals to examine.

- `ierror`  
  (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

The PXFSIGPENDING subroutine is used to examine pending signals (ones that have been raised while blocked). The signal mask of the pending signals is stored in the signal set associated with handle `jsigset`.

**PXFSIGPROCMASK**

**POSIX Subroutine:** Changes the list of currently blocked signals. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSIGPROCMASK (ihow, jsigset, josigset, ierror)
```

- `ihow`  
  (Input) INTEGER(4). Defines the action for `jsigset`.

- `jsigset`  
  (Input) INTEGER(4). A handle of structure `sigset`. The signals to examine.

- `josigset`  
  (Input) INTEGER(4). A handle of structure `sigset`. Stores the previous mask of blocked signals.

- `ierror`  
  (Output) INTEGER(4). The error status.
If successful, `ierror` is set to zero; otherwise, an error code.

The argument `ihow` indicates the way in which the set is to be changed, and consists of one of the following constant names:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG_BLOCK</td>
<td>The resulting set of blocked signals will be the union of the current signal set and the <code>jsigset</code> signal set.</td>
</tr>
<tr>
<td>SIG_UNBLOCK</td>
<td>The resulting set of blocked signals will be the current set of blocked signals with the signals in <code>jsigset</code> removed. It is legal to attempt to unblock a signal that is not blocked.</td>
</tr>
<tr>
<td>SIG_SETMASK</td>
<td>The resulting set of blocked signals will be the <code>jsigset</code> signal set.</td>
</tr>
</tbody>
</table>

1. These names can be used in PXFCONST or IPXFCONST.

If `josigset` is non-zero, the previous value of the signal mask is stored in the structure associated with handle `josigset`.

See Also: “IPXFCONST”, “PXFCONST”

**PXFSIGSUSPEND**

*POSIX Subroutine:* Suspends the process until a signal is received. This subroutine is only available on Linux® systems.

*Module:* USE IFPOSIX

*Syntax*

```fortran
CALL PXFSIGSUSPEND (jsigset, ierror)
```

- `ierror` (Output) INTEGER(4). The error status.

If successful, `ierror` is set to zero; otherwise, an error code.

PXFSIGSUSPEND temporarily replaces the signal mask for the process with that given by the structure associated with the `jsigset` handle; it then suspends the process until a signal is received.

**PXFSLEEP**

*POSIX Subroutine:* Forces the process to sleep.
Module: USE IFPOSIX

Syntax

    CALL PXFSLEEP (iseconds, isecleft, ierror)

iseconds
(Input) INTEGER(4). The number of seconds to sleep.

isecleft
(Output) INTEGER(4). The number of seconds left to sleep.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFSLEEP subroutine forces the current process to sleep until seconds isecconds have elapsed or a signal arrives that cannot be ignored.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFSSTAT

POSIX Subroutine: Gets a file's status information.

Module: USE IFPOSIX

Syntax

    CALL PXFSTAT (path, ilen, jstat, ierror)

path
(Input) Character. The path to the file.

ilen
(Input) INTEGER(4). The length of path string.

jstat
(Input) INTEGER(4). A handle of structure stat.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFSTAT subroutine puts the status information for the file specified by path into the structure associated with handle jstat.
NOTE. To get a handle for an instance of the stat structure, use PXFSTRUCTCREATE with the string 'stat' for the structure name.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: “PXFSTRUCTCREATE”

PXFSTRUCTCOPY
POSIX Subroutine: Copies the contents of one structure to another.
Module: USE IFPOSIX
Syntax
CALL PXFSTRUCTCOPY (structname, jhandle1, jhandle2, ierror)
structname
(Input) Character. The name of the structure.
jhandle1
(Input) INTEGER(4). A handle to the structure to be copied.
jhandle2
(Input) INTEGER(4). A handle to the structure that will receive the copy.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

Example
See the example in “PXFSTRUCTCREATE”.

PXFSTRUCTCREATE
POSIX Subroutine: Creates an instance of the specified structure.
Module: USE IFPOSIX
Syntax

CALL PXFSTRUCTCREATE (structname, jhandle, ierror)

structname
(Input) Character. The name of the structure.
As for any character string, the name must be specified in single or double quotes; for example, the structure sigaction would be specified as 'sigaction'. (For more information on available structures, see the table below.)

jhandle
(Output) INTEGER(4). The handle of the newly-created structure.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
If your application passes information to the system, you should call one of the PXF<TYPE>SET subroutines. If your application needs to get information from the structure, you should call one of the PXF<TYPE>GET subroutines.

The following table shows:
• The structures that are available in the Fortran POSIX library
• The fields within each structure
• The subroutines you must use to access the structure fields
The subroutine needed to access the field may depend on whether you are using an IA-32 processor or Intel Itanium processor. In these cases, the subroutine is labeled as i32 or i64, respectively, in the table:

<table>
<thead>
<tr>
<th>Structure Name</th>
<th>Field Name</th>
<th>Subroutines for Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigset¹</td>
<td>Fields are hidden.</td>
<td>PXFSGEMPTYSET¹, PXFSGFILLSET¹, PXFSGADDSET¹, or PXFSGDELSET¹</td>
</tr>
<tr>
<td>sigaction</td>
<td>sa_handler</td>
<td>PXFINTGET/PXFINTSET or PXFINT8GET/PXFINT8SET</td>
</tr>
<tr>
<td></td>
<td>sa_mask</td>
<td>PXFINTGET/PXFINTSET or PXFINT8GET/PXFINT8SET</td>
</tr>
<tr>
<td></td>
<td>sa_flags</td>
<td>PXFINTGET/PXFINTSET or PXFINT8GET/PXFINT8SET</td>
</tr>
<tr>
<td>utsname</td>
<td>sysname</td>
<td>For all fields: PXFSTRGET</td>
</tr>
<tr>
<td></td>
<td>nodename</td>
<td></td>
</tr>
<tr>
<td></td>
<td>release</td>
<td></td>
</tr>
<tr>
<td></td>
<td>version</td>
<td></td>
</tr>
<tr>
<td></td>
<td>machine</td>
<td></td>
</tr>
</tbody>
</table>
## Structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Field Name</th>
<th>Subroutines for Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>tms</td>
<td>tms_utime</td>
<td>For all fields: PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>tms_stime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tms_stime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tms_cstime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tms_cstime</td>
<td></td>
</tr>
<tr>
<td>dirent</td>
<td>d_name</td>
<td>PXFSRGET</td>
</tr>
<tr>
<td>stat</td>
<td>st_mode</td>
<td>For all fields: PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>st_ino</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_dev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_nlink</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_uid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_gid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_size</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_atime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_mtime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st_ctime</td>
<td></td>
</tr>
<tr>
<td>utimbuf</td>
<td>actime</td>
<td>For all fields: PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>modtime</td>
<td></td>
</tr>
<tr>
<td>flock</td>
<td>l_type</td>
<td>For all fields: PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>l_whence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>l_start</td>
<td></td>
</tr>
<tr>
<td></td>
<td>l_len</td>
<td></td>
</tr>
<tr>
<td></td>
<td>l_pid</td>
<td></td>
</tr>
<tr>
<td>termios</td>
<td>c_iflag</td>
<td>PXFINGET/PXFINSET or PXFIN8GET/PXFIN8SET</td>
</tr>
<tr>
<td></td>
<td>c_oflag</td>
<td>PXFINGET/PXFINSET or PXFIN8GET/PXFIN8SET</td>
</tr>
<tr>
<td></td>
<td>c_cflag</td>
<td>PXFINGET/PXFINSET or PXFIN8GET/PXFIN8SET</td>
</tr>
<tr>
<td></td>
<td>c_lflag</td>
<td>PXFINGET/PXFINSET or PXFIN8GET/PXFIN8SET</td>
</tr>
<tr>
<td></td>
<td>c_cc</td>
<td>PXFINGET/PXFINSET or PXFIN8GET/PXFIN8SET</td>
</tr>
<tr>
<td>group</td>
<td>gr_name</td>
<td>PXFSRGET</td>
</tr>
<tr>
<td></td>
<td>gr_gid</td>
<td>PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>gr_nmem</td>
<td>PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>gr_mem</td>
<td>PXFESTRGET</td>
</tr>
<tr>
<td>passwd</td>
<td>pw_name</td>
<td>PXFSRGET</td>
</tr>
<tr>
<td></td>
<td>pw_uid</td>
<td>PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>pw_gid</td>
<td>PXFINGET or PXFIN8GET</td>
</tr>
<tr>
<td></td>
<td>pw_dir</td>
<td>PXFSRGET</td>
</tr>
<tr>
<td></td>
<td>pw_shell</td>
<td>PXFSRGET</td>
</tr>
</tbody>
</table>

1. L**X only

### Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

See Also: "PXFSTRUCTFREE", the example in "PXFTIMES"

Example

program test4
use ifposix
implicit none
integer(jhandle_size) jhandle1,jhandle2
integer(4) ierror,ilen1

print *," Create a first instance for structure 'utsname' ",
call PXFSTRUCTCREATE("utsname",jhandle1,ierror)
if(ierror.NE.0) STOP 'Error: cannot create structure for jhandle1'

print *," Create a second instance for structure 'utsname' ",
call PXFSTRUCTCREATE("utsname",jhandle2,ierror)
if(ierror.NE.0) then
    call PXFSTRUCTFREE(jhandle1,ierror)
    STOP 'test failed - cannot create structure for jhandle2'
end if

print *,"Fill the structure associated with jhandle1 with arbitrary data"
call PXFSTRSET(jhandle1,"sysname","00000000000000",14,ierror)
if(ierror.NE.0) call Error('Error: can't set component sysname for jhandle1')
call PXFSTRSET(jhandle1,"Nodename","11111111111111",14,ierror)
if(ierror.NE.0) call Error('Error: can't set component nodename for jhandle1')
call PXFSTRSET(jhandle1,"RELEASE","22222222222222",14,ierror)
if(ierror.NE.0) call Error('Error: can't set component release for jhandle1')
call PXFSTRSET(jhandle1,"verSION","33333333333333",14,ierror)
if(ierror.NE.0) call Error('Error: can't set component version for jhandle1')
call PXFSTRSET(jhandle1,"machine","44444444444444",14,ierror)
if(ierror.NE.0) call Error('Error: can't set component machine for jhandle1')

print *,"Fill the structure associated with jhandle2 with arbitrary data"
call PXFSTRSET(jhandle2,"sysname","aaaaaaaaa",7,ierror)
if(ierror.NE.0) call Error('Error: can't set component sysname for jhandle2')
call PXFSTRSET(jhandle2,"Nodename","BBBBBBBBBB BBB",14,ierror)
if(ierr.0) call Error('Error: can't set component nodename for jhandle2')
call PXFSTRSET(jhandle2,"RELEASE","cCCC cc-cccnc",12,ierror)
if(ierr.0) call Error('Error: can't set component release for jhandle2')
call PXFSTRSET(jhandle2,"verSION","ddddd",1,ierror)
if(ierr.0) call Error('Error: can't set component version for jhandle2')
call PXFSTRSET(jhandle2,"machine","eeeeeee",6,ierror)
if(ierr.0) call Error('Error: can't set component machine for jhandle2')
print *,"Print contents of the structure associated with jhandle1"
call PRINT_UTSNAME(jhandle1)
print *,"Print contents of the structure associated with jhandle2"
call PRINT_UTSNAME(jhandle2)
print *,"Get operating system info into structure associated with jhandle1"
call PXFUNAME(jhandle1,ierror)
if(ierr.0) call Error('Error: call to PXFUNAME has failed')
print *,"Print contents of the structure associated with jhandle1"
print*,"  returned from PXFUNAME"
call PRINT_UTSNAME(jhandle1)
print *,"Copy the contents of the structure associated with jhandle1"
print *,"  into the structure associated with jhandle2"
call PXFSTRUCTCOPY("utsname",jhandle1,jhandle2,ierror)
if(ierr.0) call Error('Error: can't copy jhandle1 contents into jhandle2')
print *,"Print the contents of the structure associated with jhandle2."
print *,"  It should be the same after copying."
call PRINT_UTSNAME(jhandle2)
print *,"Free memory for instance of structure associated with jhandle1"
call PXFSTRUCTFREE(jhandle1,ierror)
if(ierr.0) STOP 'Error: can't free instance of structure for jhandle1'
print *,"Free memory for instance of structure associated with jhandle2"
call PXFSTRUCTFREE(jhandle2,ierror)
if(ierr.0) STOP 'Error: can't free instance of structure for jhandle2'
print *,"Program terminated normally"
call PXFEXIT(0)
end

**PXFSTRUCTFREE**

**POSIX Subroutine:** Deletes the instance of a structure.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSTRUCTFREE (jhandle, ierror)
```

- **jhandle**
  (Input) INTEGER(4). A handle of a structure.

- **ierror**
  (Output) INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

The PXFSTRUCTFREE subroutine deletes the instance of the structure associated with handle *jhandle*.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

**Example**

See the example in “PXFSTRUCTCREATE”, the example in “PXFTIMES”

**PXFSYSCONF**

**POSIX Subroutine:** Gets values for system limits or options.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFSYSCONF (name, ival, ierror)
```

- **name**
  (Input) INTEGER(4). The system option you want information about.

- **ival**
  (Output) INTEGER(4). The returned value.

- **ierror**
  (Output) INTEGER(4). The error status.
If successful, \textit{ierror} is set to zero; otherwise, an error code.

PXFSYSCONF lets you determine values for system limits or system options at runtime.

The value for \textit{name} can be any of the following constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_ARG_MAX$^1$</td>
<td>Indicates the maximum length of the arguments to the PXFEXEC family of routines.</td>
</tr>
<tr>
<td>_SC_CHILD_MAX$^1$</td>
<td>Indicates the number of simultaneous processes per user ID.</td>
</tr>
<tr>
<td>_SC_CLK_TCK</td>
<td>Indicates the number of clock ticks per second.</td>
</tr>
<tr>
<td>_SC_STREAM_MAX$^2$</td>
<td>Indicates the maximum number of streams that a process can have open at any time.</td>
</tr>
<tr>
<td>_SC_TZNAME_MAX</td>
<td>Indicates the maximum number of bytes in a timezone name.</td>
</tr>
<tr>
<td>_SC_OPEN_MAX</td>
<td>Indicates the maximum number of files that a process can have open at any time.</td>
</tr>
<tr>
<td>_SC_JOB_CONTROL$^1$</td>
<td>Indicates whether POSIX-style job control is supported.</td>
</tr>
<tr>
<td>_SC_SAVED_IDS$^1$</td>
<td>Indicates whether a process has a saved set-user-ID and a saved set-group-ID.</td>
</tr>
<tr>
<td>_SC_VERSION$^1$</td>
<td>Indicates the year and month the POSIX.1 standard was approved in the format YYYYMML; the value 199009L indicates the most recent revision, 1990.</td>
</tr>
<tr>
<td>_SC_BC_BASE_MAX$^1$</td>
<td>Indicates the maximum obase value accepted by the bc(1) utility.</td>
</tr>
<tr>
<td>_SC_BC_DIM_MAX$^1$</td>
<td>Indicates the maximum value of elements that bc(1) permits in an array.</td>
</tr>
<tr>
<td>_SC_BC_SCALE_MAX$^1$</td>
<td>Indicates the maximum scale value allowed by bc(1).</td>
</tr>
<tr>
<td>_SC_BC_STRING_MAX$^1$</td>
<td>Indicates the maximum length of a string accepted by bc(1).</td>
</tr>
<tr>
<td>_SC_COLL_WEIGHTS_MAX$^1$</td>
<td>Indicates the maximum numbers of weights that can be assigned to an entry of the LC_COLLATE order keyword in the locale definition file.</td>
</tr>
<tr>
<td>_SC_EXPR_NEST_MAX$^{1,3}$</td>
<td>Indicates the maximum number of expressions that can be nested within parentheses by expr(1).</td>
</tr>
<tr>
<td>_SC_LINE_MAX$^1$</td>
<td>Indicates the maximum length of a utility's input line length, either from standard input or from a file. This includes the length for a trailing newline.</td>
</tr>
<tr>
<td>_SC_RE_DUP_MAX$^1$</td>
<td>Indicates the maximum number of repeated occurrences of a regular expression when the interval notation {m,n} is used.</td>
</tr>
<tr>
<td>_SC_2_VERSION$^1$</td>
<td>Indicates the version of the POSIX.2 standard; it is in the format YYYYMML.</td>
</tr>
</tbody>
</table>
On Linux* systems, the corresponding macros are defined in <bits/confname.h>. The values for argument name can be obtained by using PXFCONST or IPXFCONST when passing the string names of predefined macros in <bits/confname.h>.

See Also: "PXFCONST", "IPXFCONST"

PXFTCDRAIN

POSIX Subroutine: Waits until all output written has been transmitted. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

CALL PXFTCDRAIN (ifildes, ierror)

ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.
PXFTCFLOW

POSIX Subroutine: Suspends the transmission or reception of data. This subroutine is only available on Linux* systems.
Module: USE IFPOSIX
Syntax
CALL PXFTCFLOW (ifildes, iaction, ierror)

ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.

iaction
(Input) INTEGER(4). The action to perform.

ierror
(Output) INTEGER(4). The error status.
If successful, *ierror* is set to zero; otherwise, an error code.

The PXFTCFLOW subroutine suspends or resumes transmission or reception of data from the terminal referred to by ifildes. The action performed depends on the value of iaction, which must be one of the following constant names:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCOFF</td>
<td>Output is suspended.</td>
</tr>
<tr>
<td>TCOON</td>
<td>Output is resumed.</td>
</tr>
<tr>
<td>TCIOFF</td>
<td>A STOP character is transmitted. This should cause the terminal to stop transmitting data to the system.</td>
</tr>
<tr>
<td>TCION</td>
<td>A START character is transmitted. This should cause the terminal to resume transmitting data to the system.</td>
</tr>
</tbody>
</table>

1. These names can be used in PXCONST or IPXCONST.

See Also: “IPXCONST”, “PXCONST”
PXFTCFLUSH

**POSIX Subroutine:** Discards terminal input data, output data, or both. This subroutine is only available on Linux® systems.

**Module:** USE IFPOSIX

**Syntax**

CALL PXFTCFLUSH (ifildes, iaction, ierror)

*ifildes*

(Input) INTEGER(4). The file descriptor associated with the terminal.

*iaction*

(Input) INTEGER(4). The action to perform.

*ierror*

(Output) INTEGER(4). The error status.

If successful, *ierror* is set to zero; otherwise, an error code.

The action performed depends on the value of *iaction*, which must be one of the following constant names:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCIFLUSH</td>
<td>Discards all data that has been received but not read.</td>
</tr>
<tr>
<td>TCOFLUSH</td>
<td>Discards all data that has been written but not transmitted.</td>
</tr>
<tr>
<td>TCIOFLUSH</td>
<td>Discards both data received but not read and data written but not transmitted. (Performs TCIFLUSH and TCOFLUSH actions.)</td>
</tr>
</tbody>
</table>

1. These names can be used in PXFCONST or IPXFCONST.

**See Also:** “IPXFCONST”, “PXFCONST”

PXFTCGETATTR

**POSIX Subroutine:** Returns current terminal settings. This subroutine is only available on Linux® systems.

**Module:** USE IFPOSIX

**Syntax**

CALL PXFTCGETATTR (ifildes, jtermios, ierror)
ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.

jtermios
(Output) INTEGER(4). A handle for structure termios. Stores the terminal settings.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

NOTE. To get a handle for an instance of the termios structure, use PXFSTRUCTCREATE with the string 'termios' for the structure name.

See Also: "PXFSTRUCTCREATE", "PXFTCSETATTR"

PXFTCGETPGRP

POSIX Subroutine: Gets the foreground process group ID associated with the terminal. This subroutine is only available on Linux* systems.
Module: USE IFPOSIX
Syntax
   CALL PXFTCGETPGRP (ifildes, ipgid, ierror)
ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.

ipgid
(Output) INTEGER(4). The returned process group ID.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
See Also: "PXFTCSETPGRP"

PXFTCSENDBREAK

POSIX Subroutine: Sends a break to the terminal. This subroutine is only available on Linux* systems.
Module: USE IFPOSIX

Syntax

CALL PXFTCSENDBREAK (ifildes, iduration, ierror)

ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.
iduration
(Input) INTEGER(4). Indicates how long the break should be.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFTCSENDBREAK subroutine sends a break (a '0' with a framing error) to the terminal associated with ifildes.

PXFTCSETATTR

POSIX Subroutine: Creates new terminal settings. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

CALL PXFTCSETATTR (ifildes, ioptacts, jtermios, ierror)

ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.
ioptacts
(Input) INTEGER(4). Specifies when the terminal changes take effect.
jtermios
(Input) INTEGER(4). A handle for structure termios. Contains the new terminal settings.
ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFTCSETATTR subroutine copies all terminal parameters from structure termios into the terminal associated with ifildes. When the terminal settings will change depends on the value of ioptacts, which must be one of the following constant names:
POSIX Subroutine: Sets the foreground process group ID associated with the terminal. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

```
CALL PXFTCSETPGRP (ifildes, ipgid, ierror)
```

<table>
<thead>
<tr>
<th>Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCSANOW</td>
<td>The changes occur immediately.</td>
</tr>
<tr>
<td>TCSADRAIN</td>
<td>The changes occur after all output written to ifildes has been transmitted.</td>
</tr>
<tr>
<td>TCSAFLUSH</td>
<td>The changes occur after all output written to ifildes has been transmitted, and all input that had been received but not read has been discarded.</td>
</tr>
</tbody>
</table>

1. These names can be used in PXFCONST or IPXFCONST.

NOTE. To get a handle for an instance of the termios structure, use PXFSTRUCTCREATE with the string 'termios' for the structure name.

See Also: “PXFSTRUCTCREATE”, “PXFTCGETATTR”

PXFTCSETPGRP

POSIX Subroutine: Sets the foreground process group ID associated with the terminal. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

```
CALL PXFTCSETPGRP (ifildes, ipgid, ierror)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifildes</td>
<td>(Input) INTEGER(4). The file descriptor associated with the terminal.</td>
</tr>
<tr>
<td>ipgid</td>
<td>(Input) INTEGER(4). The foreground process group ID for ifildes.</td>
</tr>
<tr>
<td>ierror</td>
<td>(Output) INTEGER(4). The error status.</td>
</tr>
</tbody>
</table>

If successful, ierror is set to zero; otherwise, an error code.

See Also: “PXFTCGETPGRP”

PXFTIME

POSIX Subroutine: Returns the current system time.

Module: USE IFPOSIX
Syntax
   CALL PXFTIME (itime, ierror)

itime
(Output) INTEGER(4). The returned system time.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFTIME subroutine returns the number of seconds since Epoch (00:00:00 UTC, January 1, 1970).

Compatibility
   CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

Example
   See the example in “PXFTIMES”.

PXFTIMES

   POSIX Subroutine: Returns process times.
Module: USE IFPOSIX
Syntax
   CALL PXFTIMES (jtms, itime, ierror)

jtms
(Output) INTEGER(4). A handle of structure tms.

itime
(Output) INTEGER(4). The returned time since system startup.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFTIMES subroutine fills the fields of structure tms associated with handle jtms with components of time that was spent by the current process. The structure fields are:
   • tms_utime – User CPU time
   • tms_stime – System CPU time
   • tms_cutime – User time of child process
   • tms_cstime – System time of child process
All members are measured in system clocks. The values can be converted to seconds by dividing by value \( ival \) returned from the following call:

\[
\text{PXFSYS CONF} (\text{IPXFCONST} (\text{'SC_CLK_TCK'}), \ ival, \ ierror)
\]

User time is the time charged for the execution of user instructions of the calling process. System time is the time charged for execution by the system on behalf of the calling process.

**NOTE.** To get a handle for an instance of the \text{tms} structure, use \text{PXFSTRUCTCREATE} with the string 'tms' for the structure name.

### Compatibility

| CONSOLE | STANDARD GRAPHICS | QUICKWIN GRAPHICS | WINDOWS | DLL | LIB |

### See Also:

"\text{PXFSTRUCTCREATE"}, \ "\text{PXFTIME}""

### Example

```fortran
program test_uname
  use ifposix
  implicit none
  integer(jhandle_size) jtms1, jtms2
  integer(4) ierror,i
  integer(4),parameter :: n=10000000
  integer(SIZEOF_CLOCK_T) itime,time1,time2, user_time1,user_time2
  integer(SIZEOF_CLOCK_T) system_time1,system_time2
  integer(4) clocks_per_sec, iname
  real(8) s, PI
  real(8) seconds_user, seconds_system
  print *,"Create a first instance for structure 'tms'"
  call PXFSTRUCTCREATE("tms",jtms1,ierror)
  if(ierror.NE.0) STOP 'Error: cannot create structure for handle jtms1'
  print *,"Create a second instance for structure 'tms'"
  call PXFSTRUCTCREATE("tms",jtms2,ierror)
  if(ierror.NE.0) then
    call PXFSTRUCTFREE(jtms1,ierror)
    STOP 'Error: cannot create structure for handle jtms2'
  end if
  print *, 'Do some calculations'
  call PXFTIMES(jtms1, itime,ierror)
```

2-350
if(ierrNE.0) then
   call PXFSTRUCTFREE(jtms1,ierror)
   call PXFSTRUCTFREE(jtms2,ierror)
   STOP 'Error: the first call of PXFTIMES fails'
end if

call PXFTIME(timel, ierror)
if(ierrNE.0) then
   call PXFSTRUCTFREE(jtms1,ierror)
   call PXFSTRUCTFREE(jtms2,ierror)
   STOP 'Error: the first call of PXFTIME fails'
end if

s = 0._8
PI = atan(1._8)*4
do i=0, n
   s = s + cos(i*PI/n)*sin(i*PI/n)
end do
print *," s=",s

call PXFTIMES(jtms2, itime,ierror)
if(ierrNE.0) then
   call PXFSTRUCTFREE(jtms1,ierror)
   call PXFSTRUCTFREE(jtms2,ierror)
   STOP 'Error: the second call of PXFTIMES fails'
end if

call PXFTIME(time2, ierror)
if(ierrNE.0) then
   call PXFSTRUCTFREE(jtms1,ierror)
   call PXFSTRUCTFREE(jtms2,ierror)
   STOP 'Error: the second call of PXFTIME fails'
end if
!DEC$ IF DEFINED(_M_IA64)
call PXFINT8GET(jtms1,"tms_utime",user_time1,ierror)
call PXFINT8GET(jtms1,"tms_stime",system_time1,ierror)
call PXFINT8GET(jtms2,"tms_utime",user_time2,ierror)
call PXFINT8GET(jtms2,"tms_stime",system_time2,ierror)
!DEC$ ELSE
call PXFINTGET(jtms1,"tms_utime",user_time1,ierror)
call PXFINTGET(jtms1,"tms_stime",system_time1,ierror)
call PXFINTGET(jtms2,"tms_utime",user_time2,ierror)
call PXFINTGET(jtms2,"tms_stime",system_time2,ierror)
!DEC$ ENDIF
iname = IPXFCONST("_SC_CLK_TCK")
call PXFSYSCONF(iname,clocks_per_sec, ierror)
if(ierror.NE.0) then
    call PXFSTRUCTFREE(jtms1,ierror)
    call PXFSTRUCTFREE(jtms2,ierror)
    STOP 'Error: the call of PXFSYSCONF fails'
end if
seconds_user = (user_time2 - user_time1)/DBLE(clocks_per_sec)
seconds_system = (system_time2 - system_time1)/DBLE(clocks_per_sec)
print *," The processor time of calculations:"
print *," User code execution(in seconds)":", seconds_user
print *," Kernal code execution(in seconds)":", seconds_system
print *," Total processor time(in seconds)":", seconds_user +
seconds_system
print *," Elapsed wall clock time(in seconds)":", time2 - timel
print *,"Free memory for instance of structure associated with jtms"
call PXFSTRUCTFREE(jtms1,ierror)
call PXFSTRUCTFREE(jtms2,ierror)
end program

PXFTTYNAM

POSIX Subroutine: Gets the terminal pathname. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
    CALL PXFTTYNAM (ifildes, s, ilen, ierror)

ifildes
(Input) INTEGER(4). The file descriptor associated with the terminal.

s
(Output) Character. The returned terminal pathname.
ilen
(Output) INTEGER(4). The length of the string stored in s.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero, otherwise, an error code.

**PXFUCOMPARE**

**POSIX Subroutine:** Compares two unsigned integers.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFUCOMPARE (i1, i2, icmpr, idiff)
```

*i1, i2*
(Input) INTEGER(4). The two unsigned integers to compare.

*icmpr*
(Output) INTEGER(4). The result of the comparison; one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>−1</td>
<td>i1 &lt; i2</td>
</tr>
<tr>
<td>0</td>
<td>i1 = i2</td>
</tr>
<tr>
<td>1</td>
<td>i1 &gt; i2</td>
</tr>
</tbody>
</table>

*idiff*
(Output) INTEGER(4). The absolute value of the difference.

The PXFUCOMPARE subroutine compares two unsigned integers and returns the absolute value of their difference into idiff.

**Compatibility**

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

**PXFUMASK**

**POSIX Subroutine:** Sets a new file creation mask and gets the previous one.

**Module:** USE IFPOSIX

**Syntax**

```fortran
CALL PXFUMASK (icmask, iprevcmask, ierror)
```
icmask
(Input) INTEGER(4). The new file creation mask.

iprevcmask
(Output) INTEGER(4). The previous file creation mask.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFUNAME

POSIX Subroutine: Gets the operation system name.
Module: USE IFPOSIX
Syntax
CALL PXFUNAME (jutsname, ierror)

jutsname
(Input) INTEGER(4). A handle of structure utsname.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.
The PXFUNAME subroutine provides information about the operation system. The information is stored in the structure associated with handle jutsname.

Compatibility
CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

Example
See the example in “PXFSTRUCTCREATE”

PXFUNLINK

POSIX Subroutine: Removes a directory entry.
Module: USE IFPOSIX
Syntax

CALL PXFUNLINK  (path, ilen, ierror)

path
(Input) Character. The name of the directory entry to remove.

ilen
(Input) INTEGER(4). The length of path string.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB

PXFUTIME

POSIX Subroutine:  Sets file access and modification times.

Module:  USE IFPOSIX

Syntax

CALL PXFUTIME (path, ilen, jutimbuf, ierror)

path
(Input) Character. The path to the file.

ilen
(Input) INTEGER(4). The length of path string.

jutimbuf
(Input) INTEGER(4). A handle of structure utimbuf.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFUTIME subroutine sets access and modification times for the file pointed to by path. The time values are retrieved from structure utimbuf.

Compatibility

CONSOLE  STANDARD GRAPHICS  QUICKWIN GRAPHICS  WINDOWS  DLL  LIB
PXFWAIT

POSIX Subroutine: Waits for a child process. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax

CALL PXFWAIT (istat, iretpid, ierror)

istat
(Output) INTEGER(4). The returned status of the child process.

iretpid
(Output) INTEGER(4). The process ID of the stopped child process.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFWAIT subroutine suspends execution of the current process until a child has exited, or
until a signal is delivered whose action terminates the current process or calls a signal handling
routine. If the child has already exited by the time of the call (a "zombie" process), a return is
immediately made. Any system resources used by the child are freed.

The subroutine returns in iretpid the value of the process ID of the child that exited, or zero if no
child was available. The returned value in istat can be used in subroutines IPXFWEXITSTATUS,
IPXFWSTOPSIG, IPXFWTERMSIG, PXFWIFEXITED, PXFWIFSIGNALLED, and
PXFWIFSTOPPED.

See Also: "PXFWAITPID", "IPXFWEXITSTATUS", "IPXFWSTOPSIG", "IPXFWTERMSIG",
"PXFWIFEXITED", "PXFWIFSIGNALLED", "PXFWIFSTOPPED"

Example

program t1
use ifposix
integer(4) ipid, istat, ierror, ipid_ret, istat_ret
print *," the child process will be born"
call PXFFORK(IPID, IERROR)
call PXFGETPID(IPID, IERROR)
if(IPID.EQ.0) then
  print *," I am a child process"
  print *," My child's pid is", IPID_RET
call PXFGETPPID(IPID, IERROR)
print *," The pid of my parent is",IPID_RET
print *," Now I have exited with code 0xABCD"
call PXFEXIT(Z'ABCD')
else
print *," I am a parent process"
print *," My parent pid is ", IPID_RET
print *," I am creating the process with pid", IPID
print *," Now I am waiting for the end of the child process"
call PXFWAIT(ISTAT, IPID_RET, IERROR)
print *," The child with pid ", IPID_RET," has exited"
if( PXFWIIFEXITED(ISTAT) ) then
print *, " The child exited normally"
istat_ret = IPXFWEXITSTATUS(ISTAT)
print 10," The low byte of the child exit code is", istat_ret
end if
end if
10 FORMAT (A,Z)
end program

**PXFWAITPID**

**POSIX Subroutine:** Waits for a specific PID. This subroutine is only available on Linux® systems.

**Module:** USE IFPOSIX

**Syntax**

```
CALL PXFWAITPID (ipid, istat, ioptions, iretpid, ierror)
```

**ipid**

(Input) INTEGER(4). The PID to wait for. One of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; −1</td>
<td>Specifies to wait for any child process whose process group ID is equal to the absolute value of ipid.</td>
</tr>
<tr>
<td>−1</td>
<td>Specifies to wait for any child process; this is the same behavior as PXFWAIT.</td>
</tr>
<tr>
<td>0</td>
<td>Specifies to wait for any child process whose process group ID is equal to that of the calling process.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>Specifies to wait for the child whose process ID is equal to the value of ipid.</td>
</tr>
</tbody>
</table>
istat
(Output) INTEGER(4). The returned status of the child process.

ioptions
(Input) INTEGER(4). One or more of the following constant values (which can be passed to PXFCONST or IPXFCONST):

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WNOHANG</td>
<td>Specifies to return immediately if no child process has exited.</td>
</tr>
<tr>
<td>WUNTRACED</td>
<td>Specifies to return for child processes that have stopped, and whose status has not been reported.</td>
</tr>
</tbody>
</table>

iretpid
(Output) INTEGER(4). The PID of the stopped child process.

ierror
(Output) INTEGER(4). The error status.
If successful, ierror is set to zero; otherwise, an error code.

The PXFWAITPID subroutine suspends execution of the current process until the child specified by ipid has exited, or until a signal is delivered whose action terminates the current process or calls a signal handling routine. If the child specified by ipid has already exited by the time of the call (a "zombie" process), a return is immediately made. Any system resources used by the child are freed.

The returned value in istat can be used in subroutines IPXFWEXITSTATUS, IPXFWSTOPSIG, IPXFWTERMSIG, PXFWFEXITED, PXFWFISIGNALLED, and PXFWFSTOPPED.

See Also: “PXFWAIT”, “IPXFWEXITSTATUS”, “IPXFWSTOPSIG”, “IPXFWTERMSIG”, “PXFWFEXITED”, “PXFWFISIGNALLED”, “PXFWFSTOPPED”

PXFWFEXITED

POSIX Function: Determines if a child process has exited. This subroutine is only available on Linux* systems.

Module: USE IFPOSIX

Syntax
result = PXFWFEXITED (istat)
**istat**

(Output) INTEGER(4). The status of the child process (obtained from PXFWAIT or PXFWAITPID).

**Results:**
The result type is logical. The result value is .TRUE. if the child process has exited normally; otherwise, .FALSE..

**See Also:** "PXFWIFSIGNALED", "PXFWIFSTOPPED"

**Example**

```fortran
program t1
  use ifposix
  integer(4) ipid, istat, ierror, ipid_ret, istat_ret
  print *, "the child process will be born"
  call PXFFORK(IPID, IERROR)
  call PXFGETPID(IPID_RET,IERROR)
  if(IPID.EQ.0) then
    print *, "I am a child process"
    print *, "My child's pid is", IPID_RET
    call PXFGETPPID(IPID_RET,IERROR)
    print *, "The pid of my parent is",IPID_RET
    print *, "Now I have exited with code 0xABCD"
    call PXFEXIT(Z'ABCD')
  else
    print *, "I am a parent process"
    print *, "My parent pid is ", IPID_RET
    print *, "I am creating the process with pid", IPID
    print *, "Now I am waiting for the end of the child process"
    call PXFWAIT(ISTAT, IPID_RET, IERROR)
    print *, "The child with pid ", IPID_RET," has exited"
    if( PXFWIFEXITED(ISTAT) ) then
      print *, "The child exited normally"
      istat_ret = IPXFWEXITSTATUS(ISTAT)
      print 10," The low byte of the child exit code is", istat_ret
    end if
  end if
10 FORMAT (A,Z)
end program
```
PXFWIFSIGNALED

**POSIX Function:** Determines if a child process has exited because of a signal. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
result = PXFWIFSIGNALED (istat)
```

*istat* (Output) INTEGER(4). The status of the child process (obtained from PXFWAIT or PXFWAITPID).

**Results:**
The result type is logical. The result value is .TRUE. if the child process has exited because of a signal that was not caught; otherwise, .FALSE..

**See Also:** "PXFWIFEXITED", "PXFWIFSTOPPED"

PXFWIFSTOPPED

**POSIX Function:** Determines if a child process has stopped. This subroutine is only available on Linux* systems.

**Module:** USE IFPOSIX

**Syntax**

```fortran
result = PXFWIFSTOPPED (istat)
```

*istat* (Output) INTEGER(4). The status of the child process (obtained from PXFWAIT or PXFWAITPID).

**Results:**
The result type is logical. The result value is .TRUE. if the child process has stopped; otherwise, .FALSE..

**See Also:** "PXFWIFEXITED", "PXFWIFSIGNALED"

PXFWRITE

**POSIX Subroutine:** Writes to a file.

**Module:** USE IFPOSIX
Syntax

CALL PXFWRITE (ifildes, buf, nbyte, nwritten, ierror)

ifildes
(Input) INTEGER(4). The file descriptor to be written to.

buf
(Input) Character. The buffer that contains the data to write into the file.

nbyte
(Input) INTEGER(4). The number of bytes to write.

nwritten
(Output) INTEGER(4). The returned number of bytes written.

ierror
(Output) INTEGER(4). The error status.

If successful, ierror is set to zero; otherwise, an error code.

The PXFWRITE subroutine writes nbyte bytes from the storage buf into a file specified by file descriptor ifildes. The subroutine returns the total number of bytes read into nwritten. If no error occurs, the value of nwritten will equal the value of nbyte.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “PXFREAD”

QRANSET

Portability Subroutine: Sets the seed for a sequence of pseudo-random numbers.

Module: USE IFPORT

Syntax

CALL QRANSET (rseed)

rseed
(Input) INTEGER(4). The reset value for the seed.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
QSORT

Portability Subroutine: Performs a quick sort on an array of rank one.

Module: USE IFPORT

Syntax

CALL QSORT (array, len, isize, compar)

array
(Input) Any type. One-dimensional array to be sorted.

If the data type does not conform to one of the predefined interfaces for QSORT, you may have to create a new interface (see the Note below).

len
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. Number of elements in array.

isize
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. Size, in bytes, of a single element of array is:

• 4 if array is of type REAL(4)
• 8 if array is of type REAL(8) or complex
• 16 if array is of type COMPLEX(8)

compar
(Input) INTEGER(2). Name of a user-defined ordering function that determines sort order. The type declaration of compar takes the form:

   INTEGER(2) FUNCTION compar (arg1, arg2)

where arg1 and arg2 have the same type as array (above). Once you have created an ordering scheme, implement your sorting function so that it returns the following:

• Negative if arg1 should precede arg2
• Zero if arg1 is equivalent to arg2
• Positive if arg1 should follow arg2

Dummy argument compar must be declared as external.

In place of an INTEGER kind, you can specify the constant SIZEOF_SIZE_T, defined in IFPORT.F90, for argument len or isize. Use of this constant ensures correct compilation.
NOTE. If you use QSORT with different data types, your program must have a USE IFPORT statement so that all the calls work correctly. In addition, if you wish to use QSORT with a derived type or a type that is not in the predefined interfaces, you must include an overload for the generic subroutine QSORT. Examples of how to do this are in the portability module’s source file, IFPORT.F90.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

Example

PROGRAM SORTQ
  USE IFPORT
  integer(2), external :: cmp_function
  integer(2) insort(26), i
  integer (SIZEOF_SIZE_T) array_len, array_size
  array_len = 26
  array_size = 2
  do i=90,65,-1
    insort(i-64)=91 - i
  end do
  print *, "Before: ", insort
  CALL qsort(insort,array_len,array_size,cmp_function)
  print *, 'After: ', insort
END
!
  integer(2) function cmp_function(a1, a2)
  integer(2) a1, a2
  cmp_function=a1-a2
end function

RAISEQQ

Portability Function: Sends a signal to the executing program.
Module: USE IFPORT

Syntax

\[
\text{result} = \text{RAISEQQ} (\text{sig})
\]

\(\text{sig}\)

(Input) INTEGER(4). Signal to raise. One of the following constants (defined in IFPORT.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 with exit code 3.

Results:

The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero.

If a signal-handling routine for \(\text{sig}\) 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).

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SIGNALQQ”, “SIGNAL”, “KILL”

Example

See the example in “SIGNALQQ”.

RAND, RANDOM

Portability Functions: Return real random numbers in the range 0.0 through 1.0.

Module: USE IFPORT

Syntax

\[
\text{result} = \text{RAND} ([\text{iflag}])
\]

\(\text{result} = \text{RANDOM} ([\text{iflag}])
\]

\(\text{iflag}\)

(Input) INTEGER(4). Optional for RAND. Controls the way the random number is selected.
Results:
The result type is REAL(4). RAND and RANDOM return random numbers in the range 0.0 through 1.0.

<table>
<thead>
<tr>
<th>Value of iflag</th>
<th>Selection Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The generator is restarted and the first random 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 iflag, restarted, and the first random value is selected.</td>
</tr>
</tbody>
</table>

When RAND is called without an argument, iflag is assumed to be 0.

There is no difference between RAND and RANDOM. Both functions are included to ensure portability of existing code that references one or both of them. The intrinsic functions RANDOM_NUMBER and RANDOM_SEED provide the same functionality.

You can use SRAND to restart the pseudorandom number generator used by RAND.

**NOTE.** RANDOM is available as a function or subroutine.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "RANDOM", "SRAND", the RANDOM_NUMBER and RANDOM_SEED intrinsic subroutines in the Language Reference

Example

The following example shows how to use both the RANDOM function and the RANDOM subroutine:

```fortran
use ifport
real(4) ranval
!from libifcore.lib
call seed(1995)  ! initialize
!also from for_m_irand.c in libfor
call random(ranval) ! get next random number
print *,ranval
!from libifport.lib
ranval = random(1)  ! initialize
! same
```
ranval = random(0)  ! get next random number
print *, ranval
end

RANDOM

Portability Subroutine: Returns a pseudorandom number greater than or equal to zero and less than one from the uniform distribution.

Module: USE IFPORT

Syntax

CALL RANDOM (ranval)

ranval
(Output) REAL(4). Pseudorandom number, 0 ≤ ranval < 1, from the uniform distribution.

A given seed always produces the same sequence of values from RANDOM.

If SEED is not called before the first call to RANDOM, RANDOM begins with a seed value of one. If a program must have a different pseudorandom sequence each time it runs, pass the constant RND$TIMESEED (defined in IFQWIN.F90) to SEED before the first call to RANDOM.

The portability routines DRAND, DRANDM, IRAND, IRANDM, RAN, RAND, and the RANDOM portability function and subroutine use the same algorithms and thus return the same answers. They are all compatible and can be used interchangeably. The algorithm used is a "Prime Modulus M Multiplicative Linear Congruential Generator," a modified version of the random number generator by Park and Miller in "Random Number Generators: Good Ones Are Hard to Find," CACM, October 1988, Vol. 31, No. 10.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

See Also: "SEED", "DRAND, DRANDM", "IRAND, IRANDM", "RAND, RANDOM", the RANDOM_NUMBER intrinsic subroutine in the Language Reference

Example

USE IFPORT
REAL (4) ran
CALL SEED(1995)
CALL RANDOM(ran)

See also the second example in "RAND, RANDOM", which shows how to use both the RANDOM function and the RANDOM subroutine.
Descriptions of the Library Routines

RANF

**Portability Function:** Generates a random number between 0.0 and RAND_MAX.

**Module:** USE IFPORT

**Syntax**

```
result = RANF ( )
```

**Results:**

The result type is REAL(4). The result value is a single-precision pseudo-random number between 0.0 and RAND_MAX as defined in the C library, normally 0x7FFF 215–1.

The initial seed is set by the following:

```
CALL SRAND (ISEED)
```

where ISEED is type INTEGER(4).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

RANGET

**Portability Subroutine:** Returns the current seed.

**Module:** USE IFPORT

**Syntax**

```
CALL RANGET (seed)
```

**seed**

(Output) INTEGER(4). The current seed value.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "DRANSET", "RANSET"

RANSET

**Portability Subroutine:** Sets the seed for the random number generator.

**Module:** USE IFPORT

**Syntax**

```
CALL RANSET (seed)
```

---

2-367
seed
(Input) REAL(4). The reset value for the seed.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “RANGET”

RECTANGLE, RECTANGLE_W

Graphics Functions: Draw a rectangle using the current graphics color, logical write mode, and line style. These functions are only available on Windows* systems.

Module: USE IFQWIN

Syntax
result = RECTANGLE (control, x1, y1, x2, y2)
result = RECTANGLE_W (control, wx1, wy1, wx2, wy2)

control
(Input) INTEGER(2). Fill flag. One of the following symbolic constants (defined in IFQWIN.F90):
• $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.

wx1, wy1
(Input) REAL(8). Window coordinates for upper-left corner of rectangle.

wx2, wy2
(Input) REAL(8). Window coordinates for lower-right corner of rectangle.

Results:
The result type is INTEGER(2). The result is nonzero if successful; otherwise, 0.

The RECTANGLE function uses the viewport-coordinate system. The viewport coordinates (x1, y1) and (x2, y2) are the diagonally opposed corners of the rectangle.

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.

NOTE. The RECTANGLE routine described here is a QuickWin routine. If you are trying to use the Microsoft® Platform SDK version of the Rectangle routine by including the IFWIN module, you need to specify the routine name as MSFWINSREncle. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

Compatibility
STANDARD GRAPHICS   QUICKWIN GRAPHICS   LIB


Example
This program draws the rectangle shown below.

! Build as a QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(2) dummy, x1, y1, x2, y2
x1 = 80; y1 = 50
x2 = 240; y2 = 150
dummy = RECTANGLE( $GBORDER, x1, y1, x2, y2 )
END
REGISTERMOUSEEVENT

QuickWin Function: Registers the application-supplied callback routine to be called when a specified mouse event occurs in a specified window. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = REGISTERMOUSEEVENT (unit, mouseevents, callbackroutine)

unit
(Input) INTEGER(4). 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 (defined in IFQWIN.F90) 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) Routine to be called on the specified mouse event in the specified window. It must be declared EXTERNAL. For a prototype mouse callback routine, see "Using QuickWin" in Building Applications.

Results:
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.

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.

**Compatibility**

QUIKWIN GRAPHICS LIB

**See Also:** "UNREGISTERMOUSEEVENT", "WAITONMOUSEEVENT", "Using QuickWin" in *Building Applications*

**Example**

The following example registers the routine CALCLULATE, to be called when the user double-clicks the left mouse button while the mouse cursor is in the child window opened as unit 4:

```fortran
USE IFQWIN
INTEGER(4) result
OPEN (4, FILE= 'USER')
...  
result = REGISTERMOUSEEVENT (4, MOUSE$LBUTTONDBLCLK, CALCULATE)
```

**REMAPALLPALETTERGB**

**Graphics Function:** Remaps a set of Red-Green-Blue (RGB) color values to indexes recognized by the video hardware. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = REMAPALLPALETTERGB (colors)
```

**colors**

(Input) INTEGER(4). Ordered array of RGB color values to be mapped in order to indexes. Must hold 0–255 elements.

**Results:**

The result type is INTEGER(4). The result is 0 if successful; otherwise, –1.

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. The 16 default colors are provided with symbolic constants in `IFQWIN.F90`.
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.

Any RGB colors can be mapped into the 236 palette indexes. Thus, you could specify a palette with 236 shades of red. For further details on using different color procedures see "Adding Color" in Building Applications.

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 or REMAPPALETTERGB, red is the rightmost byte, followed by green and blue. The RGB value’s internal structure is as follows:

```
Bit 31 (MSB)  24  23  16  15   8  7  0
RGB  O O O O O O O B B B B B B G G G G G G G G R R R R R R R R
```

Larger numbers correspond to stronger color intensity with binary 11111111 (hex FF) the maximum for each of the three components. For example, Z'008080' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'FFFFFF' full-intensity for all three, resulting in bright white.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB
See Also: "REMAPPALETTERGB", "SETBKCOLORRGB", "SETCOLORRGB", "SETBKCOLOR", "SETCOLOR"

Example

! Build as QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(4) colors(3)
INTEGER(2) status
colors(1) = Z'00FFFF' ! yellow
colors(2) = Z'FFFFFFF' ! bright white
colors(3) = 0       ! black
status = REMAPALLPALETTERGB(colors)
status = REMAPPALETTERGB(INT2(47), Z'45A315')
END

REMAPPALETTERGB

Graphics Function: Remaps one color index to an RGB color value. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = REMAPPALETTERGB (index, color)

index
(Input) INTEGER(4). Color index to be reassigned an RGB color.

color
(Input) INTEGER(4). RGB color value to assign to a color index.

Results:
The result type is INTEGER(4). The result value is the previous color assigned to the index.

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. The 16 default colors are provided with symbolic constants in IFQWIN.F90.

<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>
</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`.

Any RGB colors can be mapped into the 236 palette indexes. Thus, you could specify a palette with 236 shades of red. For further details on using different color procedures see "Adding Color" in Building Applications.

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` or `REMAPPALETTERGB`, red is the rightmost byte, followed by green and blue. The RGB value’s internal structure is as follows:

```
   Bit 31 (MSB) 24 23 16 15 8 7 0
   RGB 00000000 BBBBBBBB GGGGGGGG GGGGGGGG RRRRRRRR
```

Larger numbers correspond to stronger color intensity with binary 11111111 (hex FF) the maximum for each of the three components. For example, Z’008080’ yields full-intensity red, Z’00FF00’ full-intensity green, Z’FF0000’ full-intensity blue, and Z’FFFFFF’ full-intensity for all three, resulting in bright white.

**Compatibility**

<table>
<thead>
<tr>
<th>STANDARD GRAPHICS</th>
<th>QUICKWIN GRAPHICS</th>
<th>LIB</th>
</tr>
</thead>
</table>

**See Also:** "REMAPALLPALETTERGB", "SETBKCOLORRGB", "SETCOLORRGB", "SETBKCOLOR", "SETCOLOR"

**Example**

See the example in "REMAPALLPALETTERGB".
RENAME

**Portability Function:** Renames a file.

**Module:** USE IFPORT

**Syntax**

```
result = RENAME (from, to)
```

- **from** (Input) Character*(*). Path of an existing file.
- **to** (Input) Character*(*). The new path for the file (see Caution note below).

**Results:**

The result type is INTEGER(4). The result is zero if successful; otherwise, an error code, such as:

- **EACCES** – The file or directory specified by `to` could not be created (invalid path). This error is also returned if the drive specified is not currently connected to a device.
- **ENOENT** – The file or path specified by `from` could not be found.
- **EXDEV** – Attempt to move a file to a different device.

---

⚠️ **CAUTION.** *This routine can cause data to be lost. If the file specified in “to” already exists, RENAME deletes the pre-existing file.*

---

It is possible to rename a file to itself without error.

The paths can use forward (/) or backward (\) slashes as path separators and can include drive letters (if permitted by your operating system).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “RENAMEFILEQQ”

RENAMEFILEQQ

**Portability Function:** Renames a file.

**Module:** USE IFPORT

**Syntax**

```
result = RENAMEFILEQQ (oldname, newname)
```

---
2

Intel Fortran Libraries Reference

oldname
(Input) Character*(*). Current name of the file to be renamed.
newname
(Input) Character*(*). New name of the file to be renamed.
Results:
The result type is LOGICAL(4). 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, 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.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS

See Also: “FINDFILEQQ”, “RENAME”, “GETLASTERRORQQ”
Example
USE IFPORT
USE IFCORE
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

2-376

DLL

LIB


RGBTOINTEGER

**QuickWin Function:** Converts three integers specifying red, green, and blue color intensities into a four-byte RGB integer for use with RGB functions and subroutines. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```plaintext
result = RGBTOINTEGER (red, green, blue)
```

**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.

**Results:**

The result type is INTEGER(4). The result is the combined RGB color value.

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 with RGBTOINTEGER, red is the rightmost byte, followed by green and blue. The RGB value’s internal structure is as follows:

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O</td>
<td>O O B B B B B B</td>
<td>G G G G G G G R R R R R R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Larger numbers correspond to stronger color intensity with binary 1111111 (hex Z'FF') the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'FFFF00' full-intensity for all three, resulting in bright white.

**Compatibility**

QUICKWIN GRAPHICS LIB

Example

! Build as a QuickWin App.
USE IFQWIN
INTEGER r, g, b, rgb, result
INTEGER(2) status
r = Z‘F0’
g = Z‘F0’
b = 0
rgb = RGBTOINTEGER(r, g, b)
result = SETCOLORRGB(rgb)
status = ELLIPSE($GFILLINTERIOR, INT2(40), INT2(55), &
INT2(90), INT2(85))
END

RINDEX

Portability Function: Locates the index of the last occurrence of a substring within a string.
Module: USE IFPORT

Syntax

result = RINDEX (string, substr)

string
(Input) Character*(*). Original string to search.
substr
(Input) Character*(*). String to search for.

Results:
The result type is INTEGER(4). The result is the starting position of the final occurrence of substr in string. The result is zero if substr does not occur in string.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: the INDEX intrinsic function in the Language Reference

Example
USE IFPORT
character*80 mainstring
character*4 shortstr
integer(4) where
mainstring="Hello Hello Hello Hello There There There"
shortstr="Hello"
where=rindex(mainstring,shortstr)
! where is 19

RTC

Portability Function: Returns the number of seconds elapsed since a specific Greenwich mean time.

Module: USE IFPORT

Syntax
result = RTC ()

Results:
The result type is REAL(8). The result is the number of seconds elapsed since 00:00:00 Greenwich mean time, January 1, 1970.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "TIMEF", the DATE_AND_TIME intrinsic procedure in the Language Reference

Example
USE IFPORT
real(8) s, s1, time_spent
INTEGER(4) i, j
s = RTC()
call sleep(4)
s1 = RTC( )
time_spent = s1 - s
PRINT *, 'It took ',time_spent, 'seconds to run.'

RUNQQ

Portability Function: Executes another program and waits for it to complete.

Module: USE IFPORT
Syntax

```fortran
result = RUNQQ (filename, cmdline)
```

`filename`
(Input) Character*(*) . File name of a program to be executed.

`cmdline`
(Input) Character*(*) . 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “SYSTEM”, “NLSEnumCodepages”

**Example**

```fortran
USE IFPORT
INTEGER(2) result
result = RUNQQ('myprog', '-c -r')
END
```

See also the example in “NLSEnumCodepages”.

---

**SAVEIMAGE, SAVEIMAGE_W**

**Graphics Functions:** Save an image from a specified portion of the screen into a Windows bitmap file. These functions are only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = SAVEIMAGE (filename, ulxcoord, ulycoord, lrxcoord, lrycoord)
result = SAVEIMAGE_W (filename, ulxwcoord, ulwycoord, lrxwcoord, lrywcoord)
```

`filename`
(Input) Character*(*) . Path of the bitmap file.
ulxcoord, ulycoord
(Input) INTEGER(4). Viewport coordinates for upper-left corner of the screen image to be captured.

lrxcoord, lrycoord
(Input) INTEGER(4). Viewport coordinates for lower-right corner of the screen image to be captured.

ulwcoord, ulwcoord
(Input) REAL(8). Window coordinates for upper-left corner of the screen image to be captured.

lrwcoord, lrwcoord
(Input) REAL(8). Window coordinates for lower-right corner of the screen image to be captured.

**Results:**
The result type is INTEGER(4). The result is zero if successful; otherwise, a negative value.
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 filename. The image is stored with a palette containing the colors displayed on the screen.
SAVEIMAGE defines the bounding rectangle in viewport coordinates. SAVEIMAGE_W defines the bounding rectangle in window coordinates.

**Compatibility**
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETIMAGE, GETIMAGE_W”, “IMAGE_SIZE, IMAGE_SIZE_W”, “LOADIMAGE, LOADIMAGE_W”, “PUTIMAGE, PUTIMAGE_W”

**SCANENV**

**Portability Subroutine:** Scans the environment for the value of an environment variable.

**Module:** USE IFPORT

**Syntax**

```
CALL SCANENV (envname, envtext, envvalue)
```

**envname**
(Input) Character*(*) . Contains the name of an environment variable you need to find the value for.

**envtext**
(Output) Character*(*) . Set to the full text of the environment variable if found, or to ‘ ’ if nothing is found.
envvalue
(Output) Character*(*) Set to the value associated with the environment variable if found or to '' if nothing is found.

SCANENV scans for an environment variable that matches envname and returns the value or string it is set to.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

SCROLLTEXTWINDOW

Graphics Subroutine: Scrolls the contents of a text window. This subroutine is only available on Windows® systems.

Module: USE IFQWIN

Syntax
CALL SCROLLTEXTWINDOW (rows)

rows
(Input) INTEGER(2). Number of rows to scroll.

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 (SGWINDOW). A value of 0 for rows has no effect.

Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: CLEARSCREEN, GETTEXTPOSITION, GETTEXTWINDOW, GRSTATUS, OUTTEXT, SETTEXTPOSITION, SETTEXTWINDOW, WRAPON

Example
! Build as QuickWin or Standard Graphics app.
USE IFQWIN
INTEGER(2) row, istat
CHARACTER(18) string
TYPE (rccoord) oldpos

CALL SETTEXTWINDOW (INT2(1), INT2(0), &
                     INT2(25), INT2(80))
CALL CLEARSCREEN ( $GCLEARSCREEN )

CALL SETTEXTPOSITION (INT2(1), INT2(1), oldpos)
DO row = 1, 6
    string = 'Hello, World # ' 
    CALL SETTEXTPOSITION( row, INT2(1), oldpos )
    WRITE(string(15:16), '(I2)') row
    CALL OUTTEXT( string )
END DO
istat = displaycursor($GCURSORON)
WRITE(*,'(1x,A)') 'Hit ENTER'
READ (*,*) ! wait for ENTER
! Scroll window down 4 lines
CALL SCROLLTEXTWINDOW(INT2( -4) )
CALL SETTEXTPOSITION (INT2(10), INT2(18), oldpos)
WRITE(*,'(2X,A)') "Hit ENTER"
READ( *,* ) ! wait for ENTER
! Scroll window up 5 lines
CALL SCROLLTEXTWINDOW( INT2(5 ) )
END

SCWRQQ

Portability Subroutine: Returns the floating-point processor control word.
Module: USE IFPORT
Syntax
   CALL SCWRQQ (control)
control
(Output) INTEGER(2). Floating-point processor control word.
SCRWQQ performs the same function as the run-time subroutine GETCONTROLFPQQ, and is
provided for compatibility.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “GETCONTROLFPQQ”, “LCWRQQ”
Example
See the example in “LCWRQQ”.
SECNDS

Portability Function: Returns the number of seconds that have elapsed since midnight, less the value of its argument.

Module: USE IFPORT

Syntax:

\[
\text{result} = \text{SECNDS} (r)
\]

\(r\) (Input) REAL(4). Number of seconds, precise to a hundredth of a second (0.01), to be subtracted.

Results:
The result type is REAL(4). The result value is the number of seconds that have elapsed since midnight, minus \(r\), with a precision of a hundredth of a second (0.01).

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.

NOTE. SECNDS is an intrinsic procedure unless you specify USE IFPORT.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: “RTC”, “TIME”, the DATE_AND_TIME and SYSTEM_CLOCK subroutines and the SECNDS function in the Language Reference

Example

USE IFPORT
REAL(4) s
INTEGER(4) i, j
s = SECNDS(0.0)
DO I = 1, 100000
J = J + 1
END DO
s = SECNDS(s)
PRINT *, 'It took ', s, ' seconds to run.'

SEED

Portability Subroutine: Changes the starting point of the pseudorandom number generator.
Module: USE IFPORT

Syntax
CALL SEED (iseed)

iseed
(Input) INTEGER(4). Starting point for RANDOM.
SEED uses iseed to establish the starting point of the pseudorandom number generator. A given seed always produces the same sequence of values from RANDOM.
If SEED is not called before the first call to RANDOM, RANDOM always begins with a seed value of one. If a program must have a different pseudorandom sequence each time it runs, pass the constant RNDSTIMESEED (defined in IFPORT.F90) to the SEED routine before the first call to RANDOM.
This routine is not thread-safe.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “RANDOM”, the RANDOM_SEED and RANDOM_NUMBER intrinsic subroutines in the Language Reference

Example
USE IFPORT
REAL myrand
CALL SEED(7531)
CALL RANDOM(myrand)

SETACTIVEQQ

QuickWin Function: Makes a child window active, but does not give it focus. This function is only available on Windows* systems.
Module: USE IFQWIN

Syntax
result = SETACTIVEQQ (unit)

unit
(Input) INTEGER(4). Unit number of the child window to be made active.

Results:
The result type is INTEGER(4). The result is 1 if successful; otherwise, 0.
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 IOFOCUS='FALSE'. By default, IOFOCUS='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 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.

Compatibility
QUICKWIN GRAPHICS LIB
See Also: “GETACTIVEQQ”, “FOCUSQQ”, “INQFOCUSQQ”, "Using QuickWin" in Building Applications

SETBKCOLOR

Graphics Function: Sets the current background color index for both text and graphics. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax
   result = SETBKCOLOR (color)

   color
   (Input) INTEGER(4). Color index to set the background color to.

Results:
   The result type is INTEGER(4). The result is the previous background color index.

   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.

**NOTE.** The SETBKCOLOR routine described here is a QuickWin routine. If you are trying to use the Microsoft® Platform SDK version of the SetBkColor routine by including the IFWIN module, you need to specify the routine name as MSFWIN$SetBkColor. For more information, see “Special Naming Convention for Certain QuickWin and Win32 Graphics Routines” in Building Applications.

**Compatibility**
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: “SETBKCOLORRGB”, “GETBKCOLOR”, “REMAPALLPALETTERGB”, “REMAPPALETTERGB”, “SETCOLOR”, “SETTEXTCOLOR”

**Example**
USE IFQWIN
INTEGER(4) i
i = SETBKCOLOR(14)

**SETBKCOLORRGB**

**Graphics Function:** Sets the current background color to the given Red-Green-Blue (RGB) value. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

\[
\text{result} = \text{SETBKCOLORRGB}\left(\text{color}\right)
\]

**color**

(Input) INTEGER(4). RGB color value to set the background color to. Range and result depend on the system’s display adapter.
Results:
The result type is INTEGER(4). The result is the previous background RGB color value.

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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O O O O O</td>
<td>B B B B B B</td>
<td>G G G G G G G G</td>
<td>R R R R R R R R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Larger numbers correspond to stronger color intensity with binary 1111111 (hex Z’FF’) the
maximum for each of the three components. For example, Z’0000FF’ yields full-intensity red,
Z’00FF00’ full-intensity green, Z’FF0000’ full-intensity blue, and Z’FFFFFF’ full-intensity for all
three, resulting in bright white.

The default background color is value 0, which is black. Changing the background color value
does not change the screen immediately, but becomes effective when CLEARSCREEN is
executed or when doing text input or output such as READ, WRITE, or OUTTEXT. The graphics
output function OUTGTEXT does not affect the color of the background.

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 restricted 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.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also:  “GETBKCOLORRGB” “SETCOLORRGB”, “SETTEXTCOLORRGB”,
“SETPIXELRGB, SETPIXELRGB_W”, “SETPIXELRGB”, “SETBKCOLOR”

Example
! Build as a QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(4) oldcolor
INTEGER(2) status, x1, y1, x2, y2
x1 = 80; y1 = 50
x2 = 240; y2 = 150
oldcolor = SETBKCOLORRGB(Z'FF0000') !blue
oldcolor = SETCOLORRGB(Z'FF') ! red
CALL CLEARSCREEN ($GCLEARSCREEN)
status = ELLIPSE($GBORDER, x1, y1, x2, y2)
END

SETCLIPRGN

Graphics Subroutine: Limits graphics output to part of the screen. This subroutine is only available on Windows® systems.
Module: USE IFQWIN

Syntax
CALL SETCLIPRGN (x1, y1, x2, y2)

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.

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.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETPHYSCOORD”, “GRSTATUS”, “SETTEXTWINDOW”, “SETVIEWORG”, “SETVIEWPORT”, “SETWINDOW”

Example
This program draws an ellipse lying partly within a clipping region, as shown below.
! Build as QuickWin or Standard Graphics ap.
USE IFQWIN
INTEGER(2) status, x1, y1, x2, y2
INTEGER(4) oldcolor
x1 = 10;  y1 = 50
x2 = 170; y2 = 150
! Draw full ellipse in white
status = ELLIPSE($GBORDER, x1, y1, x2, y2)
oldcolor = SETCOLORRGB(Z'FF0000') !blue
WRITE(*,*) "Hit enter"
READ(*,*)
CALL CLEARSCREEN($GCLEARSCREEN) ! clear screen
CALL SETCLIPRGN( INT2(0), INT2(0), &
        INT2(150), INT2(125))
! only part of ellipse inside clip region drawn now
status = ELLIPSE($GBORDER, x1, y1, x2, y2)
END
The following figure shows the output of this program.

**SETCOLOR**

**Graphics Function:** Sets the current graphics color index. This function is only available on Windows* systems.

**Module:** USE IFQWIN
Syntax

\[
\text{result} = \text{SETCOLOR}(color)
\]

\(color\)  
(Input) INTEGER(2). Color index to set the current graphics color to.

Results:
The result type is INTEGER(2). The result is the previous color index if successful; otherwise, –1.

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.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “SETCOLORRGB”, “GETCOLOR”, “REMAPPALETTERGB”, “SETBKCOLOR”, “SETTEXTCOLOR”, “SETPIXEL, SETPIXEL_W”, “SETPIXELS”

Example

USE IFQWIN
INTEGER(2) color, oldcolor
LOGICAL status
TYPE (windowconfig) wc

status = GETWINDOWCONFIG(wc)
color = wc\%numcolors - 1
oldcolor = SETCOLOR(color)
END

SETCOLORRGB

Graphics Function: Sets the current graphics color to the specified Red-Green-Blue (RGB) value. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

\[
\text{result} = \text{SETCOLORRGB}(color)
\]
color

(Input) INTEGER(4). RGB color value to set the current graphics color to. Range and result depend on the system’s display adapter.

Results:
The result type is INTEGER(4). The result is the previous RGB color value.

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:

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O O O O O</td>
<td>B B B B B B</td>
<td>G G G G G G</td>
<td>G G G G G G</td>
<td>R R R R R R</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Larger numbers correspond to stronger color intensity with binary 1111111 (hex Z’FF’) the maximum for each of the three components. For example, Z’0000FF’ yields full-intensity red, Z’00FF00’ full-intensity green, Z’FF0000’ full-intensity blue, and Z’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 restricted 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.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “SETBKCOLORRGB”, “SETTEXTCOLORRGB”, “FLOODFILLRGB, FLOODFILLRGB_W”, “SETCOLOR”, “REMAPPALETTERGB”, “SETPIXELRGB, SETPIXELRGB_W”, “SETPIXELSRGB”
Example

! Build as a QuickWin or Standard Graphics App.
USE IFQWIN
INTEGER(2) numfonts
INTEGER(4) oldcolor
TYPE (xycoord) xy
numfonts = INITIALIZFONTS( )
oldcolor = SETCOLORRGB(Z'0000FF') ! red
oldcolor = SETBKCOLORRGB(Z'00FF00') ! green
CALL MOVETO(INT2(200), INT2(100), xy)
CALL OUTGTEXT("hello, world")
END

SETCONTROLFPQQ

Portability Subroutine: Sets the value of the floating-point processor control word.
Module: USE IFPORT

Syntax

CALL SETCONTROLFPQQ (controlword)

controlword
(Input) INTEGER(2). Floating-point processor control word.

The floating-point control word specifies how various exception conditions are handled by the
floating-point math coprocessor, sets the floating-point precision, and specifies the floating-point
rounding mechanism used.

The control word can be any of the following constants (defined in IFPORT.F90):

<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>
</tbody>
</table>
The defaults for the floating-point control word 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.

Setting the floating-point precision and rounding mechanism can be useful if you are reusing old
code that is 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. Most users do not need to change the default
settings. 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.

For a full discussion of the floating-point control word, exceptions, and error handling, see "The
Floating-Point Environment" in Building Applications.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Hex value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

NOTE. The Intel® Visual Fortran exception handler allows for software
masking of invalid operations, but does not allow the math chip to mask them.
If you choose to use the software masking, be aware that this can affect
program performance if you compile code written for Visual Fortran with
another compiler.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: “GETCONTROLFPQQ”, “GETSTATUSFPQQ”, “LCWRQQ”, “SCWRQQ”, “CLEARSTATUSFPQQ”

Example

USE IFPORT
INTEGER(2) status, control, controlo

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

Portability Function: Sets the system date.

Module: USE IFPORT

Syntax

result = SETDAT (iyr, imon, iday)

iyr
(Input) INTEGER(2) or INTEGER(4). Year (xxxx AD).

imon
(Input) INTEGER(2) or INTEGER(4). Month (1-12).

iday
(Input) INTEGER(2) or INTEGER(4). Day of the month (1-31).

Results:
The result type is LOGICAL(4). The result is .TRUE. if the system date is changed; .FALSE. if no change is made.
Actual arguments of the function SETDAT can be any valid INTEGER(2) or INTEGER(4) expression. Refer to your operating system documentation for the range of permitted dates.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "GETDAT", "GETTIM", "SETTIM"

**Example**

USE IFPORT
LOGICAL(4) success
success = SETDAT(INT2(1997+1), INT2(2*3), INT2(30))
END

---

## SETENVQQ

**Portability Function:** Sets the value of an existing environment variable, or adds and sets a new environment variable.

**Module:** USE IFPORT

**Syntax**

\[
\text{result} = \text{SETENVQQ} (\text{varname} = \text{value})
\]

\text{varname} = \text{value}

(Input) Character*(*) String containing both the name and the value of the variable to be added or modified. Must be in the form: \text{varname} = \text{value}, where \text{varname} is the name of an environment variable and \text{value} is the value being assigned to it.

**Results:**

The result is of type LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

Environment variables define 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 the string. Although the equal sign (=) is an illegal character within an environment value, you can use it to terminate \text{value} so that trailing blanks are preserved. For example, the string \text{PATH=} = sets \text{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. These child processes get new variables and/or values added by SETENVQQ.

SETENVQQ uses the C runtime routine _putenv and GETENVQQ uses the C runtime routine getenv. From the C documentation:

- getenv and _putenv use the copy of the environment pointed to by the global variable _environ to access the environment. getenv operates only on the data structures accessible to the run-time library and not on the environment segment created for the process by the operating system.

SETENVQQ and GETENVQQ will not work properly with the Windows* APIs SetEnvironmentVariable and GetEnvironmentVariable.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “GETENVQQ”, “RUNQQ”

**Example**

```fortran
USE IFPORT
LOGICAL(4) success
success = SETENVQQ("PATH=c:\mydir\tmp")
success = &
SETENVQQ("LIB=c:\mylib\bessel.lib;c:\math\difq.lib")
END
```

SETERRORMODEQQ

**Portability Subroutine:** Sets the prompt mode for critical errors that by default generate system prompts.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL SETERRORMODEQQ (pmode)
```

**pmode**

(Input) LOGICAL(4). Flag that 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 the system starts up, system error prompting is enabled by default ($pmode = .TRUE.$). You can also enable system error prompts by calling SETERRORMODEQQ with $pmode$ set to ERR$HARDPROMPT$ (defined in IFPORT.F90).

If prompt mode is turned off, critical errors that normally cause a system 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 over what happens when an error occurs. For example, you can use the ERR= specifier to designate an executable statement to branch to for error handling. You can also take a different action than that requested by the system prompt, such as opening a temporary file, giving a more informative error message, or exiting.

You can turn off prompt mode by setting $pmode$ to .FALSE. or to the constant ERR$HARDFAIL$ (defined in IFPORT.F90).

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'.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL LIB

**Example**

```fortran
! PROGRAM 1
! DRIVE B door open
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

! PROGRAM 2
! DRIVE B door open
USE IFPORT
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
```
SETEXITQQ

**QuickWin Function:** Sets a QuickWin application’s exit behavior. This function is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = SETEXITQQ (exitmode)
```

`exitmode`

(Input) INTEGER(4). Determines the program exit behavior. The following exit parameters are defined in IFQWIN.F90:

- **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.

**Results:**

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.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** "GETEXITQQ", "Using QuickWin" in Building Applications

**Example**

```fortran
! Build as QuickWin Ap
USE IFQWIN
INTEGER(4) exmode, result
WRITE(*,'(1X,A,/)') 'Please enter the exit mode 1, 2 or 3 '
READ(*,*) exmode
SELECT CASE (exmode)
  CASE (1)
    result = SETEXITQQ(QWIN$EXITPROMPT)
  CASE (2)
```

2-399
SETFILEACCESSQQ

Portability Function: Sets the file access mode for a specified file.

Module: USE IFPORT

Syntax

\[ \text{result} = \text{SETFILEACCESSQQ}(\text{filename}, \text{access}) \]

filename

(Input) Character*(*) . Name of a 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 IFPORT.F90.

Results:

The result type is 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "GETFILEINFOQQ"

**Example**

```fortran
USE IFPORT
INTEGER(4) permit
LOGICAL(4) result

permit = 0  ! clear permit
permit = IOR(FILE$READONLY, FILE$HIDDEN)
result = SETFILEACCESSQQ ('formula.f90', permit)
END
```

**SETFILETIMEQQ**

**Portability Function:** Sets the modification time for a specified file.

**Module:** USE IFPORT

**Syntax**

```
result = SETFILETIMEQQ (filename, timedate)
```

- **filename**

- **timedate**
  (Input) INTEGER(4) . Time and date information, as packed by PACKTIMEQQ.

**Results:**

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

The modification time is the time the file was last modified and is useful for keeping track of different versions of the file. The process that calls SETFILETIMEQQ must have write access to the file; otherwise, the time cannot be changed. If you set timedate to FILE$CURTIME (defined in IFPORT.F90), SETFILETIMEQQ sets the modification time to the current system time.

If the function fails, call GETLASTERRORQQ to determine the reason. It can be one of the following:

- **ERR$ACCES** – Permission denied. The file’s (or directory’s) permission setting does not allow the specified access.
- **ERR$INVAL** – Invalid argument; the timedate argument is invalid.
• ERR$MFILE – Too many open files (the file must be opened to change its modification time).
• ERR$NOENT – File or path 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.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “PACKTIMEQQ”, “UNPACKTIMEQQ”, “GETLASTERRORQQ”

Example

USE IFPORT
INTEGER(2) day, month, year
INTEGER(2) hour, minute, second, hund
INTEGER(4) timedate
LOGICAL(4) result

CALL GETDAT(year, month, day)
CALL GETTIM(hour, minute, second, hund)
CALL PACKTIMEQQ (timedate, year, month, day, &
hour, minute, second)
result = SETFILETIMEQQ('myfile.dat', timedate)
END

SETFILLMASK

Graphics Subroutine: Sets the current fill mask to a new pattern. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL SETFILLMASK (mask)

mask

(Input) INTEGER(1). One-dimensional array of length 8.

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.

<table>
<thead>
<tr>
<th>Bit pattern</th>
<th>Value In mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>● ● ● ● ● ● ● ●</td>
<td>mask(1) = Z'93'</td>
</tr>
<tr>
<td>● ● ● ● ● ● ● ●</td>
<td>mask(2) = Z'C9'</td>
</tr>
<tr>
<td>○ ● ● ● ● ● ● ●</td>
<td>mask(3) = Z'64'</td>
</tr>
<tr>
<td>● ● ● ● ● ● ● ○</td>
<td>mask(4) = Z'B2'</td>
</tr>
<tr>
<td>○ ● ● ● ● ● ● ●</td>
<td>mask(5) = Z'59'</td>
</tr>
<tr>
<td>○ ● ● ● ● ● ● ●</td>
<td>mask(6) = Z'2C'</td>
</tr>
<tr>
<td>● ● ● ● ● ● ● ●</td>
<td>mask(7) = Z'96'</td>
</tr>
<tr>
<td>○ ● ● ● ● ● ● ●</td>
<td>mask(8) = Z'4B'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “ELLIPSE, ELLIPSE_W”, “FLOODFILLRGB, FLOODFILLRGB_W”, “GETFILLMASK”, “PIE, PIE_W”, “POLYGON, POLYGON_W”, “RECTANGLE, RECTANGLE_W”

Example

This program draws six rectangles, each with a different fill mask, as shown below.

USE IFQWIN

INTEGER(1), TARGET :: style1(8) &
    /Z'18',Z'18',Z'18',Z'18',Z'18',Z'18',Z'18'/
INTEGER(1), TARGET :: style2(8) &
    /Z'08',Z'08',Z'08',Z'08',Z'08',Z'08',Z'08'/
INTEGER(1), TARGET :: style3(8) &
The following shows the output of this program.
SETFONT

**Graphics Function:** Finds a single font that matches a specified set of characteristics and makes it the current font used by the OUTGTEXT function. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```plaintext
result = SETFONT (options)
```

**options**

(Input) Character*(*) String describing font characteristics (see below for details).

**Results:**

The result type is INTEGER(2). The result is the index number (x as used in the nx option) of the font if successful; otherwise, –1.

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 neither case sensitive nor position sensitive.

- **t 'fontname'** Name of the desired typeface. It can be any installed font.
- **h y** Character height, where y is the number of pixels.
- **w x** 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).

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.
You can specify as many options as you want, except with \texttt{n\_x}, which should be used alone. If you specify options that are mutually exclusive (such as the pairs \texttt{f/p} or \texttt{r/v}), the \texttt{SETFONT} function ignores them. There is no error detection for incompatible parameters used with \texttt{n\_x}.

If the \texttt{b} option is specified and at least one font is initialized, \texttt{SETFONT} sets a font and returns 0 to indicate success.

In selecting a font, the \texttt{SETFONT} routine uses the following criteria, rated from highest precedence to lowest:

1. Pixel height
2. Typeface
3. Pixel width
4. 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 \texttt{b} option, \texttt{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, \texttt{SETFONT} selects Arial 10.

If you choose a nonexistent value for pixel height and width, the \texttt{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 \texttt{x} option (raster-mapped font), or if you request a specific typeface and do not specify the \texttt{b} option (best-fit).

If you specify the \texttt{n\_x} parameter, \texttt{SETFONT} ignores any other specified options and supplies only the font number corresponding to \texttt{x}.

If a height is given, but not a width, \texttt{SETFONT} computes the a width to preserve the correct font proportions.
If a width is given, but not a height, SETFONT uses a default height, which may vary from font type to font type. This may lead to characters that appear distorted, particularly when a very wide width is specified. This behavior is the same as that of the Windows® API CreateFontIndirect. The second example below shows how to calculate the correct height for a given width.

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.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETFONTINFO”, “GETGTEXTTEXTENT”, “GRSTATUS”, “OUTGTEXT”, “INITIALIZEFONTS”, “SETGTEXTROTATION”

Example
! Build as a Graphics ap.
USE IFQWIN
INTEGER(2) fontnum, numfonts
TYPE (xycoord) pos
numfonts = INITIALIZEFONTS ( )
! Set typeface to Arial, character height to 18,
! character width to 10, and italic
fontnum = SETFONT ('t''Arial''h18w10i')
CALL MOVETO (INT2(10), INT2(30), pos)
CALL OUTGTEXT('Demo text')
END

Another example follows:
! The following program shows you how to compute
! an appropriate font height for a given font width
!
! Build as a Graphics ap.
USE IFQWIN
INTEGER(2) fontnum, numfonts
TYPE (xycoord) pos
TYPE (rccoord) rcc
TYPE (FONTINFO) info
CHARACTER*11 str, str1
CHARACTER*22 str2
real rh
integer h, inw
str = "t'Arial'bih"
str1= " "
numfonts = INITIALIZEFONTS ( )
! Default both height and width. This seems to work
! properly. From this setting get the ratio between
! height and width.
fontnum = SETFONT ("t'Arial'")
ireturn = GETFONTINFO(info)
rh = real(info%pixheight)/real(info%avgwidth)

! Now calculate the height for a width of 40
write(*,*) 'Input desired width:'
read(*,*) inw
h =int(inw*rh)
write(str1,'(I3.3)') h
str2 = str//str1
print *,str2
fontnum = SETFONT (str2)
CALL MOVETO (INT2(10), INT2(50), pos)
CALL OUTGTEXT('ABCDEFGabcdefg12345!@#$%')
CALL MOVETO (INT2(10), INT2(50+10+h), pos)
CALL OUTGTEXT('123456789012345678901234')
ireturn = GETFONTINFO(info)
call settextposition(4,1, rcc)
print *, info%avgwidth, info%pixheight
END

SETGTEXTROTATION

Graphics Subroutine: Sets the orientation angle of the font text output in degrees. The current
orientation is used in calls to OUTGTEXT. This subroutine is only available on Windows*
systems.

Module: USE IFQWIN

Syntax

CALL SETGTEXTROTATION (degree-tenths)
degree-tenths
(Input) INTEGER(4). Angle of orientation, in tenths of degrees, of the font text output.

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.

Bitmap fonts cannot be rotated; TruType fonts should be used instead.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETGTEXTROTATION"

Example
! Build as a Graphics ap.
USE IFQWIN
INTEGER(2) fontnum, numfonts
INTEGER(4) oldcolor, deg
TYPE (xycoord) pos
numfonts = INITIALIZEFONTS ( )
fontnum = SETFONT ('t''Arial''h18w10i')
CALL MOVETO (INT2(10), INT2(30), pos)
CALL OUTGTEXT('Straight text')
deg = -1370
CALL SETGTEXTROTATION(deg)
oldcolor = SETCOLORRGB(Z'008080')
CALL OUTGTEXT('Slanted text')
END

SETLINESTYLE

Graphics Subroutine: Sets the current line style to a new line style. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax
CALL SETLINESTYLE (mask)
mask
(Input) INTEGER(2). Desired Quickwin line-style mask. (See the table below.)

The mask is mapped to the style that most closely equivalences 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>0xEC00</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 SETCOLORRGB or SETCOLOR. SETWRITEMODE
affects how the line is displayed.

Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: "GETLINESTYLE", "GRSTATUS", "LINETO_LINETO_W", "POLYGON_POLYGON_W", "RECTANGLE_RECTANGLE_W", "SETCOLOR", "SETWRITEMODE"

Example
! Build as a Graphics ap.
USE IFQWIN
INTEGER(2)    status, style
TYPE (xycoord) xy
style = Z'FFFF'
CALL SETLINESTYLE(style)
CALL MOVETO(INT2(50), INT2(50), xy )
status = LINETO(INT2(300), INT2(300))
END
SETMESSAGEQQ

QuickWin Subroutine: Changes QuickWin status messages, state messages, and dialog box messages. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL SETMESSAGEQQ (msg, id)

msg
(Input) Character*(*)(). 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;Exit Window?&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_FINISHED</td>
<td>&quot;Finished&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PAUSED</td>
<td>&quot;Paused&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_RUNNING</td>
<td>&quot;Running&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_FILEOPENDLG</td>
<td>&quot;Text Files(<em>.txt), <em>.txt; Data Files(</em>.dat), <em>.dat; All Files(</em>.</em>), <em>.</em>;&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_BMPSAVEDLG</td>
<td>&quot;Bitmap Files(<em>.bmp), <em>.bmp; All Files(</em>.</em>), <em>.</em>;&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_INPUTPEND</td>
<td>&quot;Input pending in&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PASTEINPUTPEND</td>
<td>&quot;Paste input pending&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_MOUSEINPUTPEND</td>
<td>&quot;Mouse input pending in&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_SELECTTEXT</td>
<td>&quot;Select Text in&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_SELECTGRAPHICS</td>
<td>&quot;Select Graphics in&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTABORT</td>
<td>&quot;Error! Printing Aborted.&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTLOAD</td>
<td>&quot;Error loading printer driver&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTNODEFAULT</td>
<td>&quot;No Default Printer.&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTRIVER</td>
<td>&quot;No Printer Driver.&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTERROR</td>
<td>&quot;Print: Printing Error.&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTING</td>
<td>&quot;Printing&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_PRINTCANCEL</td>
<td>&quot;Cancel&quot;</td>
</tr>
</tbody>
</table>
QWIN$MSG_FILEOPENDLG and QWIN$MSG_BMPSAVEDLG control the text in file choosing dialog boxes and have the following syntax:

"file description, file designation"

You can change any string produced by QuickWin by calling SETMESSAGEQQ with the appropriate 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. (For more information on multibyte characters, see "Using National Language Support Routines" in Building Applications.) To change menu messages, use MODIFYMENUSTRINGQQ.

Compatibility
QUICKWIN GRAPHICS LIB

See Also: “MODIFYMENUSTRINGQQ”

Example
USE IFQWIN
print*, "Hello"
CALL SETMESSAGEQQ('Changed exit text', QWIN$MSG_EXITQ)

SETMOUSECURSOR

Quickwin Function: Sets the shape of the mouse cursor for the window in focus. This function is only available on Windows* systems.

Modules: USE IFQWIN, USE IFWIN

Syntax

\[
\text{oldcursor} = \text{SETMOUSECURSOR} ( \text{newcursor} )
\]

newcursor

(Input) INTEGER(4). A Windows HCURSOR value. For many predefined shapes, LoadCursor(0, shape) is a convenient way to get a legitimate value. See the list of predefined shapes in the table shown below in Results.

A value of zero causes the cursor not to be displayed.
Results:
The result type is INTEGER(4). This is also an HCURSOR Value. The result is the previous cursor value.

The window in focus at the time SETMOUSECURSOR is called has its cursor changed to the specified value. Once changed, the cursor retains its shape until another call to SETMOUSECURSOR.

In Standard Graphics applications, units 5 and 6 (the default screen input and output units) are always considered to be in focus.

The following predefined values for cursor shapes are available:

<table>
<thead>
<tr>
<th>Predefined Value</th>
<th>Cursor Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDC_APPSTARTING</td>
<td>Standard arrow and small hourglass</td>
</tr>
<tr>
<td>IDC_ARROW</td>
<td>Standard arrow</td>
</tr>
<tr>
<td>IDC_CROSS</td>
<td>Crosshair</td>
</tr>
<tr>
<td>IDC_IBeam</td>
<td>Text I-beam</td>
</tr>
<tr>
<td>IDC_ICON</td>
<td>Obsolete value</td>
</tr>
<tr>
<td>IDC_NO</td>
<td>Slashed circle</td>
</tr>
<tr>
<td>IDC_SIZE</td>
<td>Obsolete value; use IDC_SIZEALL</td>
</tr>
<tr>
<td>IDC_SIZEALL</td>
<td>Four-pointed arrow</td>
</tr>
<tr>
<td>IDC_SIZENESW</td>
<td>Double-pointed arrow pointing northeast and southwest</td>
</tr>
<tr>
<td>IDC_SIZENS</td>
<td>Double-pointed arrow pointing north and south</td>
</tr>
<tr>
<td>IDC_SIZEWSE</td>
<td>Double-pointed arrow pointing northwest and southeast</td>
</tr>
<tr>
<td>IDC_SIZEWE</td>
<td>Double-pointed arrow pointing west and east</td>
</tr>
<tr>
<td>IDC_UPARROW</td>
<td>Vertical arrow</td>
</tr>
<tr>
<td>IDC_WAIT</td>
<td>Hour glass</td>
</tr>
</tbody>
</table>

A LoadCursor must be done on these values before they can be used by SETMOUSECURSOR.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

Example:
! Build as Standard Graphics or QuickWin
use IFQWIN

    integer*4  cursor, oldcursor
    write(6,*') 'The cursor will now be changed to an hour glass shape'
write(6,*) 'Hit <return> to see the next change'
cursor = LoadCursor(0, IDC_WAIT)
oldcursor = SetMouseCursor(cursor)
read(5,*)

write(6,*) 'The cursor will now be changed to a cross-hair shape'
write(6,*) 'Hit <return> to see the next change'
cursor = LoadCursor(0, IDC_CROSS)
oldcursor = SetMouseCursor(cursor)
read(5,*)

write(6,*) 'The cursor will now be turned off'
write(6,*) 'Hit <return> to see the next change'
oldcursor = SetMouseCursor(0)
read(5,*)

write(6,*) 'The cursor will now be turned on'
write(6,*) 'Hit <return> to see the next change'
oldcursor = SetMouseCursor(oldcursor)
read(5,*)

stop
end

SETPIXEL, SETPIXEL_W

Graphics Functions: Set a pixel at a specified location to the current graphics color index. These functions are only available on Windows® systems.

Module: USE IFQWIN

Syntax

result = SETPIXEL (x, y)
result = SETPIXEL_W (wx, wy)

x, y
(Input) INTEGER(2). Viewport coordinates for target pixel.

wx, wy
(Input) REAL(8). Window coordinates for target pixel.
Results:
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 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.

If you use color indexes, you are restricted 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. SETPIXELRGB and SETPIXELRGB_W give access to the full color capacity of the system by using direct color values rather than indexes to a palette.

NOTE. The SETPIXEL routine described here is a QuickWin routine. If you are trying to use the Microsoft* Platform SDK version of the SetPixel routine by including the IFWIN module, you need to specify the routine name as MSFWIN$SetPixel. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: “SETPIXELRGB, SETPIXELRGB_W”, “GETPIXEL, GETPIXEL_W”, “SETPIXELS”, “GETPIXELS”, “GETCOLOR”, “SETCOLOR”

Example

! Build as a Graphics ap.
USE IFQWIN
INTEGER(2) status, x, y
status = SETCOLOR(INT2(2))
x = 10
! Draw pixels.
DO y = 50, 389, 3
   status = SETPIXEL( x, y )
   x = x + 2
END DO
READ (*,*) ! Wait for ENTER to be pressed
END
SETPIXELRGB, SETPIXELRGB_W

**Graphics Functions:** Set a pixel at a specified location to the specified Red-Green-Blue (RGB) color value. These functions are only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
result = SETPIXELRGB (x, y, color)
result = SETPIXELRGB_W (wx, wy, color)
```

- **x, y**
  
  (Input) INTEGER(2). Viewport coordinates for target pixel.

- **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.

**Results:**

The result type is INTEGER(4). The result is the previous RGB color value of the pixel.

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 or SETPIXELRGB_W, red is the rightmost byte, followed by green and blue. The RGB value’s internal structure is as follows:

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O O O O</td>
<td>B B B B B B B</td>
<td>G G G G G G G</td>
<td>R R R R R R R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Larger numbers correspond to stronger color intensity with binary 1111111 (hex Z'FF') the maximum for each of the three components. For example, Z'0000FF' yields full-intensity red, Z'00FF00' full-intensity green, Z'FF0000' full-intensity blue, and Z'FFFF00' full-intensity for all three, resulting in bright white.

If any of the pixels are outside the clipping region, those pixels are ignored.

SETPIXELRGB (and the other RGB color selection functions such as SETPIXELSRGB, SETCOLORRRGB) 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 restricted 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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also:  “GETPIXELRGB, GETPIXELRGB_W”, “GETPIXELSRGB”, “SETCOLORRGB”, “SETPIXELSRGB”

**Example**

! Build as a Graphics ap.
USE IFQWIN
INTEGER(2) x, y
INTEGER(4) color
DO i = 10, 30, 10
   SELECT CASE (i)
      CASE(10)
         color = Z'0000FF'
      CASE(20)
         color = Z'00FF00'
      CASE (30)
         color = Z'FF0000'
   END SELECT
! Draw pixels.
   DO y = 50, 180, 2
      status = SETPIXELRGB( x, y, color )
      x      = x + 2
   END DO
END DO
READ (*,*) ! Wait for ENTER to be pressed
END

**SETPIXELS**

**Graphics Subroutine:** Sets the color indexes of multiple pixels. This subroutine is only available on Windows® systems.

**Module:** USE IFQWIN
Syntax

    CALL SETPIXELS (n, x, y, color)

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.

SETPIXELS sets the pixels specified in the arrays x and y to the color indexes 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.

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 restricted 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.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETPIXELS", "SETPIXEL, SETPIXEL_W", "SETPIXELSRGB"

Example

    ! Build as a Graphics ap.
    USE IFQWIN
    INTEGER(2) color(9)
    INTEGER(2) x(9), y(9), i
    DO i = 1, 9
        x(i) = 20 * i
        y(i) = 10 * i
        color(i) = INT2(i)
    END DO
CALL SETPIXELS(9, x, y, color)
END

SETPIXELSRGB

**Graphics Subroutine:** Sets multiple pixels to the given Red-Green-Blue (RGB) color. This subroutine is only available on Windows® systems.

**Module:** USE IFQWIN

**Syntax**

CALL SETPIXELSRGB (n, x, y, color)

**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.

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 color values, 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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>0 0 0 0 0 0 0</td>
<td>B B B B B B B</td>
<td>G G G G G G G</td>
<td>R R R R R R R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Larger numbers correspond to stronger color intensity with binary 11111111 (hex Z’FF’) the maximum for each of the three components. For example, Z’0000FF’ yields full-intensity red, Z’00FF00’ full-intensity green, Z’FF0000’ full-intensity blue, and Z’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 restricted 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.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: “GETPIXELSRGB”, “SETPIXELRGB, SETPIXELRGB_W”, “GETPIXELRGB,
GETPIXELRGB_W”, “SETPIXELS”

Example

! Buffering replacement for SetPixelRGB and
! SetPixelRGB_W. This can improve performance by
! doing batches of pixels together.

USE IFQWIN
PARAMETER (I$SIZE = 200)
INTEGER(4) bn, bc(I$SIZE), status
INTEGER(2) bx(I$SIZE), by(I$SIZE)

bn = 0
DO i = 1, I$SIZE
    bn = bn + 1
    bx(bn) = i
    by(bn) = i
    bc(bn) = GETCOLORRGB()
    status = SETCOLORRGB(bc(bn)+1)
END DO
CALL SETPIXELSRGB(bn, bx, by, bc)
END
### SETTEXTCOLOR

**Graphics Function:** Sets the current text color index. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```plaintext
result = SETTEXTCOLOR (index)
```

- `index` (Input) INTEGER(2). Color index to set the text color to.

**Results:**

The result type is INTEGER(2). The result is the previous text color index.

**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.

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.

---

**NOTE.** The SETTEXTCOLOR routine described here is a QuickWin routine. If you are trying to use the Microsoft* Platform SDK version of the SetTextColor routine by including the IFWIN module, you need to specify the routine name as MSFWIN$SetTextColor. For more information, see "Special Naming Convention for Certain QuickWin and Win32 Graphics Routines" in Building Applications.

---

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “GETTEXTCOLOR”, “REMAPPALETTERGB”, “SETCOLOR”, “SETTEXTCOLORRGB”

**Example**

```plaintext
! Build as a Graphics ap.
USE IFQWIN
```
INTEGER(2) oldtc
oldtc = SETTEXTCOLOR(INT2(2)) ! green
WRITE(*,*) "hello, world"
END

SETTEXTCOLORRGB

Graphics Function: Sets the current text color to the specified Red-Green-Blue (RGB) value. This function is only available on Windows® systems.

Module: USE IFQWIN

Syntax

result = SETTEXTCOLORRGB (color)

color

(Input) INTEGER(4). RGB color value to set the text color to. Range and result depend on the system’s display adapter.

Results:
The result type is INTEGER(4). The result is the previous text RGB color value.

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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>31 (MSB)</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>O O O O O</td>
<td>O O</td>
<td>B B</td>
<td>B B</td>
<td>B B</td>
<td>G G</td>
<td>G G</td>
<td>G G</td>
</tr>
</tbody>
</table>

Larger numbers correspond to stronger color intensity with binary 1111111 (hex Z’FF’) the maximum for each of the three components. For example, Z’0000FF’ yields full-intensity red, Z’00FF00’ full-intensity green, Z’FF0000’ full-intensity blue, and Z’FFFFFF’ full-intensity for all three, resulting in bright white.

SETTEXTCOLORRGB sets the current text RGB color. The default value is Z’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 restricted 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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** "SETBKCOLORRGB", "SETCOLORRGB", "GETTEXTCOLORRGB", "GETWINDOWCONFIG", "OUTTEXT"

**Example**

! Build as a Graphics ap.
USE IFQWIN
INTEGER(4) oldtc

oldtc = SETTEXTCOLORRGB(Z'000000FF')
WRITE(*,-) 'I am red'
oldtc = SETTEXTCOLORRGB(Z'0000FF00')
CALL OUTTEXT ('I am green'//CHAR(13)//CHAR(10))
oldtc = SETTEXTCOLORRGB(Z'00FF0000')
PRINT *, 'I am blue'
END

---

**SETTEXTCURSOR**

**Graphics Function:** Sets the height and width of the text cursor (the caret) for the window in focus. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

result = SETTEXTCURSOR (newcursor)

**newcursor**

(Input) INTEGER(2). The leftmost 8 bits specify the width of the cursor, and the rightmost 8 bits specify the height of the cursor. These dimensions can range from 1 to 8, and represent a fraction of the current character cell size. For example:

- Z'0808' – Specifies the full character cell; this is the default size.
- Z'0108' – Specifies 1/8th of the character cell width, and 8/8th (or all) of the character cell height.

If either of these dimensions is outside the range 1 to 8, it is forced to 8.

**Results:**

The result type is INTEGER(2); it is the previous text cursor value in the same format as `newcursor`.

---

**NOTE.** After calling `SETTEXTCURSOR`, you must call `DISPLAYCURSOR($GCURSORON)` to actually see the cursor.

---

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “DISPLAYCURSOR”

**Example**

```
use IFQWIN
integer(2) oldcur
integer(2) istat
type(rccoord) rc
open(10, file='user')
istat  = displaycursor($GCURSORON)
write(10,*) 'Text cursor is now character cell size, the default.'
read(10,*)
write(10,*) 'Setting text cursor to wide and low.'
oldcur = settextcursor(Z'0801')
istat  = displaycursor($GCURSORON)
read(10,*)
write(10,*) 'Setting text cursor to high and narrow.'
oldcur = settextcursor(Z'0108')
istat  = displaycursor($GCURSORON)
read(10,*)
write(10,*) 'Setting text cursor to a dot.'
oldcur = settextcursor(Z'0101')
istat  = displaycursor($GCURSORON)
read(10,*)
end
```
**SETTEXTPOSITION**

**Graphics Subroutine:** Sets the current text position to a specified position relative to the current text window. This subroutine is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```fortran
CALL SETTEXTPOSITION (row, column, t)
```

- **row** (Input) INTEGER(2). New text row position.
- **column** (Input) INTEGER(2). New text column position.
- **t** (Output) Derived type rccoord. Previous text position. The derived type rccoord is defined in IFQWIN.F90 as follows:

```fortran
TYPE rccoord
  INTEGER(2) row ! Row coordinate
  INTEGER(2) col ! Column coordinate
END TYPE rccoord
```

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, column).

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** "CLEARSCREEN", "GETTEXTPOSITION", "OUTTEXT", "SCROLLTEXTWINDOW", "SETTEXTWINDOW", "WRAPON"

**Example**

```fortran
USE IFQWIN
TYPE (rccoord) curpos
WRITE(*,*) "Original text position"
CALL SETTEXTPOSITION (INT2(6), INT2(5), curpos)
WRITE (*,*) 'New text position'
END
```
SETTEXTWINDOW

Graphics Subroutine: Sets the current text window. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

    CALL SETTEXTWINDOW (r1, c1, r2, c2)

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.

SETTEXTWINDOW 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.

SETTEXTWINDOW does not affect the output of the graphics text routine OUTGTEXT. Use the SETVIEWPORT function to control the display area for graphics output.

Compatibility

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "GETTEXTPOSITION", "GETTEXTWINDOW", "GRSTATUS", "OUTTEXT", "SCROLLTEXTWINDOW", "SETTEXTPOSITION", "SETVIEWPORT", "WRAPON"

Example

USE IFQWIN
TYPE (rccoord) curpos

CALL SETTEXTWINDOW(INT2(5), INT2(1), INT2(7), & INT2(40))
CALL SETTEXTPOSITION (INT2(5), INT2(5), curpos)
WRITE(*,*) "Only two lines in this text window"
WRITE(*,*) "so this line will be overwritten"
WRITE(*,*) "by this line"
END
SETTIM

Portability Function: Sets the system time in your programs.

Module: USE IFPORT

Syntax

result = SETTIM(ihr, imin, isec, i100th)

ihr
(Input) INTEGER(4) or INTEGER(2). Hour (0-23).

imin
(Input) INTEGER(4) or INTEGER(2). Minute (0-59).

isec
(Input) INTEGER(4) or INTEGER(2). Second (0-59).

i100th
(Input) INTEGER(4) or INTEGER(2). Hundredth of a second (0-99).

Results:
The result type is LOGICAL(4). The result is .TRUE. if the system time is changed; .FALSE. if no change is made.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “GETDAT”, “GETTIM”, “SETDAT”

Example
USE IFPORT
LOGICAL(4) success
success = SETTIM(INT2(21), INT2(53+3),
                 INT2(14*2), INT2(88))

SETVIEWORG

Graphics Subroutine: Moves the viewport-coordinate origin (0, 0) to the specified physical point. This subroutine is only available on Windows* systems.

Module: USE IFQWIN

Syntax

CALL SETVIEWORG(x, y, t)
Input) INTEGER(2). Physical coordinates of new viewport origin.

\( t \)

(Output) Derived type \texttt{xycoord}. Physical coordinates of the previous viewport origin. The derived type \texttt{xycoord} is defined in \texttt{IFQWIN.F90} as follows:

\begin{verbatim}
TYPE xycoord
  INTEGER(2) xcoord ! x-coordinate
  INTEGER(2) ycoord ! y-coordinate
END TYPE xycoord
\end{verbatim}

The \texttt{xycoord} type variable \( t \), defined in \texttt{IFQWIN.F90}, returns the physical coordinates of the previous viewport origin.

Compatibility

\texttt{STANDARD GRAPHICS QUICKWIN GRAPHICS LIB}

See Also: “\texttt{GETCURRENTPOSITION, GETCURRENTPOSITION_W, GETPHYSCOORD}”, “\texttt{GETVIEWCOORD, GETVIEWCOORD_W, GETWINDOWCOORD}”, “\texttt{GRSTATUS}”, “\texttt{SETCLIPRGN}”, “\texttt{SETVIEWPORT}”

Example

\begin{verbatim}
USE IFQWIN
TYPE (xycoord) xy
CALL SETVIEWORG(INT2(30), INT2(30), xy)
\end{verbatim}

\section*{SETVIEWPORT}

\textbf{Graphics Subroutine:} Redefines the graphics viewport by defining a clipping region in the same manner as \texttt{SETCLIPRGN} and then setting the viewport-coordinate origin to the upper-left corner of the region. This subroutine is only available on Windows\textsuperscript{*} systems.

\textbf{Module:} \texttt{USE IFQWIN}

\textbf{Syntax}

\begin{verbatim}
CALL SETVIEWPORT (x1, y1, x2, y2)
\end{verbatim}

\( x1, y1 \)

(Input) INTEGER(2). Physical coordinates for upper-left corner of viewport.

\( x2, y2 \)

(Input) INTEGER(2). Physical coordinates for lower-right corner of viewport.
The physical coordinates \((x_1, y_1)\) and \((x_2, y_2)\) 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.

**Compatibility**

STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

**See Also:** “GETVIEWCOORD, GETVIEWCOORD_W”, “GETPHYSCOORD”, “GRSTATUS”, “SETCLIPRGN”, “SETVIEWORG”, “SETWINDOW”

**Example**

USE IFQWIN

INTEGER(2) upx, upy
INTEGER(2) downx, downy

upx = 0
upy = 30
downx = 250
downy = 100

CALL SETVIEWPORT(upx, upy, downx, downy)

**SETWINDOW**

**Graphics Function:** Defines a window bound by the specified coordinates. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

result = SETWINDOW (finvert, wx1, wy1, wx2, wy2)

**finvert**

(Input) LOGICAL(2). Direction of increase of the y-axis. If \(\text{finvert}\) is .TRUE., the y-axis increases from the window bottom to the window top (as Cartesian coordinates). If \(\text{finvert}\) 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.
Results:
The result type is INTEGER(2). The result is nonzero if successful; otherwise, 0 (for example, if
the program that calls SETWINDOW is not in a graphics mode).

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 \(wx1\) equals \(wx2\) or \(wy1\) equals \(wy2\), SETWINDOW fails.

Compatibility
STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

See Also: “GETWINDOWCOORD”, “SETCLIPRGN”, “SETVIEWORG”, “SETVIEWPORT”,
“GRSTATUS”

Example
USE IFQWIN
INTEGER(2) status
LOGICAL(2) invert /.TRUE./
REAL(8) upx /0.0/, upy /0.0/
REAL(8) downx /1000.0/, downy /1000.0/
status = SETWINDOW(invert, upx, upy, downx, downy)

SETWINDOWCONFIG

QuickWin Function: Sets the properties of a child window. This function is only available on
Windows® systems.

Module: USE IFQWIN

Syntax
   result = SETWINDOWCONFIG (wc)

wc
   (Input) Derived type windowconfig. Contains window properties. The windowconfig
derived type is defined in IFQWIN.F90 as follows:
TYPE 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$EXTENDFONT when specifying
                      ! extended attributes, in which
                      ! case extendfontsize sets the
                      ! font size.
  CHARACTER(80) title ! The window title.
  INTEGER(2) bitsperpixel ! The number of bits per pixel.
  INTEGER(2) numvideopages ! Unused.
  INTEGER(2) mode ! Controls scrolling mode (see
                   ! wc%mode below).
  INTEGER(2) adapter ! Unused.
  INTEGER(2) monitor ! Unused.
  INTEGER(2) memory ! Unused.
  INTEGER(2) environment ! Unused.
  ! The next three parameters provide extended font
  ! attributes.
  CHARACTER(32) extendfontname ! The name of the desired font.
  INTEGER(4) extendfontsize ! Takes the same values as fontsize,
                           ! when fontsize is set to
                           ! QWIN$EXTENDFONT.
  INTEGER(4) extendfontattributes ! Font attributes such as bold
                                   ! and italic.
END TYPE windowconfig

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
The following value can be used to configure a QuickWin window so that it will show the last line
written and the text cursor (if it is on):
  wc%mode = QWIN$SCROLLDOWN
Be aware that if you scroll the window to another position, you will have to scroll back to the last
line to see your input.
The following values can be used with SETWINDOWCONFIG extended fonts:

<table>
<thead>
<tr>
<th>Style:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QWIN$EXTENDFONT_NORMAL</td>
<td>Gives no underline, no italic, and a font weight of 400 out of 1000.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_UNDERLINE</td>
<td>Gives underlined characters.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_BOLD</td>
<td>Gives a font weight of 700 out of 1000.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_ITALIC</td>
<td>Gives italic characters.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pitch:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QWIN$EXTENDFONT_FIXED_PITCH</td>
<td>QuickWin default. Equal character widths.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_VARIABLE_PITCH</td>
<td>Variable character widths.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Font Families:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QWIN$EXTENDFONT_FF_ROMAN</td>
<td>Variable stroke width, serified. Times Roman, Century Schoolbook, etc.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_FF_SWISS</td>
<td>Variable stroke width, sans-serifed. Helvetica, Swiss, etc.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_FF_MODERN</td>
<td>QuickWin default. Constant stroke width, serified or sans-serifed. Pica, Elite, Courier, etc.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_FF_SCRIPT</td>
<td>Cursive, etc.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_FF_DECORATIVE</td>
<td>Old English, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character Sets:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QWIN$EXTENDFONT_ANSI_CHARSET</td>
<td>QuickWin default.</td>
</tr>
<tr>
<td>QWIN$EXTENDFONT_OEM_CHARSET</td>
<td>Use this to get Microsoft* LineDraw.</td>
</tr>
</tbody>
</table>

Using QWIN$EXTENDFONT_OEM_CHARSET with the font name 'MS LineDraw'C will get the old DOS-style character set with symbols that can be used to draw lines and boxes. The pitch and font family items can be specified to help guide the font matching algorithms used by CreateFontIndirect, the Windows* API used by SETWINDOWCONFIG.

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:

```fortran
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.

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.

**NOTE.** You must call DISPLAYCURSOR($GCURSORON) to make the cursor visible after calling SETWINDOWCONFIG.

**Compatibility**

STANDARD GRAPHICS QUICKWIN GRAPHICS LIB

**See Also:** "GETWINDOWCONFIG", "DISPLAYCURSOR", "Using QuickWin" in *Building Applications*

**Example**

```fortran
USE IFQWIN
TYPE (windowconfig) wc
LOGICAL status /.FALSE./
! Set the x & y pixels to 800X600 and font size to 8x12
wc%numxpixels  = 800
wc%numypixels  = 600
wc%numtextcols = -1
```
wc%numtextrows = -1
wc%numcolors = -1
wc%title = 'This is a test'
wc%fontsize = Z'0008000C'
status = SETWINDOWCONFIG(wc) ! attempt to set configuration with above values
   ! if attempt fails, set with system estimated values
if (.NOT.status) status = SETWINDOWCONFIG(wc)

SETWINDOWMENUQQ

QuickWin Function: Sets a top-level menu as the menu to which a list of current child window names is appended. This function is only available on Windows® systems.

Module: USE IFQWIN

Syntax
   result = SETWINDOWMENUQQ (menuID)

menuID
   (Input) INTEGER(4). Identifies the menu to hold the child window names, starting with 1 as the leftmost menu.

Results:
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
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.

Compatibility
QUICKWIN GRAPHICS LIB

See Also: "APPENDMENUQQ", "Using QuickWin" and "Customizing QuickWin Applications" in Building Applications,

Example
USE IFQWIN
TYPE (windowconfig) wc
LOGICAL(4) result, status /.FALSE./
   ! Set title for child window
wc%numx pixels  = -1
wc%numy pixels  = -1
wc%numtextcols = -1
SETWRITEMODE

Graphics Function: Sets the current logical write mode, which is used when drawing lines with the LINETO, POLYGON, and RECTANGLE functions. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = SETWRITEMODE (wmode)

wmode

(Input) INTEGER(2). Write mode to be set. One of the following symbolic constants (defined in IFQWIN.F90):

- $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 Windows* API SetROP2):

- $GR2_BLACK
- $GR2_NOTMERGEPEN
- $GR2_MASKNOTPEN
- $GR2_NOTCOPYPEN (same as $GPRESET)
- $GR2_MASKPENNOT

wc%numtextrows = -1
wc%numcolors   = -1
wc%fontsize    = -1
wc%title = "I am child window name"C
if (.NOT.status) status = SETWINDOWCONFIG(wc)
result = SETWINDOWMENUQQ(3)
END
$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

Results:
The result type is INTEGER(2). The result is the previous write mode if successful; otherwise, -1.

The current graphics color is set with SETCOLORRGB (or SETCOLOR) and the current background color is set with SETBKCOLORRGB (or SETBKCOLOR). As an example, suppose you set the background color to yellow (Z'00FFFF') and the graphics color to purple (Z'FF00FF') with the following commands:

oldcolor = SETBKCOLORRGB(Z'00FFFF')
CALL CLEARSCREEN($GCLEARSCREEN)
oldcolor = SETCOLORRGB(Z'FF00FF')

If you then set the write mode with the $GAND option, lines are drawn in red (Z'0000FF'); with the $GOR option, lines are drawn in white (Z'FFFFFF'); with the $GXOR option, lines are drawn in turquoise (Z'FFFF00'); and with the $GPRESET option, lines are drawn in green (Z'00FF00'). Setting the write mode to $GPSET causes lines to be drawn in the graphics color.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB


Example
! Build as a Graphics ap.
USE IFQWIN
INTEGER(2) result, oldmode
INTEGER(4) oldcolor
TYPE (xycoord) xy

2 Intel Fortran Libraries Reference
oldcolor = SETBKCOLORRGB(Z'00FFFF')
CALL CLEARSCREEN ($GCLEARSCREEN)
oldcolor = SETCOLORRGB(Z'FF00FF')
CALL MOVETO(INT2(0), INT2(0), xy)
result = LINETO(INT2(200), INT2(200)) ! purple

oldmode = SETWRITEMODE( $GAND)
CALL MOVETO(INT2(50), INT2(0), xy)
result = LINETO(INT2(250), INT2(200)) ! red
END

SETWSIZEQQ

QuickWin Function: Sets the size and position of a window. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = SETWSIZEQQ (unit, winfo)

unit

(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 unit to the symbolic constant QWINSFRAMEWINDOW (defined in IFQWIN.F90).

When called from INITIALSETTINGS, SETWSIZEQQ behaves slightly differently than when called from a user routine after initialization. See below under Results.

winfo

(Input) Derived type 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). The derived type qwinfo is defined in IFQWIN.F90 as follows:

```
TYPE 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 TYPE QWINFO
```
This function’s behavior depends on the value of QWININFO%TYPE, 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 qwinfo.

**Results:**

The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero (unless called from INITIALSETTINGS). If called from INITIALSETTINGS, the following occurs:

- SETWSIZEQQ always returns –1.
- Only QWIN$SET will work.

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.

**Compatibility**

QUICKWIN GRAPHICS LIB

**See Also:** "GETWSIZEQQ", "INITIALSETTINGS", "Using QuickWin" in **Building Applications**

**Example**

```
USE IFQWIN
LOGICAL(4)     result
INTEGER(2)     numfonts, fontnum
TYPE (qwinfo)   winfo
TYPE (xycoord)  pos
! Maximize frame window
winfo%TYPE = QWIN$MAX
result =     SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
! Maximize child window
result =   SETWSIZEQQ(0, winfo)
numfonts = INITIALIZEFONTS( )
fontnum =  SETFONT ('t''Arial''h50w34i')
CALL MOVETO (INT2(10), INT2(30), pos)
CALL OUTGTEXT("BIG Window")
END
```
**SHORT**

**Portability Function:** Converts an INTEGER(4) argument to INTEGER(2) type.

**Module:** USE IFPORT

**Syntax**

\[
\text{result} = \text{SHORT} \left( \text{int4} \right)
\]

**int4**

(Input) INTEGER(4). Value to be converted.

**Results:**

The result type is INTEGER(2). The result is equal to the lower 16 bits of \text{int4}. If the \text{int4} value is greater than 32,767, the converted INTEGER(2) value is not equal to the original.

**Compatibility**

CONSOLE STANDRAD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** the INT intrinsic function and the TYPE statement in the Language Reference

**Example**

```fortran
USE IFPORT
INTEGER(4) THIS_ONE
INTEGER(2) THAT_ONE
READ(*,*) THIS_ONE
THAT_ONE = SHORT(THIS_ONE)
WRITE(*,10) THIS_ONE, THAT_ONE
END
```

**SIGNAL**

**Portability Function:** Controls interrupt signal handling. Changes the action for a specified signal.

**Module:** USE IFPORT

**Syntax**

\[
\text{result} = \text{SIGNAL} \left( \text{signum}, \text{proc}, \text{flag} \right)
\]

**signum**

(Input) INTEGER(4). Number of the signal to change. The numbers and symbolic names for the signals are listed in a table below.
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.

Results:
The result type is INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. 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 EINV AL. If the signal number supplied is invalid, or if the flag value is greater than 1, SIGNAL returns –(EINV AL) and a call to IERRNO returns EINV AL.

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. You cannot use the following kinds of signal-handler routines:

• Routines that perform low-level (such as FGETC) or high-level (such as READ) I/O.
• Heap routines or any routine that uses the heap routines (such as MALLOC and ALLOCATE).
• Functions that generate a system call (such as TIME).

The following table lists signal names and values:
The default action for all signals is to terminate the program with exit code. ABORT does not assert the SIGABRT signal. The only way to assert SIGABRT or SIGTERM is to use KILL.

SIGNAL can be used to catch SIGFPE exceptions, but it cannot be used to access the error code that caused the SIGFPE. To do this, use SIGNALQQ instead.

Compatibility

See Also: "SIGNALQQ"

Example

```
USE IFPORT
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) FUNCTION h_abort (sig_num)
INTEGER(4) sig_num

WRITE(*,*) 'In signal handler for SIG$ABORT'
```

<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(^1)</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>

\(^1\) SIGKILL can neither be caught nor ignored.


```fortran
WRITE(*,*) 'signum = ', sig_num
h_abort = 1
END
```

**SIGNALQQ**

**Portability Function:** Registers the function to be called if an interrupt signal occurs.

**Module:** USE IFPORT

**Syntax**

```fortran
result = SIGNALQQ (sig, func)
```

- **sig**
  
  (Input) INTEGER(2). Interrupt type. One of the following constants, defined in IFPORT.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

- **func**
  
  (Input) Function to be executed on interrupt. It must be declared EXTERNAL.

**Results:**

The result type is INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. The result is a positive integer if successful; otherwise, –1 (SIG$ERR).

SIGNALQQ installs the function `func` as the handler for a signal of the type specified by `sig`. If you do not install a handler, the system by default terminates the program with exit code 3 when an interrupt signal occurs.

The argument `func` 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.

**NOTE.** All signal-handler functions must be declared with the cDECS ATTRIBUTES C option.
When an interrupt occurs, except a SIG$FPE interrupt, the `sig` argument SIG$INT is passed to `func`, and then `func` is executed.

When a SIG$FPE interrupt occurs, the function `func` is passed two arguments: SIG$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 `IFPORT.F90`. Floating-point exceptions are described and discussed in "The Floating-Point Environment" in *Building Applications*.

If `func` returns, the calling process resumes execution immediately after the point at which it received the interrupt signal. This is true regardless of the type of interrupt 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, do 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 RAISEQQ, which causes your program either to branch to the signal handlers set with SIGNALQQ, or to perform the system default behavior if SIGNALQQ has set no signal handler.

The example below demonstrates a signal handler for SIG$ABORT. A sample signal handler for SIG$FPE is given in "Handling Floating-Point Exceptions" in *Building Applications*.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “RAISEQQ”, “SIGNAL”, “KILL”, “GETEXCEPTIONPTRSQO”

**Example**

```fortran
! This program shows a signal handler for SIG$ABORT
USE IFPORT
INTERFACE
FUNCTION h_abort (signum)
    !DEC$ ATTRIBUTES C :: h_abort
    INTEGER(4) h_abort
    INTEGER(2) signum
END FUNCTION
END INTERFACE

INTEGER(2) i2ret
INTEGER(4) i4ret
```
i4ret = SIGNALQQ(SIG$ABORT, h_abort)
WRITE(*,*) 'Set signal handler. Return = ', i4ret

i2ret = RAISEQQ(SIG$ABORT)
WRITE(*,*) 'Raised signal. Return = ', i2ret
END

!  Signal handler routine
!
INTEGER(4) FUNCTION h_abort (signum)
  !DEC$ ATTRIBUTES C :: h_abort
  INTEGER(2) signum
  WRITE(*,*) 'In signal handler for SIG$ABORT'
  WRITE(*,*) 'signum = ', signum
  h_abort = 1
END

SLEEP

Portability Subroutine: Suspends the execution of a process for a specified interval.
Module: USE IFPORT
Syntax
  CALL SLEEP (time)

  time
  (Input) INTEGER(4). Length of time, in seconds, to suspend the calling process.
Compatibility
  CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: "SLEEPQQ"
Example
USE IFPORT
integer(4) hold_time
hold_time = 1 !lets the loop execute
DO WHILE (hold_time .NE. 0)
  write(*,'(A)') "Enter the number of seconds to suspend"
  read(*,*) hold_time
  CALL SLEEP (hold_time)
SLEEPQQ

**Portability Subroutine:** Delays execution of the program for a specified duration.

**Module:** USE IFPORT

**Syntax**

CALL SLEEPQQ (duration)

**duration**

(Input) INTEGER(4). Number of milliseconds the program is to sleep (delay program execution).

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**Example**

USE IFPORT

INTEGER(4) delay, freq, duration

delay = 2000
freq = 4000
duration = 1000
CALL SLEEPQQ(delay)
CALL BEEPQQ(freq, duration)
END

SORTQQ

**Portability Subroutine:** Sorts a one-dimensional array. The array elements cannot be derived types or record structures.

**Module:** USE IFPORT

**Syntax**

CALL SORTQQ (adrarray, count, size)

**adrarray**

(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. Address of the array (returned by LOC).
count
(Input; output) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. 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 IFPORT.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$INTEGER4</td>
<td>INTEGER(4) or equivalent</td>
</tr>
<tr>
<td>SRT$INTEGER8</td>
<td>INTEGER(8) 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>
<tr>
<td>SRT$REAL16</td>
<td>REAL(16) 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 intrinsic 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.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “BSEARCHQQ”, the LOC intrinsic function in the Language Reference
Example
!
Sort a 1-D array
!
USE IFPORT
INTEGER(2) array(10)
INTEGER(2) i
DATA ARRAY /143, 99, 612, 61, 712, 9112, 6, 555, 2223, 67/
!
Sort the array
Call SORTQQ (LOC(array)), 10, SRT$INTEGER2)
!
Display the sorted array
DO i = 1, 10
    WRITE (*, 9000) i, array (i)
9000 FORMAT(1X, ' Array(',I2, '): ', I5)
END DO
END

SPLITPATHQQ

Portability Function: Breaks a file path or directory path into its components.
Module: USE IFPORT
Syntax
    result = SPLITPATHQQ (path, drive, dir, name, ext)
path
(Input) Character*(*) . Path to be broken into components. Forward slashes (/), backslashes (\), or both can be present in path.
drive
(Output) Character*(*) . Drive letter followed by a colon.
dir
(Output) Character*(*) . Path of directories, including the trailing slash.
nname
(Output) Character*(*) . Name of file or, if no file is specified in path, name of the lowest directory. A file name must not include an extension.
ext
(Output) Character*(*) . File name extension, if any, including the leading period (.).
Results:
The result type is INTEGER(4). The result is the length of dir.
The path parameter can be a complete or partial file specification.
$MAXPATH is a symbolic constant defined in module IFPORT.F90 as 260.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “FULLPATHQQ”

Example
USE IFPORT
CHARACTER($MAXPATH) buf
CHARACTER(3) drive
CHARACTER(256) dir
CHARACTER(256) name
CHARACTER(256) ext
CHARACTER(256) file
INTEGER(4) length

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

SPORT_CANCEL_IO

Serial Port I/O Function: Cancels any I/O in progress to the specified port. This function is only available on Windows® systems.
Module: USE IFPORT

Syntax
result = SPORT_CANCEL_IO (port)

port
(Input) Integer. The port number.
Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

NOTE. This call also kills the thread that keeps an outstanding read operation to the serial port. This call must be done before any of the port characteristics are modified.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB "

See Also: "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft® Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_CANCEL_IO( 2 )
END

SPORT_CONNECT

Serial Port I/O Function: Establishes the connection to a serial port and defines certain usage parameters. This function is only available on Windows® systems.

Module: USE IFPORT

Syntax
result = SPORT_CONNECT (port [, options])

port
(Input) Integer. The port number of connection. The routine will open COMn, where n is the port number specified.

options
(Optional; input) Integer. Defines the connection options. These options define how the nnn_LINE routines will work and also effect the data that is passed to the user. If more than one option is specified, the operator .OR. should be used between each option. Options are as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_TOSS_CR</td>
<td>Removes carriage return (CR) characters on input.</td>
</tr>
</tbody>
</table>
If `options` is not specified, the following occurs by default:

\[(\text{DL\_OUT\_CR} \ \text{OR} \ \text{DL\_TERM\_CR} \ \text{OR} \ \text{DL\_TOSS\_CR} \ \text{OR} \ \text{DL\_TOSS\_LF})\]

This specifies to remove carriage returns and linefeeds on input, to follow output lines with a carriage return, and to return input lines when a carriage return is encountered.

**Results:**

The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

**Compatibility**

<table>
<thead>
<tr>
<th>Console</th>
<th>Standard</th>
<th>Graphics</th>
<th>QuickWin</th>
<th>Graphics</th>
<th>Windows</th>
<th>DLL</th>
<th>Lib</th>
</tr>
</thead>
</table>

**See Also:** [“SPORT\_RELEASE”](#), "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft® Platform SDK

**Example**

```fortran
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_CONNECT( 2 )
END
```

## SPORT\_CONNECT\_EX

**Serial Port I/O Function:** Establishes the connection to a serial port, defines certain usage parameters, and defines the size of the internal buffer for data reception. This function is only available on Windows® systems.

**Module:** USE IFPORT

**Syntax**

```
result = SPORT\_CONNECT\_EX (port [, options] [, BufferSize])
```
port
(Input) Integer. The port number of connection. The routine will open COMn, where n is the port number specified.

options
(Optional; input) Integer. Defines the connection options. These options define how the nnn_LINE routines will work and also effect the data that is passed to the user. If more than one option is specified, the operator .OR. should be used between each option. Options are as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_TOSS_CR</td>
<td>Removes carriage return (CR) characters on input.</td>
</tr>
<tr>
<td>DL_TOSS_LF</td>
<td>Removes linefeed (LF) characters on input.</td>
</tr>
<tr>
<td>DL_OUT_CR</td>
<td>Causes SPORT_WRITE_LINE to add a CR to each record written.</td>
</tr>
<tr>
<td>DL_OUT_LF</td>
<td>Causes SPORT_WRITE_LINE to add a LF to each record written.</td>
</tr>
<tr>
<td>DL_TERM_CR</td>
<td>Causes SPORT_READ_LINE to terminate READ when a CR is encountered.</td>
</tr>
<tr>
<td>DL_TERM_LF</td>
<td>Causes SPORT_READ_LINE to terminate READ when a LF is encountered.</td>
</tr>
<tr>
<td>DL_TERM_CRLF</td>
<td>Causes SPORT_READ_LINE to terminate READ when CR+LF is encountered.</td>
</tr>
</tbody>
</table>

If options is not specified, the following occurs by default:

(DL_OUT_CR .OR. DL_TERM_CR .OR. DL_TOSS_CR .OR. DL_TOSS_LF)

This specifies to remove carriage returns and linefeeds on input, to follow output lines with a carriage return, and to return input lines when a carriage return is encountered.

BufferSize
(Optional; input) Integer. Size of the internal buffer for data reception. If BufferSize is not specified, the size of the buffer is 16384 bytes (the default).

The size of the buffer must be 4096 bytes or larger. If you try to specify a size smaller than 4096 bytes, your specification will be ignored and the buffer size will be set to 4096 bytes.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_CONNECT”, “SPORT_RELEASE”, ”Using the Serial I/O Port Routines” in Building Applications, ”Communications and "Communications Functions” in the Microsoft* Platform SDK
Example
USE IFPORT
INTEGER(4) iresult
iresult =  SPORT_CONNECT_EX( 2, BufferSize = 8196 )
END

SPORT_GET_HANDLE

Serial Port I/O Function: Returns the Windows* handle associated with the communications port. This is the handle that was returned by the Windows API CreateFile. This function is only available on Windows systems.
Module: USE IFPORT
Syntax
result = SPORT_GET_HANDLE (port, handle)
port
(Input) Integer. The port number.
handle
(Output) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. This is the Windows handle that was returned from CreatFile( ) on the serial port.
Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB
See Also: "Using the Serial I/O Port Routines" in Building Applications, "Communications and Communications Functions" in the Microsoft* Platform SDK
Example
USE IFPORT
INTEGER(4) iresult
INTEGER(KIND=INT_PTR_KIND( )) handle
iresult =  SPORT_GETHANDLE( 2, handle )
END
SPORT_GET_STATE

Serial Port I/O Function: Returns the baud rate, parity, data bits setting, and stop bits setting of the communications port. This function is only available on Windows® systems.

Module: USE IFPORT

Syntax

result = SPORT_GET_STATE (port [, baud] [, parity] [, dbits] [, sbits])

port
(Input) Integer. The port number.

baud
(Optional; output) Integer. The baud rate of the port.

parity
(Optional; output) Integer. The parity setting of the port (0 - 4 = no, odd, even, mark, space).

dbits
(Optional; output) Integer. The data bits for the port.

sbits
(Optional; output) Integer. The stop bits for the port (0, 1, 2 = 1, 1.5, 2).

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_SET_STATE”, “Using the Serial I/O Port Routines” in Building Applications, "Communications and "Communications Functions" in the Microsoft® Platform SDK

Example

USE IFPORT
INTEGER(4) iresult
INTEGER    baud
INTEGER    parity
INTEGER    dbits
INTEGER    sbits

iresult =  SPORT_GET_STATE( 2, baud, parity, dbits, sbits )
END
SPORT_GET_STATE_EX

Serial Port I/O Function: Returns the baud rate, parity, data bits setting, stop bits, and other settings of the communications port. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax


port
(Input) Integer. The port number.

baud
(Optional; output) Integer. The baud rate of the port.

parity
(Optional; output) Integer. The parity setting of the port (0 - 4 = no, odd, even, mark, space).

dbits
(Optional; output) Integer. The data bits for the port.

sbits
(Optional; output) Integer. The stop bits for the port (0, 1, 2 = 1, 1.5, 2).

Binmode
(Optional; output) Integer. 1 if binary mode is enabled; otherwise, 0. Currently, the value of this parameter is always 1.

DTRcntrl
(Optional; output) Integer. 1 if DTR (data-terminal-ready) flow control is used; otherwise, 0.

RTScntrl
(Optional; output) Integer. 1 if RTS (request-to-send) flow control is used; otherwise, 0.

OutCTSFlow
(Optional; output) Integer. 1 if the CTS (clear-to-send) signal is monitored for output flow control; otherwise, 0.

OutDSRFlow
(Optional; output) Integer. 1 if the DSR (data-set-ready) signal is monitored for output flow control; otherwise, 0.
**DSRSense**
(Optional; output) Integer. 1 if the communications driver is sensitive to the state of the DSR signal; otherwise, 0.

**OutXonOff**
(Optional; output) Integer. 1 if XON/XOFF flow control is used during transmission; otherwise, 0.

**InXonOff**
(Optional; output) Integer. 1 if XON/XOFF flow control is used during reception; otherwise, 0.

**XonLim**
(Optional; output) Integer. The minimum number of bytes accepted in the input buffer before the XON character is set.

**XoffLim**
(Optional; output) Integer. The maximum number of bytes accepted in the input buffer before the XOFF character is set.

**TXContOnXoff**
(Optional; output) Integer. 1 if transmission stops when the input buffer is full and the driver has transmitted the XoffChar character; otherwise, 0.

**ErrAbort**
(Optional; output) Integer. 1 if read and write operations are terminated when an error occurs; otherwise, 0.

**ErrCharEnbl**
(Optional; output) Integer. 1 if bytes received with parity errors are replaced with the ErrChar character; otherwise, 0.

**NullStrip**
(Optional; output) Integer. 1 if null bytes are discarded; otherwise, 0.

**XonChar**
(Optional; output) Character. The value of the XON character that is used for both transmission and reception.

**XoffChar**
(Optional; output) Character. The value of the XOFF character that is used for both transmission and reception.
**ErrChar**
(Optional; output) Character. The value of the character that is used to replace bytes received with parity errors.

**EofChar**
(Optional; output) Character. The value of the character that is used to signal the end of data.

**EvtChar**
(Optional; output) Character. The value of the character that is used to signal an event.

**Results:**
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

**Compatibility**
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "SPORT_SET_STATE_EX", "Using the Serial I/O Port Routines" in *Building Applications*, "Communications and "Communications Functions" in the Microsoft® Platform SDK

**Example**

```fortran
USE IFPORT
INTEGER(4) iresult
INTEGER(4) port, baud, parity, dbits, sbits
INTEGER(4) OutXonOff, InXonOff, OutDSRFlow
INTEGER(4) OutCTSFlow, DTRcntrl, RTScntrl
INTEGER(4) DSRSense, XonLim, XoffLim
CHARACTER(1) XonChar, XoffChar
iresult = SPORT_GET_STATE_EX(port, baud, parity, dbits, sbits, &
   OutXonOff=OutXonOff, InXonOff=InXonOff, OutDSRFlow=OutDSRFlow, &
   OutCTSFlow=OutCTSFlow, DTRcntrl=DTRcntrl, RTScntrl=RTScntrl, &
   DSRSense = DSRSense, XonChar = XonChar, XoffChar = XoffChar, &
   XonLim=XonLim, XoffLim=XoffLim)
END
```

**SPORT_GET_TIMEOUTS**

**Serial Port I/O Function:** Returns the user selectable timeouts for the serial port. This function is only available on Windows® systems.

**Module:** USE IFPORT
Syntax

\[
\text{result} = \text{SPORT\_GET\_TIMEOUTS} \left( \text{port} \ [, \ \text{rx\_int} \ [, \ \text{tx\_tot\_mult} \ [, \ \text{tx\_tot\_const}] \right)
\]

\text{port}  
(Input) Integer. The port number.

\text{rx\_int}  
(Optional; output) INTEGER(4). The receive interval timeout value.

\text{tx\_tot\_mult}  
(Optional; output) INTEGER(4). The transmit multiplier part of the timeout value.

\text{tx\_tot\_const}  
(Optional; output) INTEGER(4). The transmit constant part of the timeout value.

\text{Results:}

The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

\text{Compatibility}

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

\text{See Also:} "SPORT\_SET\_TIMEOUTS", "Using the Serial I/O Port Routines" in Building Applications, "Communications and Communications Functions" in the Microsoft* Platform SDK

\text{Example}

\begin{verbatim}
USE IFPORT
INTEGER(4) iresult
INTEGER*4 rx_int
INTEGER*4 tx_tot_mult
INTEGER*4 tx_tot_const

iresult = SPORT\_GET\_TIMEOUTS( 2, rx_int, tx_tot_mult, tx_tot\_const )
END
\end{verbatim}

\textbf{SPORT\_PEEK\_DATA}

\textbf{Serial Port I/O Function:} Returns information about the availability of input data. This function is only available on Windows* systems.

\textbf{Module:} USE IFPORT

Syntax

\[
\text{result} = \text{SPORT\_PEEK\_DATA} \left( \text{port} \ [, \ \text{present} \ [, \ \text{count}] \right)
\]

Descriptions of the Library Routines 2
PORT_PEEK_DATA

port
(Input) Integer. The port number.

present
(Optional; output) Integer. 1 if data is present, 0 if no data has been read.

count
(Optional; output) Integer. The count of characters that will be returned by
SPORT_READ_DATA.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

NOTE. CR and LF characters may not be returned depending on the mode
specified in the SPORT_CONNECT( ) call.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "SPORT_CONNECT", "SPORT_READ_DATA", "SPORT_PEEK_LINE", "Using the
Serial I/O Port Routines" in Building Applications, "Communications and "Communications
Functions" in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
INTEGER    present
INTEGER    count

iresult =  SPORT_PEEK_DATA( 2, present, count )
END

SPORT_PEEK_LINE

Serial Port I/O Function: Returns information about the availability of input records. This
function is only available on Windows* systems.

Module: USE IFPORT

Syntax
   result = SPORT_PEEK_LINE (port [, present] [, count])
**port**
(Input) Integer. The port number.

**present**
(Optional; output) Integer. 1 if data is present, 0 if no data has been read.

**count**
(Optional; output) Integer. The count of characters that will be returned by SPORT_READ_DATA.

**Results:**
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

This routine will only return when a line terminator has been seen - as defined by the mode specified in the SPORT_CONNECT( ) call.

---

**NOTE.** CR and LF characters may not be returned depending on the mode specified in the SPORT_CONNECT( ) call.

---

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "SPORT_CONNECT", "SPORT_READ_DATA", "SPORT_PEEK_DATA", "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft* Platform SDK

**Example**

USE IFPORT
INTEGER(4) iresult
INTEGER    present
INTEGER    count

iresult = SPORT_PEEK_LINE( 2, present, count )
END

**SPORT_PURGE**

**Serial Port I/O Function:** Executes the Windows* API communications function PurgeComm on the specified port. This function is only available on Windows systems.

**Module:** USE IFPORT
Syntax

\[
\text{result} = \text{SPORT\_PURGE} (\text{port}, \text{function}) \\
\]

\text{port}

(Input) Integer. The port number.

\text{function}

(Input) INTEGER(4). The function for PurgeComm (see the Windows documentation).

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "Using the Serial I/O Port Routines" in Building Applications, "Communications and Communications Functions" in the Microsoft* Platform SDK

Example

USE IFWINTY
USE IFPORT
INTEGER(4) iresult
iresult =  SPORT\_PURGE( 2, (PURGE\_TXABORT .or. PURGE\_RXABORT) )
END

SPORT\_READ\_DATA

Serial Port I/O Function: Reads available data from the specified port. This routine stalls until at least one character has been read. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax

\[
\text{result} = \text{SPORT\_READ\_DATA} (\text{port}, \text{buffer}[\text{, count}]) \\
\]

\text{port}

(Input) Integer. The port number.

\text{buffer}

(Output) Character*(*) . The data that was read.

\text{count}

(Optional; output) Integer. The count of bytes read.
Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

NOTE. CR and LF characters may not be returned depending on the mode specified in the SPORT_CONNECT() call.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

See Also: “SPORT_CONNECT”, “SPORT_PEEK_DATA”, “SPORT_READ_LINE”, “SPORT_WRITE_DATA”, "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4)      iresult
INTEGER         count
CHARACTER*1024  rbuff

iresult = SPORT_READ_DATA( 2, rbuff, count )
END

SPORT_READ_LINE

Serial Port I/O Function: Reads a record from the specified port. This routine stalls until at least one record has been read. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax
   result = SPORT_READ_LINE ( port, buffer [, count])

port
(Input) Integer. The port number.

buffer
(Output) Character*(*) . The data that was read.

count
(Optional; output) Integer. The count of bytes read.
Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.
This routine will only return when a line terminator has been seen – as defined by the mode specified in the SPORT_CONNECT( ) call.

NOTE. CR and LF characters may not be returned depending on the mode specified in the SPORT_CONNECT( ) call.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_CONNECT”, “SPORT_PEEK_LINE”, “SPORT_READ_DATA”, “SPORT_WRITE_LINE”, "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft* Platform SDK

Example

USE IFPORT
INTEGER(4) iresult
INTEGER count
CHARACTER*1024 rbuff

iresult = SPORT_READ_LINE( 2, rbuff, count )
END

SPORT_RELEASE

Serial Port I/O Function: Releases a serial port that was previously connected to (by using SPORT_CONNECT). This function is only available on Windows* systems.

Module: USE IFPORT

Syntax

result = SPORT_RELEASE (port)

port

(Input) Integer. The port number.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "SPORT_CONNECT", "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_RELEASE( 2 )
END

SPORT_SET_STATE

Serial Port I/O Function: Sets the baud rate, parity, data bits setting, and stop bits setting of the communications port. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax
result = SPORT_SET_STATE ( port [, baud ] [, parity ] [, dbits ] [, sbits] )

port
(Input) Integer. The port number.

baud
(Optional; input) Integer. The baud rate of the port.

parity
(Optional; input) Integer. The parity setting of the port (0 – 4 = no, odd, even, mark, space).

dbits
(Optional; input) Integer. The data bits for the port.

sbits
(Optional; input) Integer. The stop bits for the port (0, 1, 2 = 1, 1.5, 2).

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value. The following restrictions apply:
- The number of data bits must be 5 to 8 bits.
- The use of 5 data bits with 2 stop bits is an invalid combination, as is 6, 7, or 8 data bits with 1.5 stop bits.
**NOTE.** This routine must not be used when any I/O is pending. Since a read operation is always pending after any I/O has been started, you must first call SPORT_CANCEL_IO before port parameters can be changed.

**Compatibility**

CONSOLE STANDARD GRAPhICS QUICKWIN GRAPhICS WINDOWS DLL LIB

**See Also:** “SPORT_CANCEL_IO”, “SPORT_GET_STATE”, "Using the Serial I/O Port Routines” in Building Applications, "Communications and "Communications Functions” in the Microsoft* Platform SDK

**Example**

USE IFPORT
INTEGER(4) iresult
iresult =  SPORT_SET_STATE( 2, 9600, 0, 7, 1 )

END

### SPORT_SET_STATE_EX

**Serial Port I/O Function:** Sets the baud rate, parity, data bits setting, stop bits, and other settings of the communications port. This function is only available on Windows* systems.

**Module:** USE IFPORT

**Syntax**

```fortran
```

- **port**
  (Input) Integer. The port number.

- **baud**
  (Optional; input) Integer. The baud rate of the port.

- **parity**
  (Optional; input) Integer. The parity setting of the port (0 - 4 = no, odd, even, mark, space).
dbits
(Optional; input) Integer. The data bits for the port.

sbits
(Optional; input) Integer. The stop bits for the port (0, 1, 2 = 1, 1.5, 2).

Binmode
(Optional; input) Integer. 1 if binary mode should be enabled; otherwise, 0. Currently, if this parameter is used, the value must be 1.

DTRctrl
(Optional; input) Integer. 1 if DTR (data-terminal-ready) flow control should be used; otherwise, 0.

RTSctrl
(Optional; input) Integer. 1 if RTS (request-to-send) flow control should be used; otherwise, 0.

OutCTSFlow
(Optional; input) Integer. 1 if the CTS (clear-to-send) signal should be monitored for output flow control; otherwise, 0.

OutDSRFlow
(Optional; input) Integer. 1 if the DSR (data-set-ready) signal should be monitored for output flow control; otherwise, 0.

DSRSense
(Optional; input) Integer. 1 if the communications driver should be sensitive to the state of the DSR signal; otherwise, 0.

OutXonOff
(Optional; input) Integer. 1 if XON/XOFF flow control should be used during transmission; otherwise, 0.

InXonOff
(Optional; input) Integer. 1 if XON/XOFF flow control should be used during reception; otherwise, 0.

XonLim
(Optional; input) Integer. The minimum number of bytes that should be accepted in the input buffer before the XON character is set.

XoffLim
(Optional; input) Integer. The maximum number of bytes that should be accepted in the input buffer before the XOFF character is set.
**TXContOnXoff**
(Optional; input) Integer. 1 if transmission should be stopped when the input buffer is full and the driver has transmitted the *XoffChar* character; otherwise, 0.

**ErrAbort**
(Optional; input) Integer. 1 if read and write operations should be terminated when an error occurs; otherwise, 0.

**ErrCharEnbl**
(Optional; input) Integer. 1 if bytes received with parity errors should be replaced with the *ErrChar* character; otherwise, 0.

**NullStrip**
(Optional; input) Integer. 1 if null bytes should be discarded; otherwise, 0.

**XonChar**
(Optional; input) Character. The value of the XON character that should be used for both transmission and reception.

**XoffChar**
(Optional; input) Character. The value of the XOFF character that should be used for both transmission and reception.

**ErrChar**
(Optional; input) Character. The value of the character that should be used to replace bytes received with parity errors.

**EofChar**
(Optional; input) Character. The value of the character that should be used to signal the end of data.

**EvtChar**
(Optional; input) Character. The value of the character that should be used to signal an event.

**fZeroDCB**
(Optional; input) Integer. 1 if all settings of the communications port should be set to zero before parameters are set; otherwise, 0.

**Results:**
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value. The following restrictions apply:
- The number of data bits must be 5 to 8 bits.
• The use of 5 data bits with 2 stop bits is an invalid combination, as is 6, 7, or 8 data bits with 1.5 stop bits.

NOTE. This routine must not be used when any I/O is pending. Since a read operation is always pending after any I/O has been started, you must first call SPORT_CANCEL_IO before port parameters can be changed.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_CANCEL_IO”, “SPORT_GET_STATE_EX”, “Using the Serial I/O Port Routines” in Building Applications, “Communications and “Communications Functions” in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_SET_STATE_EX( 2, 9600, 0, 7, 1, OutXonOff=1, InXonOff=1, &
XonLim=1024, XoffLim=512, XonChar=CHAR(17), XoffChar=CHAR(19), &
fZeroDCB=1) )
END

SPORT_SET_TIMEOUTS

Serial Port I/O Function: Sets the user selectable timeouts for the serial port. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax
result = SPORT_SET_TIMEOUTS (port [, rx_int] [, tx_tot_mult] [, tx_tot_const])

port
(Input) Integer. The port number.

rx_int
(Optional; input) INTEGER(4). The receive interval timeout value.

tx_tot_mult
(Optional; input) INTEGER(4). The transmit multiplier part of the timeout value.
tx_tot_const
(Optional; input) INTEGER(4). The transmit constant part of the timeout value.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

NOTE. This routine must not be used when any I/O is pending. Since a read operation is always pending after any I/O has been started, you must first call SPORT_CANCEL_IO before port parameters can be changed.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_CANCEL_IO”, “SPORT_GET_STATE_EX”, "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_SET_TIMEOUTS( 2, 100, 0, 1000 )
END

SPORT_SHOW_STATE

Serial Port I/O Function: Displays the state of a port to standard output. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax
   result = SPORT_SHOW_STATE (port, level)

port
(Input) Integer. The port number.

level
(Input) Integer. Controls the level of detail displayed as follows:

<table>
<thead>
<tr>
<th>level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Basic one line display</td>
</tr>
<tr>
<td>1</td>
<td>Basic information</td>
</tr>
</tbody>
</table>
\textbf{Descriptions of the Library Routines}

\section*{SPORT_SPECIAL_FUNC}

\textbf{Serial Port I/O Function}: Executes the Windows\textsuperscript{*} API communications function \texttt{EscapeCommFunction} on the specified port. This function is only available on Windows systems.

\textbf{Module}: USE IFPORT

\textbf{Syntax}

\begin{verbatim}
result = SPORT_SPECIAL_FUNC (port, function)
\end{verbatim}

\texttt{port}

(Input) Integer. The port number.

\begin{itemize}
  \item 2 Add modem signal control flow information
  \item 3 Add XON/XOFF information
  \item 4 Add event character information
  \item 11 Add timeout information
  \item 901 Add debug information
\end{itemize}

\begin{itemize}
  \item \textbf{Results}: The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.
\end{itemize}

\begin{itemize}
  \item \textbf{NOTE.} This routine must not be used when any I/O is pending. Since a read operation is always pending after any I/O has been started, you must first call \texttt{SPORT_CANCEL_IO} before port parameters can be changed.
\end{itemize}

\begin{itemize}
  \item \textbf{Compatibility}
\end{itemize}

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

\textbf{See Also}: "SPORTCANCELIO", "Using the Serial I/O Port Routines" in \textit{Building Applications}, "Communications and "Communications Functions" in the Microsoft\textsuperscript{*} Platform SDK

\textbf{Example}

\begin{verbatim}
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_SHOW_STATE( 2, 0 )
END
\end{verbatim}
function
(Input) INTEGER(4). The function to perform.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS  DLL  LIB

See Also: "Using the Serial I/O Port Routines” in Building Applications, "Communications and
"Communications Functions” in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult =  SPORT_SPECIAL_FUNC( 2, ? )
END

SPORT_WRITE_DATA

Serial Port I/O Function: Outputs data to the specified port. This function is only available on
Windows* systems.

Module: USE IFPORT

Syntax
result = SPORT_WRITE_DATA (port, data [, count])

port
(Input) Integer. The port number.

data
(Input) Character*(*). The data to be output.

count
(Optional; input) Integer. The count of bytes to write. If the value is zero, this number is computed
by scanning the data backwards looking for a non-blank character.

Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.
NOTE. When hardware (DTR, RTS, etc.) or software (XON/XOFF) flow controls are used, the functions SPORT_WRITE_DATA and SPORT_WRITE_LINE can write less bytes than required. When this occurs, the functions return the code ERROR_IO_INCOMPLETE, and the return value of parameter "count" contains the number of bytes that were really written.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_WRITE_LINE”, “SPORT_READ_DATA”, "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions" in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_WRITE_DATA( 2, 'ATZ'//CHAR(13), 0 )
END

SPORT_WRITE_LINE

Serial Port I/O Function: Outputs data, followed by a record terminator, to the specified port. This function is only available on Windows* systems.

Module: USE IFPORT

Syntax
result = SPORT_WRITE_LINE (port, data [, count])

port
(Input) Integer. The port number.

data
(Input) Character*(*) . The data to be output.

count
(Optional; input) Integer. The count of bytes to write. If the value is zero, this number is computed by scanning the data backwards looking for a non-blank character.
Results:
The result type is INTEGER(4). The result is zero if successful; otherwise, a Windows error value.
After the data is output, a line terminator character is added based on the mode used during the
SPORT_CONNECT( ) call.

NOTE. When hardware (DTR, RTS, etc.) or software (XON/XOFF) flow controls are used, the functions SPORT_WRITE_DATA and
SPORT_WRITE_LINE can write less bytes than required. When this occurs, the functions return the code ERROR_IO_INCOMPLETE, and the return
value of parameter "count" contains the number of bytes that were really written.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SPORT_CONNECT”, “SPORT_WRITE_DATA”, “SPORT_READ_DATA”, "Using the Serial I/O Port Routines" in Building Applications, "Communications and "Communications Functions” in the Microsoft* Platform SDK

Example
USE IFPORT
INTEGER(4) iresult
iresult = SPORT_WRITE_LINE( 2, 'ATZ', 0 )
END

SRAND
Portability Subroutine: Seeds the random number generator used with IRAND and RAND.
Module: USE IFPORT
Syntax
   CALL SRAND (iseed)
iseed
(Input) INTEGER(4). Any value. The default value is 1.
SRAND seeds the random number generator used with IRAND and RAND. Calling SRAND is
equivalent to calling IRAND or RAND with a new seed.
The same value for iseed generates the same sequence of random numbers. To vary the sequence, call SRAND with a different iseed value each time the program is executed.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "RAND, RANDOM", "IRAND, IRANDM", the RANDOM_NUMBER and RANDOM_SEED intrinsic subroutines in the *Language Reference*

**Example**

```fortran
! How many random numbers out of 100 will be between .5 and .6?
USE IFPORT
ICOUNT = 0
CALL SRAND(123)
DO I = 1, 100
   X = RAND(0)
   IF ((X > .5).AND.(X < .6)) ICOUNT = ICOUNT + 1
END DO
WRITE(*,*) ICOUNT, "numbers between .5 and .6!"
END
```

**SSWRQQ**

Portability Subroutine: Returns the floating-point processor status word.

Module: USE IFPORT

Syntax

```fortran
CALL SSWRQQ (status)
```

status

(Output) INTEGER(2). Floating-point processor status word.

SSWRQQ performs the same function as the run-time subroutine GETSTATUSFPQQ and is provided for compatibility.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "LCWRQQ", "GETSTATUSFPQQ"

**Example**

```fortran
USE IFPORT
INTEGER(2) status
CALL SSWRQQ (status)
```
STAT

Portability Function: Returns detailed information about a file.

Module: USE IFPORT

Syntax

\[
\text{result} = \text{STAT} \left( \text{name}, \text{statb} \right)
\]

\text{name}

(Input) Character*(*) . Name of the file to examine.

\text{statb}

(Output) INTEGER(4) or INTEGER(8) . One-dimensional array of size 12; where the system information is stored. The elements of \text{statb} contain the following values:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Values or Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{statb}(1)</td>
<td>Device the file resides on</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 0 \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(2)</td>
<td>File inode number</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 0 \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(3)</td>
<td>Access mode of the file</td>
<td>See the table in Results</td>
</tr>
<tr>
<td>\text{statb}(4)</td>
<td>Number of hard links to the file</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 1 \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(5)</td>
<td>User ID of owner</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 1 \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(6)</td>
<td>Group ID of owner</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 1 \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(7)</td>
<td>Raw device the file resides on</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 0 \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(8)</td>
<td>Size of the file</td>
<td></td>
</tr>
<tr>
<td>\text{statb}(9)</td>
<td>Time when the file was last accessed(^1)</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Only available on non-FAT file systems; undefined on FAT systems \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(10)</td>
<td>Time when the file was last modified(^1)</td>
<td></td>
</tr>
<tr>
<td>\text{statb}(11)</td>
<td>Time of last file status change(^1)</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Same as stat(10) \text{L}^{*X}: System dependent</td>
</tr>
<tr>
<td>\text{statb}(12)</td>
<td>Blocksize for file system I/O operations</td>
<td>\text{W}^{*32}, \text{W}^{*64}: Always 1 \text{L}^{*X}: System dependent</td>
</tr>
</tbody>
</table>

\(^1\) Times are in the same format returned by the TIME function (number of seconds since 00:00:00 Greenwich mean time, January 1, 1970).
Results:
The result type is INTEGER(4).

On Windows* systems, the result is zero if the inquiry was successful; otherwise, the error code ENOENT (the specified file could not be found). On Linux* systems, the file inquired about must be currently connected to a logical unit and must already exist when STAT is called; if STAT fails, errno is set.

For a list of other error codes, see “IERRNO”.

The access mode (the third element of statb) is a bitmap consisting of an IOR of the following constants:

<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 on Windows* systems</td>
</tr>
<tr>
<td>S_IFBLK</td>
<td>O'0060000'</td>
<td>Block special</td>
<td>Never set on Windows systems</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 on Windows systems</td>
</tr>
<tr>
<td>S_IFSOCK</td>
<td>O'0140000'</td>
<td>Socket</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_ISUID</td>
<td>O'0004000'</td>
<td>Set user ID on execution</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>O'0002000'</td>
<td>Set group ID on execution</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_ISVTX</td>
<td>O'0001000'</td>
<td>Save swapped text</td>
<td>Never set on Windows systems</td>
</tr>
<tr>
<td>S_IRWXU</td>
<td>O'0000700'</td>
<td>Owner's file permissions</td>
<td></td>
</tr>
<tr>
<td>S_IRUSR, S_IREAD</td>
<td>O'0000400'</td>
<td>Owner's read permission</td>
<td>Always true on Windows systems</td>
</tr>
<tr>
<td>S_IWUSR, S_IWRITE</td>
<td>O'0000200'</td>
<td>Owner's write permission</td>
<td>Based on file extension (.EXE, .COM, .CMD, or .BAT)</td>
</tr>
<tr>
<td>S_IXUSR, S_IEXEC</td>
<td>O'0000100'</td>
<td>Owner's execute permission</td>
<td>Same as S_IRWXU on Windows systems</td>
</tr>
<tr>
<td>S_IRWXG</td>
<td>O'0000070'</td>
<td>Group's file permissions</td>
<td>Same as S_IRWXU on Windows systems</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>O'0000040'</td>
<td>Group's read permission</td>
<td>Same as S_IRUSR on Windows systems</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>O'0000020'</td>
<td>Group's write permission</td>
<td>Same as S_IWUSR on Windows systems</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>O'0000010'</td>
<td>Group's execute permission</td>
<td>Same as S_IXUSR on Windows systems</td>
</tr>
<tr>
<td>S_IRWXO</td>
<td>O'0000007'</td>
<td>Other's file permissions</td>
<td>Same as S_IRWXU on Windows systems</td>
</tr>
</tbody>
</table>
STAT returns the same information as FSTAT, but accesses files by name instead of external unit number.

On Windows systems, LSTAT returns exactly the same information as STAT. On Linux systems, 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.

You can also use the INQUIRE statement to get information about file properties.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** “GETFILEINFOQQ”, the INQUIRE statement in the Language Reference

**Example**

```fortran
USE IFPORT
CHARACTER*12 file_name
INTEGER(4) info_array(12)
print *, 'Enter file to examine: '
read *, file_name
ISTATUS = STAT (file_name, info_array)
if (.not. istatus) then
    print *, info_array
else
    print *, 'Error = ', istatus
end if
```

**SYSTEM**

**Portability Function:** Sends a command to the shell as if it had been typed at the command line.

**Module:** USE IFPORT

<table>
<thead>
<tr>
<th>Symbolic name</th>
<th>Constant</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IROTH</td>
<td>O’0000004’</td>
<td>Other’s read permission</td>
<td>Same as S_IRUSR on Windows systems</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>O’0000002’</td>
<td>Other’s write permission</td>
<td>Same as S_IWUSR on Windows systems</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>O’0000001’</td>
<td>Other’s execute permission</td>
<td>Same as S_IXUSR on Windows systems</td>
</tr>
</tbody>
</table>
Syntax

result = SYSTEM (string)

string
(Input) Character*(*) Operating system command.

Results:
The result type is INTEGER(4). The result is the exit status of the shell command. If –1, use "IERNNO" to retrieve the error. Errors can be one of the following:

• E2BIG – The argument list is too long.
• ENOENT – The command interpreter cannot be found.
• ENOEXEC – The command interpreter file has an invalid format and is not executable.
• ENOMEM – Not enough system resources are available to execute the command.

On Windows* systems, the calling process waits until the command terminates. To insure compatibility and consistent behavior, an image can be invoked directly by using the Windows* API CreateProcess( ) in your Fortran code.

Commands run with the SYSTEM routine are run in a separate shell. Defaults set with the SYSTEM function, such as current working directory or environment variables, do not affect the environment the calling program runs in.

The command line character limit for the SYSTEM function is the same limit that your operating system command interpreter accepts.

Compatibility

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SYSTEMQQ”

Example

USE IFPORT
INTEGER(4) I, errnum
I = SYSTEM("dir > file.lst")
If (I .eq. -1) then
   errnum = ierrno( )
   print *, 'Error ', errnum
end if
END
**SYSTEMQQ**

**Portability Function:** Executes a system command by passing a command string to the operating system’s command interpreter.

**Module:** USE IFPORT

**Syntax**

```
result = SYSTEMQQ (commandline)
```

`commandline`  
(Input) Character*(*). Command to be passed to the operating system.

**Results:**

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

The SYSTEMQQ function lets you pass operating-system 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* systems, the calling process waits until the command terminates. To insure compatibility and consistent behavior, an image can be invoked directly by using the Windows* API CreateProcess( ) in your Fortran code.

If the function fails, call **“GETLASTERRORQQ”** to determine the reason. One of the following errors can 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 operating system command interpreter accepts.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** **“SYSTEM”**

**Example**

```
USE IFPORT
LOGICAL(4) result
```
result = SYSTEMQQ('copy c:\bin\fmath.dat &
c:\dat\fmath2.dat')

TIME

Portability Function or Subroutine: The function returns the system time, in seconds, since 00:00:00 Greenwich mean time, January 1, 1970. The subroutine fills a parameter with the current time as a string in the format hh:mm:ss.

Module: USE IFPORT

Function Syntax:
result = TIME ( )

Subroutine Syntax:
CALL TIME ( string )

string
(Output) Character*(*) . Current time, based on a 24-hour clock, in the form hh:mm:ss, where hh, mm, and ss are two-digit representations of the current hour, minutes past the hour, and seconds past the minute, respectively.

Results:
The result type is INTEGER(4). The result value is the number of seconds that have elapsed since 00:00:00 Greenwich mean time, January 1, 1970.
The value returned by this routine can be used as input to other portability date and time functions.

NOTE. TIME is an intrinsic procedure unless you specify USE IFPORT.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS LIB

See Also: the TIME subroutine in the Language Reference

Example
USE IFPORT
INTEGER(4) int_time
character*8 char_time
int_time = TIME( )
call TIME(char_time)
print *, 'Integer: ', int_time, 'time: ', char_time
TIMEF

**Portability Function:** Returns the number of seconds since the first time it is called, or zero.

**Module:** USE IFPORT

**Syntax**

```
result = TIMEF()
```

**Results:**

The result type is REAL(4). The result value is the number of seconds that have elapsed since the first time TIMEF was called.

The first TIMEF it is called, it returns 0.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**Example**

```
USE IFPORT
INTEGER i, j
REAL(8) elapsed_time
elapsed_time = TIMEF()
DO i = 1, 100000
    j = j + 1
END DO
elapsed_time = TIMEF()
PRINT *, elapsed_time
END
```

TRACEBACKQQ

**Run-Time Subroutine:** Provides traceback information. Uses the Intel® Visual Fortran run-time library traceback facility to generate a stack trace showing the program call stack as it appeared at the time of the call to TRACEBACKQQ().

**Module:** USE IFCORE

**Syntax**

```
CALL TRACEBACKQQ([string] [, user_exit_code] [, status] [, eptr])
```
string
(Optional; input) CHARACTER*(*) . A message string to precede the traceback output. It is recommended that the string be no more than 80 characters (one line) since that length appears better on output. However, this limit is not a restriction and it is not enforced. The string is output exactly as specified; no formatting or interpretation is done.
If this argument is omitted, no header message string is produced.

user_exit_code
(Optional; input) INTEGER(4). An exit code. Two values are predefined:
• A value of –1 causes the run-time system to return execution to the caller after producing traceback.
• A value of zero (the default) causes the application to abort execution.
Any other specified value causes the application to abort execution and return the specified value to the operating system.

status
(Optional; input) INTEGER(4). A status value. If specified, the run-time system returns the status value to the caller indicating that the traceback process was successful. The default is not to return status.
Note that a returned status value is only an indication that the "attempt" to trace the call stack was completed successfully, not that it produced a useful result.
You can include the file iosdef.for in your program to obtain symbolic definitions for the possible return values. A return value of FOR$IOS_SUCCESS (0) indicates success.

eptr
(Optional; input) Cray pointer. It is required if calling from a user-specified exception filter. If omitted, the default is null.
To trace the stack after an exception has occurred, the runtime support needs access to the exception information supplied to the filter by the operating system.
The eptr argument is a pointer to T_EXCEPTION_POINTERS, returned by the Windows* API GetExceptionInformation(), which is usually passed to a C try/except filter function. This argument must be null if you are not passing a valid pointer to T_EXCEPTION_POINTERS. For more information, see "Obtaining Traceback Information with TRACEBACKQQ" in Building Applications.
The TRACEBACKQQ routine provides a standard way for an application to initiate a stack trace. It can be used to report application detected errors, debugging, and so forth. It uses the stack trace support in the Intel Visual Fortran run-time library, and produces the same output that the run-time library produces for unhandled errors and exceptions.
The error message string normally included by the run-time system is replaced with the user-supplied message text, or omitted if no string is specified. Traceback output is directed to the target destination appropriate for the application type, just as it is when traceback is initiated internally by the run-time system.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

**See Also:** "GETEXCEPTIONPTRQQ", and "Obtaining Traceback Information with TRACEBACKQQ", "Using Traceback Information", and "Run-Time Message Display and Format" in Building Applications

**Examples**

The following example generates a traceback report with no leading header message, from wherever the call site is, and aborts execution:

```fortran
USE IFCORE
CALL TRACEBACKQQ( )
```

The following example generates a traceback report with the user-supplied string as the header, and aborts execution:

```fortran
USE IFCORE
CALL TRACEBACKQQ("My application message string")
```

The following example generates a traceback report with the user-supplied string as the header, and aborts execution, returning a status code of 123 to the operating system:

```fortran
USE IFCORE
CALL TRACEBACKQQ(STRING="Bad value for TEMP",USER_EXIT_CODE=123)
```

Consider the following:

```fortran
...  
USE IFCORE
INTEGER(4) RTN_STS
INCLUDE 'IOSDEF.FOR'
...  
CALL TRACEBACKQQ(USER_EXIT_CODE=-1,STATUS=RTN_STS)
IF (RTN_STS .EQ. FOR$IOS_SUCCESS) THEN
  PRINT *,"TRACEBACK WAS SUCCESSFUL"
END IF
...  
```

This example generates a traceback report with no header string, and returns to the caller to continue execution of the application. If the traceback process succeeds, a status will be returned in variable RTN_STS.
For more examples, including one showing a Cray pointer, see “Obtaining Traceback Information with TRACEBACKQQ” in Building Applications.

TTYNAM

**Portability Subroutine:** Specifies a terminal device name.

**Module:** USE IFPORT

**Syntax**

```fortran
CALL TTYNAM (string, lunit)
```

- `string` *(Output)* Character*(*). Name of the terminal device. If the Fortran logical unit is not connected to a terminal, it returns a string filled with blanks.
- `lunit` *(Input)* INTEGER(4). A Fortran logical unit number.

**Compatibility**

CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

UNLINK

**Portability Function:** Deletes the file given by path.

**Module:** USE IFPORT

**Syntax**

```fortran
result = UNLINK (name)
```

- `name` *(Input)* Character*(*). Path of the file to delete. The path can use forward (/) or backward (\) slashes as path separators and can contain drive letters.

**Results:**

The result type is INTEGER(4). The result is zero if successful; otherwise, an error code. Errors can be one of the following:

- **ENOENT:** The specified file could not be found.
- **EACCES:** The specified file is read-only.

You must have adequate permission to delete the specified file.

On Windows systems, you will get the EACCES error if the file has been opened by any process.
Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: “SYSTEM”, “DELDIRQQ”

Example
USE IFPORT
INTEGER(4) ISTATUS
CHARACTER*20 dirname
READ *, dirname
ISTATUS = UNLINK (dirname)
IF (ISTATUS) then
    print *, 'Error ', ISTATUS
END IF
END

UNPACKTIMEQQ

Portability Subroutine: Unpacks a packed time and date value into its component parts.
Module: USE IFPORT
Syntax
CALL UNPACKTIMEQQ (timedate, iyr, imon, iday, ihr,imin, isec)
timedate
(Input) INTEGER(4). Packed time and date information.
iyr
(Output) INTEGER(2). Year (xxxx AD).
imon
(Output) INTEGER(2). Month (1 - 12).
iday
(Output) INTEGER(2). Day (1 - 31).
ihr
(Output) INTEGER(2). Hour (0 - 23).
imin
(Output) INTEGER(2). Minute (0 - 59).
isec
(Output) INTEGER(2). Second (0 - 59).

GETFILEINFOQQ returns time and date in a packed format. You can use UNPACKTIMEQQ to unpack these values. Use PACKTIMEQQ to repack times for passing to SETFILETIMEQQ. Packed times can be compared using relational operators.

Compatibility
CONSOLE STANDARD GRAPHICS QUICKWIN GRAPHICS WINDOWS DLL LIB

See Also: "PACKTIMEQQ", "GETFILEINFOQQ"

Example
USE IFPORT
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 = GETFILEINFOQQ(file, info, handle)
CALL UNPACKTIMEQQ(info%lastwrite, iyr, imon,&
                   iday, ihr, imin, isec)
WRITE(*,*) iyr, imon, iday
WRITE(*,*) ihr, imin, isec
END

UNREGISTERMOUSEEVENT

QuickWin Function: Removes the callback routine registered for a specified window by an earlier call to REGISTERMOUSEEVENT. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax
result = UNREGISTERMOUSEEVENT (unit, mouseevents)

unit
(Input) INTEGER(4) on IA-32 processors; INTEGER(8) on Intel Itanium processors. 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 (defined in IFQWIN.F90) 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

Results:
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.

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.

Compatibility

QUICKWIN GRAPHICS LIB

See Also: “REGISTERMOUSEEVENT”, “WAITONMOUSEEVENT”, "Using QuickWin" in Building Applications

WAITONMOUSEEVENT

QuickWin Function: Waits for the specified mouse input from the user. This function is only available on Windows* systems.

Module: USE IFQWIN

Syntax

result = WAITONMOUSEEVENT (mouseevents, keystate, x, y)

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 - CTRL 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.

**Results:**
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.

`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.

**Compatibility**

QUICKWIN GRAPHICS LIB

**See Also:** “REGISTERMOUSEEVENT”, “UNREGISTERMOUSEEVENT”, "Using QuickWin" in Building Applications

**Example**

```fortran
USE IFQWIN
INTEGER(4) mouseevent, keystate, x, y, result
...
mouseevent = MOUSE$RBUTTONDOWN .OR. MOUSE$LBUTTONDOWN
result = WAITONMOUSEEVENT (mouseevent, keystate, x , y)
!
! Wait until right or left mouse button clicked, then check the keystate
! with the following:
!
if ((MOUSE$KS_SHIFT .AND. keystate) == MOUSE$KS_SHIFT) then &
  & write (*,*) 'Shift key was down'
if ((MOUSE$KS_CONTROL .AND. keystate) == MOUSE$KS_CONTROL) then &
  & write (*,*) 'Ctrl key was down'
```

**WRAPON**

**Graphics Function:** Controls whether text output is wrapped. This function is only available on Windows* systems.

**Module:** USE IFQWIN

**Syntax**

```
result = WRAPON (option)
```

**option**

(Input) INTEGER(2). Wrap mode. One of the following symbolic constants:

• $GWRAPOFF – Truncates lines at right edge of window border.
• $GWRAPON – Wraps lines at window border, scrolling if necessary.

**Results:**

The result type is INTEGER(2). The result is the previous value of *option*.
WRAPON 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.

Compatibility
STANDARD GRAPHICS  QUICKWIN GRAPHICS  LIB

See Also: "OUTTEXT", "SCROLLTEXTWINDOW", "SETTEXTEXPOSITION", "SETTEXTWINDOW"

Example
USE IFQWIN
INTEGER(2) row, status2
INTEGER(4) status4
TYPE ( rccoord ) curpos
TYPE ( windowconfig ) wc
LOGICAL status

status = GETWINDOWCONFIG( wc )
wc%numtextcols = 80
wc%numxpixels = -1
wc%numypixels = -1
wc%numtextrows = -1
wc%numcolors  = -1
wc%fontsize    = -1
wc%title = "This is a test"C
wc%bitsperpixel = -1
status = SETWINDOWCONFIG( wc )
status4 = SETBKCOLORRGB(#FF0000 )
CALL CLEARSCREEN( $GCLEARSCREEN )

! Display wrapped and unwrapped text in text windows.
CALL SETTEXTWINDOW( INT2(1),INT2(1),INT2(5),INT2(25))
CALL SETTEXTEXPOSITION(INT2(1),INT2(1), curpos )
status2 = WRAPON( $GWRAPON )
status4 = SETTEXTCOLORRGB(#00FF00)
DO i = 1, 5
    CALL OUTTEXT( 'Here text does wrap. ')
END DO
CALL SETTEXTWINDOW(INT2(7),INT2(10),INT2(11),INT2(40))
CALL SETTEXTEXPOSITION(INT2(1),INT2(1),curpos)
status2 = WRAPON($GWRAPOFF)
status4 = SETTEXTCOLORRGB(#008080)
DO row = 1, 5
    CALL SETTEXTPOSITION(INT2(row), INT2(1), curpos)
    CALL OUTTEXT('Here text does not wrap. ')
    CALL OUTTEXT('Here text does not wrap.')
END DO
READ (*,*) ! Wait for ENTER to be pressed
END
Index

A
ABORT, 2-2
  example of, 2-2
About box
  function specifying text for, 2-2
ABOUTBOXQQ (W*32, W*64), 2-2
  example of, 2-3
Absolute value
  function returning, 2-187
ACCESS, 2-3
  example of, 2-4, 2-28
Access mode
  function changing, 2-26
  function setting, 2-400
  function testing, 2-3
  POSIX subroutine changing, 2-271
ActiveX* controls (W*32), 2-64
  in a dialog box, 2-63
ALARM, 2-4
  example of, 2-5
Alarm
  POSIX subroutine scheduling, 2-266
AMOD
  See your language reference
Ampersand character (&)
  using for quick-access keys, 2-7, 2-218
APPENDMENUQQ (W*32, W*64), 2-6
  constants indicating the menu state, 2-6
  example of, 2-8
  ARC (W*32, W*64), 2-8
  example of, 2-9
  ARC_W (W*32, W*64), 2-8
Arcs
  drawing elliptical, 2-8
  function testing for endpoints of, 2-98
Argument list data structure (W*32)
  function allocating, 2-11
  function deallocating, 2-12
  function invoking method with, 2-15
  subroutine adding an argument to, 2-10
  subroutine returning exception information, 2-12
Array descriptor
  subroutine creating in memory, 2-83
Array elements
  function performing binary search for, 2-21
Arrays
  function performing binary search for element of, 2-21
  function returning codepage in, 2-220
  function returning file information in, 2-91
  function returning language and country combinations in, 2-221
  portability routines for, 1-6
  subroutine performing quick sort on, 2-362
  subroutine sorting one-dimensional, 2-445
AUTO routines (W*32), 1-23
  AUTOAddArg, 2-10
  AUTOAllocateInvokeArgs, 2-11
  AUTODEallocateInvokeArgs, 2-12
  AUTOGetExceptInfo, 2-12
  AUTOGetProperty, 2-13
AUTOGetPropertyByID, 2-14
AUTOGetPropertyInvokeArgs, 2-15
AUTOInvoke, 2-15
AUTOSetProperty, 2-16
AUTOSetPropertyById, 2-17
AUTOSetPropertyInvokeArgs, 2-17
subsection to call before using, 2-41
Overview of, 1-23
table of, 1-24
USE statement for, 1-23
AUTOAddArg (W*32), 2-10
  constants indicating the intended use of the argument, 2-10
  constants indicating the variant type of the argument, 2-11
AutoAddArg (W*32)
  example of, 2-36
AUTOAllocateInvokeArgs (W*32), 2-11
  example of, 2-36
AUTODeallocateInvokeArgs (W*32), 2-12
AutoDeallocateInvokeArgs (W*32)
  example of, 2-36
AUTOGetExceptInfo (W*32), 2-12
AUTOGetProperty (W*32)
  function allocating data structure for, 2-11
AutoInvoke (W*32)
  example of, 2-36
Automation objects (W*32)
  AUTOAddArg subroutine, 2-10
  AUTOAllocateInvokeArgs function, 2-11
  AUTODeallocateInvokeArgs subroutine, 2-12
  AUTOGetException subroutine, 2-12
  AUTOGetProperty function, 2-13
  AUTOGetPropertyById function, 2-14
  AUTOGetPropertyInvokeArgs function, 2-15
  AUTOInvoke function, 2-15
  AUTOSetProperty function, 2-16
  AUTOSetPropertyById function, 2-17
  AUTOSetPropertyInvokeArgs function, 2-17
  method
    function invoking, 2-15
property value
  function using argument list structure to return, 2-15
  function using argument list structure to set, 2-17
  function using member ID to return, 2-14
  function using member ID to set, 2-17
  function using name or identifier to return, 2-13
  function using name or identifier to set, 2-16
  subroutine returning a pointer to interface of, 2-35
AUTOSetProperty (W*32), 2-16
AUTOSetPropertyById (W*32), 2-17
AUTOSetPropertyInvokeArgs (W*32), 2-17
B
BEEPQQ, 2-18
  example of, 2-18
BESJ0, 2-18
BESJ1, 2-18
BESJN, 2-18
Bessel functions
  functions computing double-precision values of, 2-44
  functions computing single-precision values of, 2-18
BESY0, 2-18
BESY1, 2-18
BESYN, 2-18
Bezier curves
  functions to draw, 2-247, 2-252
BIC, 2-19
  example of, 2-20
Binary raster operation constants, 2-435
BIS, 2-19
  example of, 2-20
BIT, 2-20
Bit-level
  function testing, 2-20
  subroutines performing set and clear, 2-19
Bitmap file
  function displaying image from, 2-196
BIT-WISE complement
  functions that return, 2-39
Index

BIT-WISE store
  function performing, 2-41
BSEARCHQQ, 2-21
  constants defining numeric arrays, 2-21
  example of, 2-22
  restriction for type checking, 2-22
Buffer for path
  constant allocating the largest possible, 2-95

C
C run-time exceptions
  function returning pointer to, 2-118
Callback routines (W*32, W*64)
  function assigning to dialog controls, 2-66
  function registering for menu items, 2-177
  function registering for mouse events, 2-370
  function unregistering for mouse events, 2-485
  predefined QuickWin, 2-6, 2-178, 2-216
CDFLOAT, 2-23
CHANGEDIRQQ, 2-23
  example of, 2-23
CHANGEDRIVEQQ, 2-24
  example of, 2-24
Character string
  function locating index of last occurrence of substring in, 2-378
  function locating last nonblank character in, 2-195
  function reading from keyboard, 2-143
  subroutine sending to screen (including blanks), 2-236, 2-238
  subroutine sending to screen (special fonts), 2-236
Character-based text
  routines displaying (W*32, W*64), 1-20
Characters
  function returning next available, 2-76, 2-101
  function writing to file, 2-89
CHDIR, 2-25
  example of, 2-25
  POSIX version of, 2-270
Child process
  POSIX function returning exit code for, 2-182
  POSIX function returning signal number of stopped, 2-183
  POSIX function returning signal number of terminated, 2-184
  POSIX function testing if exited (L*X), 2-358, 2-360
  POSIX function testing if stopped (L*X), 2-360
  POSIX subroutine creating (L*X), 2-288
  POSIX subroutine waiting for (L*X), 2-356
Child window
  function appending list of names to menu, 2-434
  function making active, 2-385
  function returning unit number of active, 2-97
  function setting properties of, 2-430
CHMOD, 2-26
  example of, 2-28
  POSIX version of, 2-271
  table of permission bits for, 2-26
Circles
  functions drawing, 2-73
CLEARSCREEN (W*32, W*64), 2-28
  constants indicating target area, 2-28
  example of, 2-29
CLEARSTATUSFPQQ, 2-29
  example of, 2-29
CLICKMENUQQ (W*32, W*64), 2-30
  constants indicating menu command selected, 2-30
Clip region
  subroutine setting, 2-389, 2-428
CLOCK, 2-31
  example of, 2-31
CLOCKX, 2-31
Codepage
  function setting current, 2-235
  function setting for current console, 2-234
  subroutine retrieving current, 2-227
Codepage number
  function returning for console codepage, 2-227
  function returning for system codepage, 2-227
Codepages
  function returning array of, 2-220
Color control or inquiry
  graphics routines for (W*32, W*64), 1-18
Color conversion
  QuickWin routines for (W*32, W*64), 1-17
Color index
function filling an area using, 2-78
function returning background, 2-99
function returning for graphics, 2-104
function returning for multiple pixels, 2-138
function returning for pixel, 2-135
function returning text, 2-144
function setting for graphics, 2-390
function setting for multiple pixels, 2-417
function setting for pixel, 2-414
function setting text, 2-421
Color RGB value
function returning graphics, 2-105
function setting current, 2-391
COM identifiers (W*32)
function testing for identical, 2-37
subroutine returning string for, 2-40
subroutine using class to return pointer to interface, 2-33, 2-34
subroutine using programmatic to return class, 2-32
subroutine using programmatic to return pointer to interface, 2-34, 2-35
COM library (W*32)
subroutine initializing, 2-36
subroutine uninitializing, 2-41
COM objects (W*32)
COMAddObjectReference subroutine, 2-32
COMCLSIDFromProgID subroutine, 2-32
COMCLSIDFromString subroutine, 2-33
COMCreateObjectByGUID subroutine, 2-33
COMCreateObjectByProgID subroutine, 2-34
COMGetActiveObjectByGUID subroutine, 2-34
COMGetActiveObjectByProgID subroutine, 2-35
COMGetFileObject subroutine, 2-35
COMInitialize subroutine, 2-36
COMIsEqualGUID function, 2-37
COMQueryInterface subroutine, 2-39
COMReleaseObject function, 2-40
COMStringFromGUID subroutine, 2-40
COMUninitialize subroutine, 2-41
example of, 2-36
COMGetActiveObjectByProgID subroutine, 2-35
COMGetFileObject subroutine, 2-36
COMInitialize subroutine, 2-36
COMIsEqualGUID function, 2-37
COMQueryInterface subroutine, 2-39
COMReleaseObject function, 2-40
COMStringFromGUID subroutine, 2-40
COMUninitialize subroutine, 2-41
function referring to interface, 2-32
COM routines (W*32), 1-23
COMAddObjectReference subroutine, 2-32
COMCLSIDFromProgID subroutine, 2-32
COMCLSIDFromString subroutine, 2-33
COMCreateObjectByGUID subroutine, 2-33
COMCreateObjectByProgID subroutine, 2-34
COMGetActiveObjectByGUID subroutine, 2-34
COMGetActiveObjectByProgID subroutine, 2-35
COMGetFileObject subroutine, 2-35
COMInitialize subroutine, 2-36
COMIsEqualGUID function, 2-37
COMQueryInterface subroutine, 2-39
COMReleaseObject function, 2-40
COMStringFromGUID subroutine, 2-40
COMUninitialize subroutine, 2-41
function referring to interface, 2-32

Index-4
COMReleaseObject (W*32), 2-40
eexample of, 2-36
COMStringFromGUID (W*32), 2-40
eexample of string, 2-41
COMUnInitialize (W*32)
eexample of, 2-36
COMUninitialize (W*32), 2-41
Console codepage
function returning number for, 2-227
Console keystrokes
function checking for, 2-243
Constants
binary raster operation, 2-435
defining interaction between Windows images, 2-258
defining interrupt signals, 2-442
defining key states, 2-487
defining numeric arrays, 2-21, 2-446
defining signals, 2-364, 2-440
defining Windows access modes, 2-93, 2-475
defining Windows cursor shapes, 2-413
defining Windows direction keys, 2-240
defining Windows mouse events, 2-370
determining properties of Windows message box, 2-214
indicating display for Windows dialog box, 2-58
indicating floating-point control words, 2-107, 2-393
indicating menu state, 2-6
indicating Windows cursor state, 2-49
indicating Windows exit behavior, 2-119
indicating Windows menu command, 2-30
indicating Windows menu state, 2-177, 2-215
indicating Windows target area, 2-28
POSIX function testing, 2-307
POSIX subroutine returning value associated with, 2-276
ternary raster operation, 2-259
Control variables
functions setting value of dialog, 2-62
Control word
constants indicating floating-point, 2-107, 2-393
defaults for floating-point, 2-108
subroutine returning floating-point, 2-107, 2-383
subroutine setting floating-point, 2-190, 2-393
Conversion
from COMPLEX(4) to double-precision real, 2-23
from integer to double-precision real, 2-48
from integer to single-precision real, 2-171
from INTEGER(2) to INTEGER(4), 2-197
from INTEGER(4) to double-precision real, 2-164
from INTEGER(4) to INTEGER(2), 2-179, 2-439
from integers to RGB color value, 2-377
from RGB color value to component values, 2-180
from system time to ASCII string, 2-31, 2-42
from viewport coordinates to physical coordinates, 2-133
from viewport coordinates to window coordinates, 2-154
from Windows unit number to handle, 2-128
Coordinates
graphics routines converting and setting (W*32, W*64), 1-19
subroutine converting from physical to viewport, 2-151
subroutine converting from viewport to physical, 2-133
subroutine returning for current graphics position, 2-109
Country
function setting current, 2-235
subroutine retrieving current, 2-227
CPU time
function returning elapsed, 2-72, 2-75
Critical errors
subroutine controlling prompt, 2-397
CSMG, 2-41
CTIME, 2-42
example of, 2-42
Currency string
function returning for current locale, 2-222
Current date
function returning, 2-43
subroutines returning, 2-43, 2-44, 2-111, 2-164
Current locale
function returning information about, 2-228
Cursor
function controlling display of, 2-49
function setting the shape of, 2-412
D

DATE, 2-43
  example of, 2-43

Date
  function returning current system, 2-43
  function returning for current locale, 2-223
  function returning Julian, 2-188
  function setting, 2-395
  subroutine returning current system, 2-43, 2-44
  subroutine returning system, 2-163
  subroutine unpacking a packed, 2-484
  subroutines returning, 2-111, 2-164

Date and time
  portability routines for, 1-4
  routine returning as ASCII string, 2-76
  subroutine packing values for, 2-238

Date and time format
  NLS functions returning (W*32, W*64), 1-10

DATE4, 2-44
DBESJ0, 2-44
  example of, 2-45
DBESJ1, 2-44
DBESJN, 2-44
DBESY0, 2-44
DBESY1, 2-44
DBESYN, 2-44
DCLOCK, 2-45
  example of, 2-45
DELDIRQQ, 2-46
DELETEMENUQQ (W*32, W*64), 2-46
  example of, 2-47
DELFFILESQQ, 2-47
  example of, 2-48
DFLOATI, 2-48
DFLOATJ, 2-48
DFLOATK, 2-48
Dialog boxes (W*32)
  assigning event handlers to controls in, 2-63
  deallocating memory associated with, 2-70
  function assigning callback routine to, 2-66
  function displaying modeless, 2-58
  function sending a message to a control, 2-60
  functions displaying, 2-56
  functions returning values associated with controls, 2-52
  functions testing messages for, 2-55
  functions to initialize, 2-53
  subroutine closing, 2-49
  subroutine setting messages for, 2-411
  subroutine setting title of, 2-69
  subroutine updating the display of, 2-50

Dialog control boxes (W*32)
  function sending a message to, 2-60

Dialog control variable (W*32)
  functions returning state of, 2-52
  functions setting state of, 2-62

Dialog controls (W*32)
  function assigning callback routine to, 2-66
  functions displaying selections, 2-56
  functions returning values associated with, 2-52

Dialog routines (W*32)
  DLGEXIT, 2-49
  DLGFLUSH, 2-50
  DLGGET, 2-52
  DLGGETCHAR, 2-52
  DLGGETINT, 2-52
  DLGGETLOG, 2-52
  DLGINIT, 2-53
  DLGINITWITHRESOURCEHANDLE, 2-53
  DLGISDLGMESSAGE, 2-55
  DLGISDLGMESSAGEWITHDLG, 2-55
  DLGMODAL, 2-56
  DLGMODALWITHPARENT, 2-56
  DLGMODELESS, 2-58
  DLGSENDCTRLMESSAGE, 2-60
  DLGSET, 2-62
  DLGSETCHAR, 2-62
  DLGSETCTRLEVENTHANDLER, 2-63
  DLGSETINT, 2-62
  DLGSETLOG, 2-62
  DLGSETRETURN, 2-65
  DLGSETSUB, 2-66
  DLGSETTITITLE, 2-69
  DLGUNINIT, 2-70
  overview of, 1-22
  table of, 1-22
  USE statement for, 1-22
Direction keys
  and INCHARQQ, 2-173
  function determining behavior of, 2-240

Directory
  function changing the default, 2-25
  function creating, 2-200
  function deleting, 2-46
  function returning full path of, 2-95
  function returning path of current working, 2-110
  function specifying current as default, 2-23
  POSIX subroutine changing the default, 2-270
  POSIX subroutine closing, 2-273
  POSIX subroutine creating a link to, 2-309
  POSIX subroutine opening, 2-316
  POSIX subroutine reading from, 2-321
  POSIX subroutine removing, 2-323
  POSIX subroutine removing entry from, 2-354
  POSIX subroutine rewinding, 2-323

Directory path
  function splitting into components, 2-447

DISPLAYCURSOR (W*32, W*64), 2-49
  constants indicating cursor state, 2-49

DLGEXIT (W*32), 2-49
  example of, 2-50

DLGFLUSH (W*32), 2-50
  when processing loop may be required, 2-51

DLGGET (W*32), 2-52, 2-65
  example of, 2-53, 2-66

DLGGETCHAR (W*32), 2-52

DLGGETINT (W*32), 2-52

DLGGETLOG (W*32), 2-52

DLGINIT (W*32), 2-53
  example of, 2-54, 2-55, 2-68

DLGINITWITHRESOURCEHANDLE (W*32), 2-53
  when to use, 2-54

DLGISDLGMEMESSAGE (W*32), 2-55, 2-59
  example of, 2-51, 2-55
  when to use DLGISDLGMEMESSAGEWITHDLG, 2-55

DLGISDLGMEMESSAGEWITHDLG (W*32), 2-55
  when to use, 2-55

DLGMODAL (W*32), 2-51, 2-56, 2-61
  default return value for, 2-57
  example of, 2-57, 2-68
  rules for determining parent window, 2-57
  subroutine setting return value of, 2-65
  when to use DLGMODALWITHPARENT, 2-57

DLGMODALWITHPARENT (W*32), 2-56
  when to use, 2-57

DLGMODELESS (W*32), 2-51, 2-55, 2-58, 2-61
  attribute of variable in, 2-58
  constants indicating display for dialog box, 2-58
  example of, 2-55, 2-60
  rules for determining parent window, 2-59

DLGSENDCTRLMESSAGE (W*32), 2-60
  example of, 2-61
  function to call before using, 2-61

DLGSET (W*32), 2-51, 2-62
  example of, 2-63

DLGSETCHAR (W*32), 2-62

DLGSETCTRLEVENTHANDLER (W*32), 2-63
  example of, 2-65
  function returning IDispatch pointer, 2-65

DLGSETINT (W*32), 2-62

DLGSETLOG (W*32), 2-62

DLGSETRETURN (W*32), 2-57, 2-65
  example of, 2-66

DLGSETSUB (W*32), 2-66
  example of, 2-55, 2-68
  interface for callback routine in, 2-67

DLGSETTITLE (W*32), 2-69
  example of, 2-69

DLGUNINIT (W*32), 2-70
  example of, 2-55, 2-70

DMOD
  See your language reference

DOUBLE PRECISION
  functions converting to, 2-48, 2-164

DRAND, 2-70
  example of, 2-71

DRANDM, 2-70

DRANSET, 2-72

Drive
  function returning available space on, 2-113
  function returning path of, 2-112
  function returning total size of, 2-113
function specifying current as default, 2-24
Drive control or inquiry
  portability routines for, 1-5
Drives
  function returning available, 2-115
DSHIFTL
  See your language reference
DSHIFTR
  See your language reference
DTIME, 2-72
  example of, 2-73

E
Elapsed time
  function causing a subroutine to run after, 2-4
  function converting, 2-42
  function returning, 2-45, 2-72, 2-75
Electromagnetic wave theory
  functions used in, 2-18, 2-44
ELLIPSE (W*32, W*64), 2-73
  constants indicating fill for, 2-73
  example of, 2-74, 2-129, 2-388
ELLIPSE_W (W*32, W*64), 2-73
  constants indicating fill for, 2-73
Ellipses
  functions drawing, 2-73
Elliptical arcs
  drawing, 2-8
Environment
  function cleaning up run-time, 2-86
  function initializing run-time, 2-87
Environment variables
  function finding file in path specified by, 2-77
  function returning value of, 2-116
  function scanning for, 2-381
  function setting value of, 2-396
  subroutine returning value of, 2-115
Errno names, 2-170
Error
  subroutine sending last detected to standard error stream, 2-244
Error codes, 2-170
Error handling
  portability routines for, 1-5
  run-time routines for, 1-26
Error numbers, 2-170
Errors
  functions returning most recent run-time, 2-130
  subroutine returning message for last detected, 2-96
ETIME, 2-75
  example of, 2-75
Exception flags
  function returning settings of floating-point, 2-86
  function setting floating-point, 2-87
  subroutine clearing in status word, 2-29
Execution
  subroutine delaying for a program, 2-445
  subroutine suspending for a process, 2-444
EXIT
  See your language reference
Exit behavior
  constants indicating (W*32, W*64), 2-119
  function returning QuickWin, 2-119
  function setting QuickWin, 2-399
Exit parameters
  function setting QuickWin, 2-399
External unit 5
  function returning next character from, 2-101
External unit 6
  function writing a character to, 2-257
External unit buffer
  subroutine flushing, 2-81

F
FDATE, 2-76
  example of, 2-77
FGETC, 2-76
  example of, 2-77
  POSIX version of, 2-286
Field component
  POSIX subroutine returning array values stored in, 2-263
  POSIX subroutine returning value stored in, 2-260
POSIX subroutine returning values stored in array element, 2-278
POSIX subroutine setting array element, 2-280
POSIX subroutine setting value of, 2-262
POSIX subroutine setting value of array, 2-264
Figure characteristics
  graphics routines for (W*32, W*64), 1-19
File access mode
  function setting, 2-400
File descriptor
  POSIX subroutine defining an action for (L*X), 2-274
  POSIX subroutine duplicating, 2-278
File directory control or inquiry
  portability routines for, 1-5
File management
  portability routines for, 1-6
  run-time routines for, 1-25
File path
  function splitting into components, 2-447
File position
  functions returning, 2-94, 2-141
Files
  function changing access mode of, 2-26
  function deleting, 2-47
  function finding specified, 2-77
  function performing flush of, 2-38
  function renaming, 2-375
  function repositioning, 2-90
  function returning full path of, 2-95
  function returning information about, 2-91, 2-120, 2-197, 2-474
  function returning next available character from, 2-76
  function setting modification time for, 2-401
  function testing access mode of, 2-3
  function using path to delete, 2-483
  function writing character to, 2-89
  functions returning character to, 2-89
  functions returning character to, 2-95
  functions returning current position of, 2-94, 2-141
  POSIX function testing for block special, 2-306
  POSIX function testing for character, 2-306
  POSIX function testing for directory, 2-307
  POSIX function testing for regular, 2-308
  POSIX function testing for special FIFO, 2-308
  POSIX subroutine changing access mode of, 2-271
  POSIX subroutine changing the owner and group of, 2-272
  POSIX subroutine closing, 2-273
  POSIX subroutine creating a FIFO, 2-313
  POSIX subroutine creating a link to, 2-309
  POSIX subroutine creating new or rewriting, 2-276
  POSIX subroutine determining accessibility of, 2-265
  POSIX subroutine flushing, 2-286
  POSIX subroutine opening, 2-214
  POSIX subroutine positioning, 2-311
  POSIX subroutine reading from, 2-321
  POSIX subroutine returning configuration value, 2-289, 2-317
  POSIX subroutine setting access times for, 2-355
  POSIX subroutine writing to, 2-360
Fill mask
  function filling an area using, 2-78, 2-80
  subroutine setting to new pattern, 2-402
Fill shapes
  subroutine returning pattern used to, 2-123
FINDFILEQQ, 2-77
  example of, 2-78
Floating-point control word
  subroutine returning, 2-383
  subroutine setting, 2-190, 2-393
Floating-point exception flags
  function returning settings of, 2-86
  function setting, 2-87
Floating-point inquiry and control
  portability routines for, 1-6
  run-time routine for, 1-26
Floating-point status word
  subroutine clearing exception flags in, 2-29
  subroutines returning, 2-141, 2-473
FLOODFILL (W*32, W*64), 2-78
  example of, 2-79
FLOODFILL_W (W*32, W*64), 2-78
FLOODFILLRGB (W*32, W*64), 2-80
  example of, 2-81
FLOODFILLRGB_W (W*32, W*64), 2-80
FLUSH, 2-81
Focus
  function determining which window has, 2-176
  function setting, 2-82
FOCUSQQ (W*32, W*64), 2-82
related to SETACTIVEQQ, 2-82

Font characteristics
function returning, 2-124

Font-based characters
routines displaying (W*32, W*64), 1-21

Font-related library functions, 2-124, 2-126, 2-174, 2-236, 2-405
effect of, 2-175

Fonts
function initializing, 2-174
function returning characteristics of, 2-124
function returning orientation of text for, 2-127
function returning size of text for, 2-126
function setting for OUTGTEXT, 2-405
function setting orientation angle for text, 2-408

FOR_DESCRIPTOR_ASSIGN (W*32, W*64), 2-83
effect of, 2-84

FOR_GET_FPE, 2-86
effect of, 2-86

FOR_SET_FPE, 2-87
effect of, 2-87

FOR_SET_REENTRANCY, 2-88
effect of, 2-88

Formatting
NLS routines for (W*32, W*64), 1-8

GETACTIVEQQ (W*32, W*64), 2-97

GETARG
See your language reference

GETBKCOLOR (W*32, W*64), 2-99
effect of, 2-100

GETBKCOLORRGB (W*32, W*64), 2-100
effect of, 2-101

GETC, 2-101
effect of, 2-102
POSIX version of, 2-295

GETCHARQQ, 2-102
and PASSDIRKEYSQQ, 2-103
and PEEKCHARQQ, 2-103
effect of, 2-104

GETCOLOR (W*32, W*64), 2-104
and FLOODFILL, 2-79

effect of, 2-105

GETCOLORRGB (W*32, W*64), 2-105
and FLOODFILLRGB, 2-80

effect of, 2-81, 2-106

GET_CONTROL_FPQQ, 2-107
 constants defining control word, 2-107
effect of, 2-108

GETCURRENTPOSITION (W*32, W*64), 2-109
effect of, 2-110

GETCURRENTPOSITION_W (W*32, W*64), 2-109
effect of, 2-110

GETCWD, 2-110
POSIX version of, 2-295

GETDAT, 2-111
effect of, 2-111

FTELL, 2-94
POSIX version of, 2-293

FTELL8, 2-94

FULLPATHQQ, 2-95
effect of, 2-95
GETDRIVEDIRQQ, 2-112
  example of, 2-113
GETDRIVESIZEQQ, 2-113
  example of, 2-114
GETDRIVESQQ, 2-115
  example of, 2-114
GETENV, 2-115
  example of, 2-116
  POSIX version of, 2-296
GETENVQQ, 2-116
  example of, 2-117
GETEXCEPTIONPTRSQQ (W*32, W*64), 2-118
  example of, 2-118
GETEXITQQ (W*32, W*64), 2-119
  example of, 2-119
GETFILEINFQQ, 2-120
  constants indicating handle for, 2-121
  example of, 2-122
GETFILLMASK (W*32, W*64), 2-123
  example of, 2-124
GETFONTINFO (W*32, W*64), 2-124
  example of, 2-125
GETGID, 2-126
  example of, 2-126
  POSIX version of, 2-297
GETGTEXTTEXTENT (W*32, W*64), 2-126
  example of, 2-127
GETGTEXTROTATION (W*32, W*64), 2-127
  example of, 2-128
GETHWNDQQ (W*32, W*64), 2-128
GETIMAGE (W*32, W*64), 2-129
  example of, 2-129
  function returning memory needed for, 2-172
GETIMAGE_W (W*32, W*64), 2-129
GETLASTERROR, 2-130
GETLASTERRORQQ, 2-130
GETLINESTYLE (W*32, W*64), 2-132
  example of, 2-133
GETLOG, 2-133
  example of, 2-133
  POSIX version of, 2-302
GETPHYSCOORD (W*32, W*64), 2-133
  example of, 2-134
GETPID, 2-135
  example of, 2-135
  POSIX version of, 2-302
GETPIXEL (W*32, W*64), 2-135
GETPIXEL_W (W*32, W*64), 2-135
GETPIXELRGB (W*32, W*64), 2-136
  example of, 2-137
GETPIXELRGB_W (W*32, W*64), 2-136
GETPIXELS (W*32, W*64), 2-138
GETPIXELSRGB (W*32, W*64), 2-139
  example of, 2-140
GETPOS, 2-141
GETPOSI8, 2-141
GETSTATUSFPQQ, 2-141
  example of, 2-143
GETSTRQQ, 2-143
  example of, 2-143
GETTEXTCOLOR (W*32, W*64), 2-144
GETTEXTCOLORRGB (W*32, W*64), 2-145
  example of, 2-146
GETTEXTPOSITION (W*32, W*64), 2-147
  example of, 2-147
GETTEXTWINDOW (W*32, W*64), 2-148, 2-154
  example of, 2-134, 2-148
GETTIM, 2-149
  example of, 2-111
GETTIMEOFDAY, 2-149
GETUID, 2-150
  example of, 2-150
  POSIX version of, 2-306
GETUNITQQ (W*32, W*64), 2-150
GETVIEWCOORD (W*32, W*64), 2-151
  example of, 2-134
GETVIEWCOORD_W (W*32, W*64), 2-151
GETWINDOWCONFIG (W*32, W*64), 2-152
and SETWINDOWCONFIG, 2-153
  example of, 2-154
GETWITEMODE (W*32, W*64), 2-155
  example of, 2-156
  values for write mode, 2-155

Index-11
GETWSIZEQQ (W*32, W*64), 2-156
GETTIME, 2-157
example of, 2-158
Graphics
routines to draw (W*32, W*64), 1-19
Graphics output
function returning background color index for, 2-99
function returning background RGB color for, 2-100
function setting background color index for, 2-386
function setting background RGB color for, 2-387
subroutine limiting to part of screen, 2-389
Graphics position
routines changing, 2-109
subroutine moving to a specified point, 2-219
subroutine returning coordinates for current, 2-109
Graphics routines (W*32, W*64), 1-16
ARC and ARC_W, 2-8
CLEARSSCREEN, 2-28
converting and setting coordinates, 1-19
DISPLAYCURRENT, 2-49
displaying character-based text, 1-20
displaying font-based characters, 1-21
ELLIPSE and ELLIPSE_W, 2-73
FLOODFILL and FLOODFILL_W, 2-78
FLOODFILLRGB and FLOODFILLRGB_W, 2-80
for color control or inquiry, 1-18
for figure characteristics, 1-19
function returning status for, 2-159
GETARCCINFO, 2-98
GETBKCOLOR, 2-99
GETBKCOLORRGB, 2-100
GETCOLOR, 2-104
GETCOLORRGB, 2-105
GETCURRENTPOSITION, 2-109
GETCURRENTPOSITION_W, 2-109
GETFILLMASK, 2-123
GETFONTEXTINFO, 2-124
GETGTEXXTENT, 2-126
GETGTEXXTROTATION, 2-127
GETIMAGE, 2-129
GETIMAGE_W, 2-129
GETLINESTYLE, 2-132
GETPHYSCOORD, 2-133
GETPIXEL, 2-135
GETPIXEL_W, 2-135
GETPIXELRGB, 2-136
GETPIXELRGB_W, 2-136
GETPIXELS, 2-138
GETPIXELSRGB, 2-139
GETTEXTCOLOR, 2-144
GETTEXTCOLORRGB, 2-144
GETTEXTWINDOW, 2-145
GETTEXTWINDOW, 2-148, 2-154
GETVIEWCOORD, 2-151
GETVIEWCOORD_W, 2-151
GETWRITEMODE, 2-155
GRSTATUS, 2-159
IMAGESIZE, 2-172
IMAGESIZE_W, 2-172
INITIALIZEFONTS, 2-172
LINETO, 2-191
LINETO_W, 2-191
LINETOAR, 2-192
LINETOAREX, 2-193
LOADIMAGE, 2-196
LOADIMAGE_W, 2-196
MOVETO, 2-219
MOVETO_W, 2-219
OUTGTEXT, 2-236
OUTTEXT, 2-238
overview of, 1-16
PIE, 2-245
PIE_W, 2-245
POLYBEZIER, 2-247
POLYBEZIER_W, 2-247
POLYBEZIERTO, 2-252
POLYBEZIERTO_W, 2-252
POLYGON, 2-253
POLYGON_W, 2-253
POLYLINEQ, 2-256
PUTIMAGE, 2-258
PUTIMAGE_W, 2-258
RECTANGLE, 2-368
RECTANGLE_W, 2-368
REMAPALLPALETTERGB, 2-371
REMAPPALETTERGB, 2-373
SAVEIMAGE, 2-380
SAVEIMAGE_W, 2-380
SCROLLTEXTWINDOW, 2-382
SETBKCOLOR, 2-386
SETBKCOLORRGB, 2-387
SETCLIPRGN, 2-389
SETCOLOR, 2-390
SETCOLORRGB, 2-391
SETFILLMASK, 2-402
SETPFONT, 2-405
SETGTEXTROTATION, 2-408
SETLINESTYLE, 2-409
SETPIXEL, 2-414
SETPIXEL_W, 2-414
SETPIXELRGB, 2-416
SETPIXELRGB_W, 2-416
SETPIXELS, 2-417
SETPIXELSRGB, 2-419
SETEXTCOLOR, 2-421
SETEXTCOLORRGB, 2-422
SETEXTCURSOR, 2-423
SETEXTPOSITION, 2-425
SETEXTWINDOW, 2-426
SETVIEWORG, 2-427
SETVIEWPORT, 2-428
SETWINDOW, 2-429
SETWRITEMODE, 2-435

GRSTATUS (W*32, W*64), 2-159

H
Handle
function converting unit number into, 2-128
function returning unit number corresponding to, 2-150
POSIX subroutine returning, 2-305
Handlers
function establishing for IEEE exceptions, 2-168
Help
function specifying text for About box, 2-2
Host computer name
function returning, 2-162
HOSTNAM, 2-162
table of, 1-18
to display character-based text, 1-19
to draw graphics, 1-19
to load and save images to files, 1-21
to transfer images in memory, 1-21
USE statement for, 1-16
WRAPON, 2-488

Graphics viewport
subroutine redefining, 2-428
Greenwich mean time
function returning seconds and microseconds since, 2-149
function returning seconds since, 2-379
subroutine returning, 2-157

Group ID
function returning, 2-126
POSIX subroutine returning information on, 2-298
POSIX subroutine returning process (L*X), 2-302
POSIX subroutine returning supplementary, 2-299
POSIX subroutine setting process (L*X), 2-325, 2-326, 2-327

Group name
POSIX subroutine returning information on, 2-299

I
I/O buffers
flushing and closing, 2-2
IARG
See your language reference
IARGC
See your language reference
IDATE, 2-163
table of, 1-18
to display character-based text, 1-19
to draw graphics, 1-19
to load and save images to files, 1-21
to transfer images in memory, 1-21
USE statement for, 1-16
WRAPON, 2-488

IEEE* exceptions
function clearing status of, 2-165
function establishing a handler for, 2-168
function getting or setting status of, 2-165
IEEE* flags
function clearing, 2-165
function getting or setting, 2-165
IEEE* functionality
portability routines for, 1-6
IEEE_FLAGS, 2-165
table of, 1-18
to display character-based text, 1-19
to draw graphics, 1-19
to load and save images to files, 1-21
to transfer images in memory, 1-21
USE statement for, 1-16
WRAPON, 2-488

Index-13
examples of, 2-167
  math exception flags for, 2-165
  mode values for, 2-165
  precision flags for, 2-165
IEEE_HANDLER, 2-168
  example of, 2-169
IERRNO, 2-170
  example of, 2-171
  subroutine returning message for last error detected by, 2-96
IFLOATI, 2-171
IFLOATJ, 2-171
Images
  function displaying from bitmap file, 2-196
  function returning storage size of, 2-172
  function saving into Windows bitmap file, 2-380
  routines to load and save (W*32, W*64), 1-21
  routines to transfer in memory (W*32, W*64), 1-21
  transferring from memory to screen, 2-258
IMAGESIZE (W*32, W*64), 2-172
  example of, 2-129
IMAGESIZE_W (W*32, W*64), 2-172
IMOD
  See your language reference
INCHARQQ (W*32, W*64), 2-173
  example of, 2-173
  NLS version of, 2-204
Index for last occurrence of substring
  function locating, 2-378
Information retrieval
  portability routines for, 1-2
INITIALIZEFONTS (W*32, W*64), 2-174
  and SETFONT, 2-175
  example of, 2-175
INITIALSETTINGS (W*32, W*64), 2-175
INMAX, 2-176
Input and output
  portability routines for, 1-3
INQFOCUSQQ (W*32, W*64), 2-176
INSERTMENUQQ (W*32, W*64), 2-177
  example of, 2-179
INTC, 2-179
Integers
  converting to RGB values, 2-377
  function converting KIND=2 to KIND=4, 2-197
  function converting KIND=4 to KIND=2, 2-179, 2-439
  function converting to single-precision type, 2-171
  function performing bit-level test for, 2-20
  function returning maximum positive value for, 2-176
  functions converting to double-precision type, 2-48, 2-164
  POSIX subroutine comparing, 2-353
  subroutine performing bit-level set and clear for, 2-19
INTEGERTORGB (W*32, W*64), 2-180
  example of, 2-181
Interrupt signal
  registering a function to call for, 2-442
Interrupt signal handling
  function controlling, 2-439
Intrinsic procedures
  See your language reference
IOFOCUS specifier (W*32, W*64)
  in OPEN statements, 2-82
IPXFARGC, 2-181
IPXFCONST, 2-181
IPXFLENTTRIM, 2-182
IPXFLENTRIM, 2-182
  and PXFWEXITSTATUS (L*X), 2-182
  and PXFWIFEXITED, 2-183
  example of, 2-183
IPXFWSTOPSIG (L*X), 2-183
  and PXFWIFSTOPPED, 2-184
IPXFWTERMSIG (L*X), 2-184
  and PXFWIFSIGNALED, 2-184
IRAND, 2-184, 2-472
  example of, 2-185
IRANDM, 2-184
IRANGET, 2-185
IRANSET, 2-186
ISATTY, 2-186
ITIME, 2-187
  example of, 2-187
Index

J
JABS, 2-187
Japan industry standard characters, 2-205
JDATE, 2-188
   example of, 2-188
JDATE4, 2-188
JIS characters
   converting to JMS, 2-205
JMS characters
   converting to JIS, 2-205
Julian date
   function returning, 2-188

K
Keyboard character
   function returning ASCII value of, 2-173
Keyboards
   run-time routines for, 1-25
Keystroke
   function checking for, 2-243
   function returning ASCII value of, 2-173
   function returning next, 2-102
KILL, 2-189
   example of, 2-190
   POSIX version of, 2-309

L
Labels
   platform, xxiv
   See also Building Applications
Language and country combinations
   function returning array of, 2-221
LCWRRQ, 2-190
   example of, 2-190
LEADZ
   See your language reference
Library routines
   AUTO (W*32), 1-23
   COM (W*32), 1-23
   dialog (W*32), 1-22
   Graphics (W*32, W*64), 1-16
   miscellaneous run-time, 1-25
   NLS (W*32, W*64), 1-8
   portability, 1-2
   POSIX, 1-11
   QuickWin (W*32, W*64), 1-16
Line style
   function returning, 2-132
   subroutine setting, 2-409
Lines
   function drawing, 2-191
   function drawing between arrays, 2-192, 2-193
   function drawing within an array, 2-256
LINETO (W*32, W*64), 2-191, 2-435
   example of, 2-140, 2-192
LINETO_W (W*32, W*64), 2-191
LINETOAR (W*32, W*64), 2-192
   example of, 2-193
LINETOAREX (W*32, W*64), 2-193
   example of, 2-195
LNBLNK, 2-195
   example of, 2-196
LOADIMAGE (W*32, W*64), 2-196
LOADIMAGE_W (W*32, W*64), 2-196
LOC intrinsic function
   using with BSEARCHQQ, 2-22
Locale
   function returning currency string for current, 2-222
   function returning date for current, 2-223
   function returning information about current, 2-228
   function returning number string for current, 2-224
   function returning time for current, 2-225
Locale setting and inquiry
   NLS routines for (W*32, W*64), 1-8
Logical .NOT.
   functions returning, 2-39
Logical unit number
   function testing whether it’s a terminal, 2-186
Login name
   subroutine returning, 2-133
LONG, 2-197
LSTAT, 2-197
   example of, 2-198
LTIME, 2-198
eexample of, 2-199

M
MAKEDIRQQ, 2-200
eexample of, 2-200
POIX version of, 2-312
Mask
POIX subroutine setting, 2-353
subroutine setting new pattern for fill, 2-402
Math exception flags
for IEEE_FLAGS, 2-165
function establishing a handler for, 2-168
function getting or setting, 2-165
MBCharLen (W*32, W*64), 2-201
MBCurMax (W*32, W*64), 2-204
MB정Index (W*32, W*64), 2-205
MBJISToJMS (W*32, W*64), 2-205
MBJMSToJIS (W*32, W*64), 2-205
MBLead (W*32, W*64), 2-206
MBLen (W*32, W*64), 2-207
MBLen_Trim (W*32, W*64), 2-208
MBLEQ (W*32, W*64), 2-208
flags for, 2-209
MBLGE (W*32, W*64), 2-208
flags for, 2-209
MBLGT (W*32, W*64), 2-208
flags for, 2-209
MBLLE (W*32, W*64), 2-208
flags for, 2-209
MBLLT (W*32, W*64), 2-208
flags for, 2-209
MBLNE (W*32, W*64), 2-208
flags for, 2-209
MBNext (W*32, W*64), 2-210
MBPrev (W*32, W*64), 2-211
MBSCAN (W*32, W*64), 2-211
MBStrLead (W*32, W*64), 2-212
MBVERIFY (W*32, W*64), 2-213
Memory assignment
run-time routine for, 1-26
Menu command
function simulating selection of, 2-30
Menu items
definition order of, 2-179
function changing callback routine of, 2-216
function changing text string of, 2-218
function deleting from QuickWin, 2-46
function modifying the state of, 2-215
function to insert, 2-177
Menu state
constants indicating, 2-6, 2-177, 2-215
Menus
definition order of, 2-179
function appending child window list to, 2-434
function appending item to, 2-6
function inserting item in, 2-177
function setting top-level for append list, 2-434
Message box
constants determining objects and properties of, 2-214
function displaying, 2-213
function specifying text for, 2-2
MESSAGEBOXQQ (W*32, W*64), 2-213
example of, 2-215
MODIFYMENUFLAGSQQ (W*32, W*64), 2-215
example of, 2-216
MODIFYMENUROUTINEQQ (W*32, W*64), 2-216
predefined routines for, 2-217
MODIFYMENUSTRINGQQ (W*32, W*64), 2-218
example of, 2-218
Mouse cursor
  function setting the shape of, 2-412

Mouse events
  constants defining, 2-370
  function registering callback routine for, 2-370
  function unregistering callback routine for, 2-485
  function waiting for, 2-486

Mouse input
  function waiting for, 2-486

MOVETO (W*32, W*64), 2-219
  example of, 2-110, 2-140, 2-220, 2-237, 2-393

MOVETO_W (W*32, W*64), 2-219

Multibyte characters
  function equivalent to INCHARQQ, 2-204
  function equivalent to INDEX, 2-205
  function equivalent to SCAN, 2-211
  function equivalent to VERIFY, 2-213
  function performing context-sensitive test for, 2-212
  function returning first, 2-206
  function returning length for codepage, 2-204
  function returning number and character, 2-204
  functions comparing strings of, 2-208

Multibyte-character string
  function converting to Unicode, 2-201
  function returning length (including blanks), 2-207
  function returning length (no blanks), 2-208
  function returning length of first character in, 2-201
  function returning position of next character in, 2-210
  function returning position of previous character in, 2-211

N

NARGS
  See your language reference

National Language Support functions (W*32, W*64), 1-8
  See also NLS functions

NLS date and time format (W*32, W*64), 1-10

NLS functions
  table of, 1-8

NLS functions (W*32, W*64), 1-8
  date and time format, 1-10
  MBCharLen, 2-201
  MBConvertMBToUnicode, 2-201
  MBConvertUnicodeToMB, 2-202
  MBCurMax, 2-204
  MBINCHARQQ, 2-204
  MBINDEX, 2-205
  MBJISToJMS, 2-205
  MBJMSToJIS, 2-205
  MBLLead, 2-206
  MBLen, 2-207
  MBLen_Trim, 2-208
  MBLEQ, 2-208
  MBLGE, 2-208
  MBLTE, 2-208
  MBLLT, 2-208
  MBLINE, 2-208
  MBNext, 2-210
  MBPrev, 2-211
  MBSCAN, 2-211
  MBStrLead, 2-212
  MBVERIFY, 2-213
  NLSEnumCodepages, 2-220
  NLSEnumLocales, 2-221
  NLSFormatCurrency, 2-222
  NLSFormatDate, 2-223
  NLSFormatNumber, 2-224
  NLSFormatTime, 2-225
  NLSGetEnvironmentCodepage, 2-227
  NLSGetLocaleInfo, 2-228
  NLSSetEnvironmentCodepage, 2-234
  NLSSetLocale, 2-235
  overview of, 1-8
  USE statement for, 1-8

NLS language (W*32, W*64)
  function setting current, 2-235
  subroutine retrieving current, 2-227

NLS locale parameters (W*32, W*64)
  table of, 2-229

NLS routines (W*32, W*64)
  for formatting, 1-8
  for locale setting and inquiry, 1-8
  for MBCS conversion, 1-9
  for MBCS inquiry, 1-8
  MBCS Fortran equivalents, 1-9

NLS subroutines (W*32, W*64)
  NLSSetLocale, 2-235
NLSSI parameters (W*32, W*64)
  table of, 2-229
NLSEnumCodepages (W*32, W*64), 2-220
NLSEnumLocales (W*32, W*64), 2-221
NLSFormatCurrency (W*32, W*64), 2-222
  example of, 2-223
  flags for, 2-222
NLSFormatDate (W*32, W*64), 2-223
  example of, 2-224
  flags for, 2-223
NLSFormatNumber (W*32, W*64), 2-224
  example of, 2-225
  flags for, 2-225
NLSFormatTime (W*32, W*64), 2-225
  example of, 2-226
  flags for, 2-226
NLSGetEnvironmentCodepage (W*32, W*64), 2-227
  flags for, 2-227
NLSGetLocale (W*32, W*64), 2-227
  example of, 2-228
NLSGetLocaleInfo (W*32, W*64), 2-228
  parameter arguments for, 2-229
NLSSetEnvironmentCodepage (W*32, W*64), 2-234
NLSSetLocale (W*32, W*64), 2-235
  predefined values for codepages, 2-235
NUL predefined QuickWin routine, 2-6
NUMARG
  See your language reference
Number string
  function returning for current locale, 2-224
Numeric conversion
  portability routines for, 1-2
Numeric values
  portability routines for, 1-2

O
Object interface (W*32)
  function adding a reference to, 2-32
  function releasing, 2-40
  subroutine returning a pointer to, 2-33, 2-39
OPEN statements
  IOFOCUS specifier in (W*32, W*64), 2-82
OUTGTEXT (W*32, W*64), 2-236
  and INITIALIZEFONTS, 2-237
  example of, 2-237
  related routines, 2-126, 2-127, 2-405, 2-408
OUTTEXT (W*32, W*64), 2-238, 2-488
  example of, 2-238, 2-382

P
PACKTIMEQQ, 2-238
  example of, 2-239
Page keys
  function determining behavior of, 2-240
PASSDIRKEYSQQ
  and GETCHARQQ, 2-103
PASSDIRKEYSQQ (W*32, W*64), 2-240
  constants defining actions for, 2-240
  example of, 2-241
Password information
  POSIX subroutine returning (L*X), 2-303, 2-304
Path
  constant allocating the largest possible length for,
  2-95
  function returning working directory, 2-112
  function splitting into components, 2-447
Pathname
  POSIX subroutine generating a terminal (L*X), 2-277
Pattern used to fill shapes
  subroutine returning, 2-123
PEEKCHARQQ, 2-243
  and GETCHARQQ, 2-103
  example of, 2-244
PERROR, 2-244
  example of, 2-245
Physical coordinates
  subroutine converting from viewport coordinates,
  2-133
  subroutine converting to viewport coordinates, 2-151
PIE (W*32, W*64), 2-245
  example of, 2-247
Pie graphic
  function testing for endpoints of, 2-98
PIE_W (W*32, W*64), 2-245
Pie-shaped wedge
  function to draw, 2-245

Pixel
  function returning color index for, 2-135
  function returning RGB color value for, 2-136
  function setting color index for, 2-414
  function setting RGB color value for, 2-416

Pixels
  function returning color index for multiple, 2-138
  function returning RGB color value for multiple, 2-139
  function setting color index for multiple, 2-417
  function setting RGB color value for multiple, 2-419

Platform
  description of, xxiv
  labels, xxiv
  POLYBEZIER (W*32, W*64), 2-247
    example of, 2-248
  POLYBEZIER_W (W*32, W*64), 2-247
    example of, 2-248
  POLYBEZIERTO (W*32, W*64), 2-252
    example of, 2-248
  POLYBEZIERTO_W (W*32, W*64), 2-252
    example of, 2-248
  POLYGON (W*32, W*64), 2-253, 2-435
    example of, 2-255
  POLYGON_W (W*32, W*64), 2-253

Polygons
  function to draw, 2-253
  POLYLINEQQ (W*32, W*64), 2-256
    example of, 2-257

POPCNT
  See your language reference

POPPAR
  See your language reference

Portability routines
  ABORT, 2-2
  ACCESS, 2-3
  ALARM, 2-4
  BEEPQQ, 2-18
  BESIN, 2-18
  BESYN, 2-18
  BIC, 2-19
  BIS, 2-19
  BIT, 2-20
  BSEARCHQQ, 2-21
  CDFLOAT, 2-23
  CHANGEDIRQQ, 2-23
  CHANGEDRIVEQQ, 2-24
  CHTER, 2-25
  CHMOD, 2-26
  CLEARSTATUSFPQQ, 2-29
  CLOCK, 2-31
  CLOCKX, 2-31
  COMPLINT, 2-39
  COMPLLOG, 2-39
  COMPLREAL, 2-39
  CSMG, 2-41
  CTIME, 2-42
  DATE, 2-43
  DATE4, 2-44
  DBESIN, 2-44
  DBESYN, 2-44
  DCLOCK, 2-45
  DELDIRQQ, 2-46
  DELFILESQQ, 2-47
  DFLOATI, 2-48
  DFLOATJ, 2-48
  DFLOATK, 2-48
  DRAND, 2-70
  DRANDM, 2-70
  DRANSET, 2-72
  DTIME, 2-72
  ETIME, 2-75
  FDATE, 2-76
  FGETC, 2-76
  FINDFILEQQ, 2-77
  FLUSH, 2-81
  for arrays, 1-6
  for date and time, 1-4
  for error handling, 1-5
  for file management, 1-6
  for floating-point inquiry and control, 1-6
  for IEEE functionality, 1-6
  for information retrieval, 1-2
  for input and output, 1-3
  for numeric values and conversion, 1-2
  for process control, 1-2
  for program call and control, 1-5
for serial port I/O, 1-6
for speakers, 1-5
for system, drive, or directory control and inquiry, 1-5
FPUTC, 2-89
FSEEK, 2-90
FSTAT, 2-91
FTELL, 2-94
FTELL8, 2-94
FULLPATHQQ, 2-95
GETC, 2-101
GETCONTROLFPQQ, 2-107
GETCWD, 2-110
GETDAT, 2-111
GETDRIVEDIRQQ, 2-112
GETDRIVESIZEQQ, 2-113
GETDRIVESQQ, 2-115
GETENV, 2-115
GETENVQQ, 2-116
GETFILEINFOQQ, 2-120
GETGID, 2-126
GETLASTERROR, 2-120
GETLASTERRORQQ, 2-130
GETLOG, 2-133
GETPID, 2-135
GETPOS, 2-141
GETPOS8, 2-141
GETSTATUSFPQQ, 2-141
GETTIM, 2-149
GETTIMEOFDAY, 2-149
GETUID, 2-150
GMTIME, 2-157
HOSTNAM, 2-162
HOSTNM, 2-162
IDATE, 2-163
IDATE4, 2-164
IDFLOAT, 2-164
IEEE_FLAGS, 2-165
IEEE_HANDLER, 2-168
IERRNO, 2-170
IFLOATI, 2-171
IFLOATJ, 2-171
INMAX, 2-176
INTC, 2-179
IRAND, 2-184
IRANDDM, 2-184
IRANGET, 2-185
IRANSET, 2-186
ISATTY, 2-186
ITIME, 2-187
JABS, 2-187
JDATE, 2-188
JDATE4, 2-188
KILL, 2-189
LCWRQQ, 2-190
LNLNK, 2-195
LONG, 2-197
LSTAT, 2-197
LTIME, 2-198
MAKEDIRQQ, 2-200
overview of, 1-2
PACKTIMEQQ, 2-238
PUTC, 2-257
QRANSET, 2-361
QSORT, 2-362
RAISEQQ, 2-363
RAND, 2-364
RANDOM, 2-364, 2-366
RANF, 2-367
RANGET, 2-367
RENAME, 2-375
RENAMEFILEQQ, 2-375
RINDEX, 2-378
RTC, 2-379
RUNQQ, 2-379
SCANENV, 2-381
SCWRQQ, 2-383
SECNDS, 2-384
SEED, 2-384
SETCONTROLFPQQ, 2-393
SETDAT, 2-395
SETENVQQ, 2-396
SETERRORMODEQQ, 2-397
SETFILEACCESSQQ, 2-400
SETFILETIMEQQ, 2-401
SETTIM, 2-427
SHORT, 2-439
SIGNAL, 2-439
SIGNALQQ, 2-442
SLEEP, 2-444
SLEEPQQ, 2-445
SORTQQ, 2-445
SPLITPATHQQ, 2-447
SPORT_CANCEL_IO (W*32, W*64), 2-448
SPORT_CONNECT (W*32, W*64), 2-449
SPORT_CONNECT_EX (W*32, W*64), 2-450
SPORT_GET_HANDLE (W*32, W*64), 2-452
SPORT_GET_STATE (W*32, W*64), 2-453
SPORT_GET_STATE_EX (W*32, W*64), 2-454
SPORT_GET_TIMEOUTS (W*32, W*64), 2-456
SPORT_PEEK_DATA (W*32, W*64), 2-457
SPORT_PEEK_LINE (W*32, W*64), 2-458
SPORT_PURGE (W*32, W*64), 2-459
SPORT_READ_DATA (W*32, W*64), 2-460
SPORT_READ_LINE (W*32, W*64), 2-461
SPORT_RELEASE (W*32, W*64), 2-462
SPORT_SET_STATE (W*32, W*64), 2-463
SPORT_SET_STATE_EX (W*32, W*64), 2-464
SPORT_SET_TIMEOUTS (W*32, W*64), 2-467
SPORT_SPECIAL_FUNC (W*32, W*64), 2-469
SPORT_WRITE_DATA (W*32, W*64), 2-470
SPORT_WRITE_LINE (W*32, W*64), 2-471
SRAND, 2-472
SSWRQQ, 2-473
STAT, 2-474
SYSTEM, 2-476
SYSTEMMQQ, 2-478
STATES, 2-479
TIME, 2-479
TIMEOF, 2-480
TTYNAM, 2-483
UNLINK, 2-483
UNPACKTIMEQQ, 2-484
USE statement for, 1-2

Position
functions returning file, 2-94, 2-141
POSIX* constant
function returning value associated with, 2-181
POSIX* I/O flag
subroutine setting, 2-319
POSIX* routines
example of, 2-183
IPXFARGC, 2-181
IPXFCONST, 2-181
IPXFLENTTRIM, 2-182
IPXFEXITSTATUS (L*X), 2-182

IPXFSTOPSIG (L*X), 2-183
IPXFTERMSIG (L*X), 2-184
overview of, 1-11
PXFTYPEGET, 2-260
PXFTYPESET, 2-262
PXF<TYPE>GET, 2-263
PXF<TYPE>GET, 2-264
PXFACCESS, 2-265
PXFACharget, 2-266
PXFACharget, 2-264
PXFAINTSET, 2-264
PXFAINTGET, 2-263
PXFAINTGET, 2-264
PXFAINTGET, 2-264
PXFALARM, 2-266
PXFALARM, 2-263
PXFALGCLGET, 2-263
PXFALGCLGET, 2-264
PXFAREALGET, 2-263
PXFAREALGET, 2-264
PXFASURET, 2-264
PXFASTRGET, 2-263
PXFASTRSET, 2-264
PXFACALLSUBHANDLE, 2-267
PXFCFGGETISPEED (L*X), 2-268
PXFCFGETOSPEED (L*X), 2-269
PXFCFSETISPEED (L*X), 2-269
PXFCFSETOSPEED (L*X), 2-270
PXFCHARGET, 2-260
PXFCHARSET, 2-262
PXFCHDIR, 2-270
PXFCHMOD, 2-271
PXFCHOWN (L*X), 2-272
PXFCLEARENV, 2-272
PXFCLOSE, 2-273
PXFCLOSEDIR, 2-273
PXFCNTL (L*X), 2-274
PXFCORDER, 2-276
PXFCREAT, 2-276
PXFCTERMD (L*X), 2-277
PXFDGLGET, 2-260
PXFDGLSET, 2-262
PXFDUP, 2-278
PXFDUP2, 2-278
PXFE<TYPE>GET, 2-278
PXFE<TYPE>SET, 2-280
PXFECHARGET, 2-278
PXFECHARSET, 2-280
PXFEDBLGET, 2-278
PXFEDBLSET, 2-280
PXFEINT8GET, 2-278
PXFEINT8SET, 2-280
PXFEINTGET, 2-278
PXFEINTSET, 2-280
PXFEGLCLGET, 2-278
PXFEGLCLSET, 2-280
PXFEALGET, 2-278
PXFEALSET, 2-280
PXFESTRGET, 2-278
PXFESTRSET, 2-280
PXFEEXECV, 2-281
PXFEEXECVE, 2-282
PXFEEXECVP, 2-283
PXFEEXIT, 2-284
PXFFASTEXIT, 2-284
PXFFDOPEN, 2-285
PXFFFLUSH, 2-286
PXFFGETC, 2-286
PXFFILENO, 2-287
PXFFORK (L*X), 2-288
PXFFPATHCONF, 2-289
PXFFPUTC, 2-291
PXFFSEEK, 2-291
PXFFSTAT, 2-292
PXFFTELL, 2-292
PXFFGETARG, 2-294
PXFFGETATTY, 2-294
PXFFGETCWD, 2-294
PXFFGETEGID (L*X), 2-295
PXFFGETENV, 2-296
PXFFGETEUID (L*X), 2-296
PXFFGETPGRP (L*X), 2-296
PXFFGETPID, 2-296
PXFFGETPWNAM (L*X), 2-296
PXFFGETSUBHANDLE, 2-305
PXFGETUID (L*X), 2-306
PXFINT8GET, 2-260
PXFINT8SET, 2-262
PXFINTGET, 2-260
PXFINTSET, 2-262
PXFISBLK, 2-306
PXFISCHR, 2-306
PXFISCONST, 2-307
PXFISFIFO, 2-308
PXFISREG, 2-308
PXFKILL, 2-309
PXFLGCLGET, 2-260
PXFLGCLSET, 2-262
PXFLINK, 2-309
PXFLOCALTIME, 2-310
PXFSEEK, 2-311
PXFMKDIR, 2-312
PXFMKFIFO (L*X), 2-313
PXFOpen, 2-314
PXFOPENDIR, 2-316
PXFPAUSE, 2-318
PXFPipe, 2-319
PXFPOSIXIO, 2-319
PXFPUTC, 2-320
PXFREAD, 2-321
PXFREALGID, 2-260
PXFREALGET, 2-260
PXFREALSET, 2-262
PXFRENABLE, 2-322
PXFREADDIR, 2-323
PXFRMIDIR, 2-323
PXFSETENV, 2-324
PXFSGETGID (L*X), 2-325
PXFSETFPGID (L*X), 2-326
PXFSGETSID (L*X), 2-327
PXFSGETUID (L*X), 2-327
PXFSIGACTION, 2-328
PXFSIGADDSET (L*X), 2-329
PXFSIGDELSET (L*X), 2-330
PXFSIGEMPTYSET (L*X), 2-330
PXFSIGFILLSET (L*X), 2-331
PXFSIGMEMBER (L*X), 2-332
PXFSIGPEND (L*X), 2-333
PXFSIGPROCMSG (L*X), 2-333
PXFSIGSUSPEND (L*X), 2-334
PXFSLEEP, 2-334
PXFSTAT, 2-335
PXFSTRGET, 2-360
PXFSTRSET, 2-262
PXFSSTRUCTCOPY, 2-336
PXFSSTRUCTCREATE, 2-336
PXFSSTRUCTFREE, 2-341
PXFSYSCONF, 2-341
PXFTCDRAIN (L*X), 2-343
PXFTCFLOW (L*X), 2-344
PXFTCFLUSH (L*X), 2-345
PXFTCGETATTR (L*X), 2-345
PXFTCGETGRP (L*X), 2-346
PXFTCSENDBREAK (L*X), 2-346
PXFTCSETATTR (L*X), 2-347
PXFTCSETPGRP (L*X), 2-348
PXFTIME, 2-348
PXFTIMES, 2-349
PXFTTYNAM (L*X), 2-352
PXFUCOMPARE, 2-353
PXFUSTRASK, 2-353, 2-354
PXFUNAME, 2-354
PXFWAIT (L*X), 2-356
PXFWAITPID (L*X), 2-357
PXFWIFEXITED (L*X), 2-358
PXFWIFSIGNALED (L*X), 2-360
PXFWIFSTOPPED (L*X), 2-360
PXFWRITE, 2-360
PXFWRTIME, 2-360
PXFWRTIME, 2-360
table of, 1-11, 1-12
USE statement for, 1-11

Predefined QuickWin routines (W*32, W*64), 2-6, 2-178, 2-216

Procedure
  function to call after a specified time, 2-4

Process
  function executing a new, 2-379
  function returning ID of, 2-135
  function returning user ID of, 2-150
  POSIX subroutine creating pipe between two, 2-319
  POSIX subroutine executing a new, 2-281, 2-282, 2-283
  POSIX subroutine exiting from, 2-284

  POSIX subroutine returning group ID of, 2-296
  POSIX subroutine returning user ID of, 2-297
  POSIX subroutine setting group ID of (L*X), 2-325, 2-326, 2-327
  POSIX subroutine setting user ID of (L*X), 2-327
  POSIX subroutine suspending (L*X), 2-334
  POSIX subroutine waiting for specific ID (L*X), 2-357

  Process control
    portability routines for, 1-2

  Process environment
    POSIX subroutine clearing, 2-272

  Process execution
    POSIX subroutine suspending, 2-318
    subroutine suspending, 2-444

  Process ID
    function returning, 2-135
    function sending signal to, 2-189
    POSIX subroutine returning for parent, 2-303

  Processor clock
    function returning, 2-45
    function returning to nearest microsecond, 2-31

  Program call and control
    portability routines for, 1-5

  Program execution
    subroutine delaying, 2-445
    subroutine terminating, 2-2

  Programs
    running within another program, 2-379

  Prompt
    subroutine controlling for critical errors, 2-397

  Pseudorandom number generators
    RANDOM, 2-366
    subroutine changing seed for, 2-384

  PUTC, 2-257
    example of, 2-258
    POSIX version of, 2-320

  PUTIMAGE (W*32, W*64), 2-258
    constants defining actions for, 2-258
    example of, 2-260

  PUTIMAGE_W (W*32, W*64), 2-258

  PXF<TYPE>GET, 2-260

  PXF<TYPE>SET, 2-262

Index-23
PXFA<TYPE>GET, 2-263  
PXFA<TYPE>SET, 2-264  
PXFACCESS, 2-265  
PXFACHARGET, 2-263  
PXFACHARSET, 2-264  
PXFADBLGET, 2-263  
PXFADBLSET, 2-264  
PXFAINTGET, 2-263  
PXFAINTSET, 2-264  
PXFAINTTGET, 2-263  
PXFAINTTSET, 2-264  
PXFALARM, 2-266  
PXFALGCLGET, 2-263  
PXFALGCLSET, 2-264  
PXFAINT8GET, 2-263  
PXFAINT8SET, 2-264  
PXFAINTGET, 2-263  
PXFAINTSET, 2-264  
PXFALARM, 2-266  
PXFALGCLGET, 2-263  
PXFALGCLSET, 2-264  
PXFINT8GET, 2-263  
PXFINT8SET, 2-264  
PXFINTGET, 2-263  
PXFINTSET, 2-264  
PXFAINTSET, 2-264  
PXFCHDIR, 2-270  
PXFCHMOD, 2-271  
PXFCHOWN (L*X), 2-272  
PXFCELENV, 2-272  
PXFCLOSE, 2-273  
PXFCLOSEDIR, 2-273  
PXFCTERMID (L*X), 2-277  
PXFCONST, 2-276  
PXFCREAT, 2-276  
PXFCTERMID (L*X), 2-277  
PXFCREATE, 2-265  
PXFCHARGET, 2-263  
PXFCHARSET, 2-264  
PXFCHDIR, 2-270  
PXFCHMOD, 2-271  
PXFCHOWN (L*X), 2-272  
PXFCELENV, 2-272  
PXFCLOSE, 2-273  
PXFCLOSEDIR, 2-273  
PXFCTERMID (L*X), 2-277  
PXFGETARG, 2-294  
PXFGETATTY, 2-294  
PXFGETC, 2-295  
PXFGETARG, 2-294  
PXFGETATTY, 2-294  
PXFGETC, 2-295  
PXFGETC, 2-295
PXGETCWD, 2-295
PXGETEgid (L*X), 2-296
PXGETENV, 2-296
PXGETEuid (L*X), 2-297
example of, 2-300
PXGETGRGid (L*X), 2-298
example of, 2-300
PXGETGRNAM (L*X), 2-299
PXGETGROUPS (L*X), 2-299
example of, 2-300
PXGETLOGIN, 2-302
PXGETPGRP (L*X), 2-302
PXGETPID, 2-302
example of, 2-284
PXGETPPID, 2-303
example of, 2-284
PXGETPWNAM (L*X), 2-303
PXGETPWUID (L*X), 2-304
PXGETSUBHANDLE, 2-305
PXGETUID (L*X), 2-306
PXINT8GET, 2-260
example of, 2-350
PXINT8SET, 2-262
PXINTGET, 2-260
example of, 2-350
PXINTSET, 2-262
PXISBLK, 2-306
PXISCHR, 2-306
PXISCONST, 2-307
PXISDIR, 2-307
PXISFIFO, 2-308
PXISREG, 2-308
PXKILL, 2-309
PXLCMDGET, 2-260
PXLCMDSET, 2-262
PXLINK, 2-309
PXLOCALTIME, 2-310
PXLOSEEK, 2-311
PXMKDIR, 2-312
PXMKFIFO (L*X), 2-313
PXOPEN, 2-314
constants defining actions, 2-314
constants defining permissions, 2-315
example of, 2-316
PXOPENDIR, 2-316
PXPATHCONF, 2-317
constants defining configuration, 2-317
macros for constants defining configuration options, 2-318
PXPAUSE, 2-318
PXPIPE, 2-319
PXPOSIXIO, 2-319
PXPUTC, 2-320
PXREAD, 2-321
PXREADDIR, 2-321
PXREALGET, 2-260
PXREALSET, 2-262
PXRENAME, 2-322
PXREWINDDIR, 2-323
PXREMDIR, 2-323
PXSETENV, 2-324
example of, 2-325
PXGETGID (L*X), 2-325
PXGETPGID (L*X), 2-326
PXGETSID (L*X), 2-327
PXGETUID (L*X), 2-327
PXSIGACTION, 2-328
PXSIGADDSET (L*X), 2-329
PXSIGDELETSET (L*X), 2-330
PXSIGEMPTYSET (L*X), 2-330
PXSIGFILLSET (L*X), 2-331
PXSIGMEMBER (L*X), 2-332
PXSIGPENDING (L*X), 2-333
PXSIGPROCMASK (L*X), 2-333
constants defining action of, 2-334
PXSIGSUSPEND (L*X), 2-334
PXSSLEEP, 2-334
PXSTAT, 2-335
PXSTRGET, 2-260
PXFSTRSET, 2-262  
    example of, 2-339  
PXFSTRUCTCOPY, 2-336  
PXFSTRUCTCREATE, 2-336  
    example of, 2-300, 2-339, 2-350  
    names of structures for, 2-337  
PXFSTRUCTFREE, 2-341  
    example of, 2-300, 2-339, 2-350  
PXFSYSCONF, 2-341  
    constants defining system options, 2-342  
PXFTCDRAIN (L*X), 2-343  
PXFTCFLOW (L*X), 2-344  
    constants defining actions for, 2-344  
PXFTCFLUSH (L*X), 2-345  
    constants defining actions for, 2-345  
PXFTGETATTR (L*X), 2-345  
PXFTGETPGRP (L*X), 2-346  
PXFTCSSENDBREAK (L*X), 2-346  
PXFTCSSETATTR (L*X), 2-347  
    constants defining actions for, 2-347  
PXFTCSSETPGRP (L*X), 2-348  
PXFTIME, 2-348  
PXFTIMES, 2-349  
    example of, 2-350  
PXFTTYNAM (L*X), 2-352  
PXFUCOMPARE, 2-353  
PXFUMASK, 2-353, 2-354  
PXFUNAME, 2-354  
    example of, 2-339  
PXFUTIME, 2-355  
PXFWAIT, 2-356  
    example of, 2-356  
PXFWAIT (L*X), 2-356  
PXFWAITPID (L*X), 2-357  
    constants defining actions for, 2-358  
    PIDs for, 2-357  
PXFWIFEXITED (L*X), 2-358  
    example of, 2-359  
PXFWIFEXITEDT  
    example of, 2-356  
PXFWIFSIGNALED (L*X), 2-360  
PXFWRITE, 2-360  

Q  
QRANSET, 2-361  
QSORT, 2-362  
    example of, 2-363  
Quick sort  
    subroutine performing on arrays, 2-362  
Quick-access keys for menu items  
    creating with INSERTMENUQQ, 2-179  
QuickWin  
    initializing with user-defined settings, 2-175  
QuickWin application enhancement  
    routines for (W*32, W*64), 1-17  
QuickWin functions (W*32, W*64)  
    ABOUTBOXQQ, 2-2  
    APPENDMENUQQ, 2-6  
    CLICKMENUQQ, 2-30  
    DELETEMENUQQ, 2-46  
    FOCUSQQ, 2-82  
    GETACTIVEQQ, 2-97  
    GETEXITQQ, 2-119  
    GETHWNDQQ, 2-128  
    GETUNITQQ, 2-150  
    GETWINDOWCONFIG, 2-152  
    GETWSIZEQQ, 2-156  
    INCHARRQ, 2-173  
    INITIALSETTINGS, 2-175  
    INQFOCUSQQ, 2-176  
    INSERTMENUQQ, 2-177  
    MESSAGEBOXQQ, 2-213  
    MODIFYMENUFLAGSQQ, 2-215  
    MODIFYMENUROUTINEQQ, 2-216  
    MODIFYMENUSTRINGQQ, 2-218  
    PASSDIRKEYSQQ, 2-240  
    REGISTERMOUSEEVENT, 2-370  
    RGBTOINTEGER, 2-377  
    SETACTIVEQQ, 2-385  
    SETEXITQQ, 2-399  
    SETMOUSESCURSOR, 2-412  
    SETWINDOWCONFIG, 2-430  
    SETWINDOWMENUQQ, 2-434  
    SETWSIZEQQ, 2-437
UNREGISTERMOUSEEVENT, 2-485
WAITONMOUSEEVENT, 2-486
QuickWin routines (W*32, W*64), 1-16
for color conversion, 1-17
for window control and inquiry, 1-16
list of predefined, 2-6
overview of, 1-16
predefined, 2-6, 2-178, 2-216
table of, 1-16
to enhance QuickWin applications, 1-17
USE statement for, 1-16
QuickWin state messages
subroutine setting, 2-411
QuickWin status messages
subroutine setting, 2-411
QuickWin subroutines (W*32, W*64)
INTEGERTORGB, 2-180
SETMESSAGEQQ, 2-411

R
RAISEQQ, 2-363
constants defining signals, 2-364
to seed, 2-443
RAN
See your language reference
RAND, 2-364, 2-472
RANDOM, 2-364, 2-366
to seed, 2-443
Random number generators
IRAND, 2-472
RAND, 2-472
subroutine to seed, 2-472
Random numbers
DRAND, 2-70
DRANDM, 2-70
function returning double-precision, 2-70
IRAND, 2-184
IRANDM, 2-184
RAND and RANDOM, 2-364
RANDOM, 2-366
RANDU
See your language reference
RANF, 2-367
RANGET, 2-367
RANSET, 2-367
Raster operation constants
binary, 2-435
ternary, 2-259
RECTANGLE (W*32, W*64), 2-368, 2-435
to enhance QuickWin applications, 1-17
subroutine setting, 2-411
RECTANGLE_W (W*32, W*64), 2-368
Rectangles
functions drawing, 2-368
subroutines storing screen image defined by, 2-129
Reentrancy mode control
run-time routine for, 1-26
Reentrancy protection
function controlling, 2-88
REGISTERMOUSEEVENT (W*32, W*64), 2-370
constants defining mouse events, 2-370
to enhance QuickWin applications, 1-17
subroutine setting, 2-411
REMAPALLPALETTERGB (W*32, W*64), 2-371
to enhance QuickWin applications, 1-17
example of, 2-371
REMAPPALETTERGB (W*32, W*64), 2-373
Remapping RGB values for video hardware, 2-371, 2-373
RENAME, 2-375
POSIX version of, 2-322
RENAMEFILEQQ, 2-375
to enhance QuickWin applications, 1-17
example of, 2-376
RGB color
function filling an area using, 2-80
subroutine converting into components, 2-180
RGB color value
function returning background, 2-100
function returning for multiple pixels, 2-139
function returning for pixel, 2-136
function returning text, 2-145
function setting for multiple pixels, 2-419
function setting for pixel, 2-416
function setting text, 2-422
RGB color values
function remapping, 2-371, 2-373
RGB components
subroutine converting color into, 2-180
RGB value
  function converting integer to, 2-377
  function returning graphics, 2-105
  function setting current, 2-391
RGBTOINTEGER (W*32, W*64), 2-377
  example of, 2-378
RINDEX, 2-378
  example of, 2-378
RTC, 2-379
  example of, 2-379
RUNQQ, 2-379
  example of, 2-380
Run-time environment
  function cleaning up, 2-86
  function initializing, 2-87
  run-time routines affecting, 1-26
Run-time errors
  functions returning most recent, 2-130
Run-Time Library (RTL)
  function controlling reentrancy protection for, 2-88
Run-time routines, 1-25
  affecting the run-time environment, 1-26
  COMMITQQ, 2-38
  for error handling, 1-26
  for file management, 1-25
  for floating-point inquiry and control, 1-26
  for keyboards and speakers, 1-25
  for memory assignment, 1-26
  for reentrancy mode control, 1-26
  FOR_DESCRIPTOR_ASSIGN (W*32, W*64), 2-83
  FOR_GET_FPE, 2-86
  for_rtl_finish_, 2-86
  for_rtl_init_, 2-87
  FOR_SET_FPE, 2-87
  FOR_SET_REENTRANCY, 2-88
  GERROR, 2-96
  GETCHARQQ, 2-102
  GETEXCEPTIONPTRSQQ (W*32, W*64), 2-118
  GETSTRQQ, 2-143
  overview of, 1-25
  PEEKCHARQQ, 2-243
  PERROR, 2-244
  table of, 1-25
  TRACEBACKQQ, 2-480
  USE statement for, 1-25

S
SAVEIMAGE (W*32, W*64), 2-380
SAVEIMAGE_W (W*32, W*64), 2-380
SCANENV, 2-381
Screen area
  subroutine erasing and filling, 2-28
Screen images
  subroutines storing rectangle, 2-129
SCROLLTEXTWINDOW (W*32, W*64), 2-382
  example of, 2-382
SCWRQQ, 2-383
  example of, 2-190
SECNDS, 2-384
  example of, 2-384
Seconds
  function returning since Greenwich mean time, 2-379
  function returning since TIMEF was called, 2-480
SEED, 2-384
  example of, 2-385
Seeds
  subroutine changing for RAND and IRAND, 2-472
  subroutine changing for RANDOM, 2-384
  subroutine returning, 2-185, 2-367
  subroutine setting, 2-72, 2-186, 2-361, 2-367
Serial port I/O
  portability routines for, 1-6
Serial port I/O routines (W*32, W*64)
  SPORT_CANCEL_IO, 2-448
  SPORT_CONNECT, 2-449
  SPORT_CONNECT_EX, 2-450
  SPORT_GET_HANDLE, 2-452
  SPORT_GET_STATE, 2-453
  SPORT_GET_STATE_EX, 2-454
  SPORT_GET_TIMEOUTS, 2-456
  SPORT_PEEK_DATA, 2-457
  SPORT_PEEK_LINE, 2-458
  SPORT_PURGE, 2-459
  SPORT_READ_DATA, 2-460
  SPORT_READ_LINE, 2-461
  SPORT_RELEASE, 2-462
  SPORT_SET_STATE, 2-463
  SPORT_SET_STATE_EX, 2-464
  SPORT_SET_TIMEOUTS, 2-467

Index-28
Index

SPORT_SHOW_STATE, 2-468
SPORT_SPECIAL_FUNC, 2-469
SPORT_WRITE_DATA, 2-470
SPORT_WRITE_LINE, 2-471

SETACTIVEQQ (W*32, W*64), 2-385
related to FOCUSQQ, 2-82

SETBKCOLOR (W*32, W*64), 2-386
eample of, 2-387

SETBKCOLORRGB (W*32, W*64), 2-387
eample of, 2-388

SETCLIPRGN (W*32, W*64), 2-389
eample of, 2-389

SETCOLOR (W*32, W*64), 2-390
and FLOODFILL, 2-79

eample of, 2-391

SETCOLORRGB (W*32, W*64), 2-391
and FLOODFILLRGB, 2-80

eample of, 2-81, 2-388, 2-393

SETCONTROLFPQQ, 2-393
constants defining control word, 2-393

eample of, 2-395

SETDAT, 2-395

eample of, 2-396

SETENVQQ, 2-396

eample of, 2-117, 2-397
POSIX version of, 2-324

SETERRORMODEQQ, 2-397

eample of, 2-398

SETEXITQQ (W*32, W*64), 2-399

eample of, 2-399

SETFILEACCESSQQ, 2-400

eample of, 2-401

SETFILETIMEQQ, 2-401

eample of, 2-402

SETFILLMASK (W*32, W*64), 2-402

eample of, 2-403

SETFONT (W*32, W*64), 2-405

eample of, 2-407
letter codes for, 2-405

SETGTEXTROTATION (W*32, W*64), 2-408

eample of, 2-409

SETLINESTYLE (W*32, W*64), 2-409

eample of, 2-410

SETMESSAGEQQ (W*32, W*64), 2-411

eample of, 2-412
messages and identifiers for, 2-411

SETMOUSECURSOR (W*32, W*64), 2-412
constants defining cursor shapes, 2-413

eample of, 2-413

SETPIXEL (W*32, W*64), 2-414

eample of, 2-415

SETPIXEL_W (W*32, W*64), 2-414

SETPIXLRGB (W*32, W*64), 2-416

eample of, 2-417

SETPIXLRGB_W (W*32, W*64), 2-416

SETPIXE (W*32, W*64), 2-417

eample of, 2-418

SETPIXE (W*32, W*64), 2-419

eample of, 2-420

SETTEXTCOLOR (W*32, W*64), 2-421

eample of, 2-421

SETTEXTCOLORRGB (W*32, W*64), 2-422

eample of, 2-423

SETTEXTCURSOR (W*32, W*64), 2-423

eample of, 2-424

SETTEXTPOSITION (W*32, W*64), 2-425

eample of, 2-382, 2-425

SETTEXTWINDOW (W*32, W*64), 2-426

eample of, 2-382, 2-426

SETTIM, 2-427

eample of, 2-427

SETVIEWORG (W*32, W*64), 2-427

eample of, 2-428

SETVIEWPORT (W*32, W*64), 2-428

eample of, 2-429

SETWINDOW (W*32, W*64), 2-429

eample of, 2-430

SETWINDOWCONFIG (W*32, W*64), 2-49, 2-430

eample of, 2-433

SETWINDOWMENUQQ (W*32, W*64), 2-434

eample of, 2-434

SETWRITEMODE (W*32, W*64), 2-435
constants defining write mode, 2-435

eample of, 2-436

Index-29
SETWSIZEQQ (W*32, W*64), 2-437
  example of, 2-438

Shapes
  subroutine returning pattern used to fill, 2-123

Shell
  function to send system command to, 2-476

SHIFTL
  See your language reference

SHIFTR
  See your language reference

SHORT, 2-439
  example of, 2-439

Shortcut keys for menu items
  creating with INSETMENUQQ, 2-179

SIGNAL, 2-439
  constants defining signals, 2-440
  example of, 2-441

SIGNALQQ, 2-442
  constants defining interrupt signals, 2-442
  example of, 2-443

Signals
  function changing the action for, 2-439
  function sending to executing program, 2-363
  function sending to process ID, 2-189
  POSIX subroutine adding to set (L*X), 2-329
  POSIX subroutine changing list of blocked (L*X), 2-333
  POSIX subroutine deleting from set (L*X), 2-330
  POSIX subroutine emptying set (L*X), 2-330
  POSIX subroutine examining pending (L*X), 2-333
  POSIX subroutine filling set (L*X), 2-331
  POSIX subroutine setting action of, 2-328
  POSIX subroutine testing whether set member (L*X), 2-332

SLEEP, 2-444
  example of, 2-444
  POSIX version of, 2-334

SLEEPQQ, 2-445
  example of, 2-445

Sorting a one-dimensional array, 2-445

SORTQQ, 2-445
  constants defining numeric arrays, 2-446
  example of, 2-447

Speakers
  portability routines for, 1-5
  run-time routines for, 1-25
  subroutine to sound, 2-18

SPLITPATHQQ, 2-447
  example of, 2-448

SPORT routines (W*32, W*64)
  SPORT_CANCEL_IO, 2-448
  SPORT_CONNECT, 2-449
  SPORT_CONNECT_EX, 2-450
  SPORT_GET_HANDLE, 2-452
  SPORT_GET_STATE, 2-453
  SPORT_GET_STATE_EX, 2-454
  SPORT_GET_TIMEOUTS, 2-456
  SPORT_PEEK_DATA, 2-457
  SPORT_PEEK_LINE, 2-458
  SPORT_PURGE, 2-459
  SPORT_READ_DATA, 2-460
  SPORT_READ_LINE, 2-461
  SPORT_RELEASE, 2-462
  SPORT_SET_STATE, 2-463
  SPORT_SET_STATE_EX, 2-464
  SPORT_SET_TIMEOUTS, 2-467
  SPORT_SHOW_STATE, 2-468
  SPORT_SPECIAL_FUNC, 2-469
  SPORT_WRITE_DATA, 2-470
  SPORT_WRITE_LINE, 2-471

SPORT_CANCEL_IO (W*32, W*64), 2-448
  example of, 2-449

SPORT_CONNECT (W*32, W*64), 2-449
  example of, 2-450

SPORT_CONNECT_EX (W*32, W*64), 2-450
  example of, 2-452

SPORT_GET_HANDLE (W*32, W*64), 2-452
  example of, 2-452

SPORT_GET_STATE (W*32, W*64), 2-453
  example of, 2-453

SPORT_GET_STATE_EX (W*32, W*64), 2-454
  example of, 2-456

SPORT_GET_TIMEOUTS (W*32, W*64), 2-456
  example of, 2-457

SPORT_PEEK_DATA (W*32, W*64), 2-457
  example of, 2-458
SPORT_PEEK_LINE (W*32, W*64), 2-458
  example of, 2-459
SPORT_PURGE (W*32, W*64), 2-459
  example of, 2-460
SPORT_READ_DATA (W*32, W*64), 2-460
  example of, 2-461
SPORT_READ_LINE (W*32, W*64), 2-461
  example of, 2-462
SPORT_RELEASE (W*32, W*64), 2-462
  example of, 2-463
SPORT_SET_STATE (W*32, W*64), 2-463
  example of, 2-464
SPORT_SET_STATE_EX (W*32, W*64), 2-464
  example of, 2-467
SPORT_SET_TIMEOUTS (W*32, W*64), 2-467
  example of, 2-468
SPORT_SHOW_STATE (W*32, W*64), 2-468
  example of, 2-469
SPORT_SPECIAL_FUNC (W*32, W*64), 2-469
  example of, 2-470
SPORT_WRITE_DATA (W*32, W*64), 2-470
  example of, 2-471
SPORT_WRITE_LINE (W*32, W*64), 2-471
  example of, 2-472
SRAND, 2-472
  example of, 2-473
SSWRQQ, 2-473
  example of, 2-473
Standard error stream
  subroutine sending a message to, 2-244
STAT, 2-474
  constants defining Windows access modes in, 2-475
  example of, 2-476
  POSIX version of, 2-335
  values of array elements in, 2-92, 2-474
Status
  function returning for graphics routines, 2-159
Status word
  subroutine clearing exception flags in floating-point, 2-29
  subroutines returning floating-point, 2-141, 2-473
Strings
  function locating last nonblank character in, 2-195
  POSIX function returning index of character in, 2-182
  Structure component
    POSIX subroutine returning array values stored in, 2-263
    POSIX subroutine returning value stored in, 2-260
    POSIX subroutine returning values stored in array element, 2-278
    POSIX subroutine setting array element, 2-280
    POSIX subroutine setting value of, 2-262
    POSIX subroutine setting value of array, 2-264
Structure termios (L*X)
  POSIX subroutine returning input baud rate from, 2-268
  POSIX subroutine returning output baud rate from, 2-269
  POSIX subroutine setting input baud rate from, 2-269
  POSIX subroutine setting output baud rate from, 2-270
Structures
  in POSIX library, 2-337
  POSIX subroutine copying contents of, 2-336
  POSIX subroutine creating, 2-336
  POSIX subroutine deleting, 2-341
Subroutines
  function to run at a specified time, 2-4
  POSIX subroutine calling associated, 2-267
Substring
  function locating index of last occurrence of, 2-378
SYSTEM, 2-476
  example of, 2-477
System codepage
  function returning number for, 2-227
System command
  function sending to command interpreter, 2-478
  function sending to shell, 2-476
System control or inquiry
  portability routines for, 1-5
System date
  function setting, 2-395
  subroutine returning, 2-163
System name
  POSIX subroutine returning, 2-354
System options
  POSIX subroutine returning values of, 2-341
System prompt
  subroutine controlling for critical errors, 2-397

System time
  function converting to ASCII string, 2-31, 2-42
  function returning, 2-479
  subroutine setting, 2-427, 2-479
SYSTEMQQ, 2-478
  example of, 2-478

T

Terminal
  POSIX subroutine creating settings for (L*X), 2-347
  POSIX subroutine discarding I/O for (L*X), 2-345
  POSIX subroutine returning group ID for (L*X), 2-346
  POSIX subroutine returning pathname of (L*X), 2-352
  POSIX subroutine returning settings for (L*X), 2-345
  POSIX subroutine sending break to (L*X), 2-346
  POSIX subroutine setting group ID for (L*X), 2-348
  POSIX subroutine suspending transmission for (L*X), 2-343, 2-344
  POSIX subroutine testing whether file is connected to, 2-294
  subroutine specifying device name for, 2-483

Terminal pathname
  POSIX subroutine generating (L*X), 2-277

Ternary raster operation constants, 2-259

Text
  function controlling truncation of, 2-488
  function controlling wrapping of, 2-488
  function returning orientation of, 2-127
  function returning width for use with OUTGTEXT, 2-126
  subroutine sending to screen (including blanks), 2-236, 2-238
  subroutine sending to screen (special fonts), 2-236

Text color
  function returning RGB value of, 2-145

Text color index
  function returning, 2-144
  function returning RGB value of, 2-145
  function setting, 2-421
  function setting RGB value of, 2-422

Text cursor
  function setting height and width of, 2-423

Text output
  function returning background color index for, 2-99
  function returning background RGB color for, 2-100
  function setting background color index for, 2-386
  function setting background RGB color for, 2-387

Text position
  subroutine returning, 2-147
  subroutine setting, 2-425

Text window
  subroutine returning boundaries of, 2-148
  subroutine scrolling the contents of, 2-382
  subroutine setting boundaries of, 2-426

TIME, 2-479
  example of, 2-479

Time
  ALARM function for subroutines, 2-4
  function returning for current locale, 2-225
  function returning seconds since midnight, 2-384
  function returning system, 2-479
  POSIX subroutine converting, 2-310
  POSIX subroutine returning, 2-348, 2-349
  subroutine returning, 2-149
  subroutine returning Greenwich mean, 2-157
  subroutine returning in array, 2-187
  subroutine returning local zone, 2-198
  subroutine setting system, 2-427, 2-479
  subroutine unpacking a packed, 2-484

Time and date
  routine returning as ASCII string, 2-76
  subroutine packing values for, 2-238

TIMEF, 2-480
  example of, 2-480

Traceback
  function returning EPTR argument for, 2-118
  run-time routine for, 1-26
  subroutine aiding in, 2-480

TRACEBACKQQ, 2-480
TRACEBACKQQ (W*32, W*64)
  examples of, 2-482

TTYNAM, 2-483
  POSIX version of, 2-352
Index

U
Unit 5
  function returning next character from, 2-101
Unit 6
  function writing a character to, 2-257
Unit number
  function testing whether it’s a terminal, 2-186
Units
  POSIX subroutine opening external, 2-285
  POSIX subroutine returning descriptor of, 2-287
UNLINK, 2-483
  example of, 2-484
UNPACKTIMEQQ, 2-484
  example of, 2-485
UNREGISTERMOUSEEVENT (W*32, W*64), 2-485
USE module, 1-1
  for AUTO routines (W*32), 1-23
  for COM routines (W*32), 1-23
  for dialog routines (W*32), 1-22
  for graphics routines W*32, W*64), 1-16
  for miscellaneous run-time routines, 1-25
  for NLS routines, 1-8
  for portability routines, 1-2
  for POSIX routines, 1-11
  for QuickWin routines (W*32, W*64), 1-16
  summary of, 2-1
User
  function returning group ID of, 2-126
  function returning ID of, 2-150
  subroutine returning login name of, 2-133
User ID
  function returning, 2-150

V
Variant type
  constants indicating, 2-11, 2-14, 2-16
Viewport
  subroutine using to redefine graphics, 2-428
Viewport area
  subroutine erasing and filling, 2-28
Viewport coordinates
  function determining endpoints of arc or pie, 2-98
  function returning color index of pixel, 2-135
  function returning RGB color of pixel, 2-136
  function using to draw a line, 2-191
  function using to draw a line between arrays, 2-192, 2-193
  function using to draw a line within an array, 2-256
  function using to draw Bezier curves, 2-247, 2-252
  function using to draw circle or ellipse, 2-73
  function using to draw elliptical arcs, 2-8
  function using to draw polygons, 2-253
  function using to draw rectangles, 2-368
  function using to draw wedge, 2-245
  function using to read from bitmap file, 2-196
  function using to return storage size of image, 2-172
  function using to save images to bitmap file, 2-380
  function using to set pixel to color index, 2-414
  function using to set pixel to RGB value, 2-416
  function using to transfer an image, 2-258
  functions filling (color index), 2-78
  functions filling (RGB), 2-80
  subroutine converting to physical coordinates, 2-133
  subroutine converting to windows coordinates, 2-154
  subroutine returning current graphics, 2-109
  subroutine storing image in rectangle, 2-129
  subroutine using to move graphics, 2-219
  subroutines converting from physical coordinates, 2-151
Viewport origin
  subroutine moving, 2-427
Viewport-coordinate origin
  subroutine moving, 2-427
  subroutine setting, 2-428

W
WAITONMOUSEEVENT (W*32, W*64), 2-486
  constants defining key states, 2-487
  example of, 2-488
WINABOUT predefined QuickWin routine, 2-6
WINARRANGE predefined QuickWin routine, 2-6
WINCASCADE predefined QuickWin routine, 2-6
WINCLEARPASTE predefined QuickWin routine, 2-6
WINCOPY predefined QuickWin routine, 2-6
Window
function defining coordinates for, 2-429
function determining which has focus, 2-176
function initializing appearance of default, 2-175
function making child active, 2-385
function returning unit number of active child, 2-97
function setting focus to, 2-82
subroutine scrolling the contents of text, 2-382
Window area
subroutine erasing and filling, 2-28
Window control and inquiry
QuickWin routines for (W*32, W*64), 1-16
Window handle
function returning unit number corresponding to, 2-150
Window unit number
function converting to handle, 2-128
Windows*
function converting unit number to handle, 2-128
function returning position of, 2-152
function returning properties of, 2-152
function returning size of, 2-156
function returning unit number of, 2-150
function setting position of, 2-437
function setting properties of child, 2-430
function setting size of, 2-437
subroutine returning boundaries of text, 2-148
subroutine scrolling the contents of text, 2-382
subroutine setting boundaries of text, 2-426

Windows* API
BitBlt, 2-258
CoCreateInstance, 2-33, 2-34
CreateBindCtx, 2-35
CreateFile, 2-452
CreateFontIndirect, 2-405, 2-430
CreateProcess, 2-476, 2-478
EscapeCommFunction, 2-469
GetActiveObject, 2-34, 2-35
GetEnvironmentVariable, 2-116
GetExceptionInformation, 2-480
MkParseDisplayName, 2-35
PurgeComm, 2-459
SetEnvironmentVariable, 2-116
SetFileApisToANSI, 2-235
SetFileApisToOEM, 2-235
SetROP2, 2-435

Windows* bitmap file
function saving an image into, 2-380
Windows* coordinates
function returning color index of pixel, 2-135
function returning RGB color of pixel, 2-136
function using to define a window, 2-429
function using to draw a line, 2-191
function using to draw Bezier curves, 2-247, 2-252
function using to draw circle or ellipse, 2-73
function using to draw elliptical arcs, 2-8
function using to draw polygons, 2-253
function using to draw rectangles, 2-368
function using to draw wedge, 2-245
function using to read from bitmap file, 2-196
function using to return storage size of image, 2-172
function using to save images to bitmap file, 2-380
function using to set pixel to color index, 2-414
function using to set pixel to RGB value, 2-416
function using to transfer an image, 2-258
functions filling (color index), 2-78
functions filling (RGB), 2-80
subroutine converting from viewport coordinates, 2-154
subroutine returning current graphics, 2-109
subroutine storing image in rectangle, 2-129
subroutine using to move graphics, 2-219
subroutines converting from physical coordinates, 2-151

Windows* properties
function returning, 2-152
function setting, 2-430, 2-437
WINEXIT predefined QuickWin routine, 2-6
WINFULLSCREEN predefined QuickWin routine, 2-6
WININDEX predefined QuickWin routine, 2-6
WININPUT predefined QuickWin routine, 2-6
WINPASTE predefined QuickWin routine, 2-6
WINPRINT predefined QuickWin routine, 2-6
WINSAFE predefined QuickWin routine, 2-6
WINSELECTCALL predefined QuickWin routine, 2-6
WINSELECTGRAPHICS predefined QuickWin routine, 2-6
WINSELECTTEXT predefined QuickWin routine, 2-6
WINSIZETOFIT predefined QuickWin routine, 2-6
WINSTATE predefined QuickWin routine, 2-6
WINSTATUS predefined QuickWin routine, 2-6
WINTILE predefined QuickWin routine, 2-6
WINUSING predefined QuickWin routine, 2-6
Working directory
  function returning path of, 2-112
WRAPON (W*32, W*64), 2-488
  example of, 2-489
Write mode
  function returning logical, 2-155
  function setting logical, 2-435
Write operations
  function committing to physical device, 2-38