© 1995–2007, QNX Software Systems GmbH & Co. KG. All rights reserved.

Published under license by:

**QNX Software Systems International Corporation**
175 Terence Matthews Crescent
Kanata, Ontario
K2M 1W8
Canada
Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com

### Publishing history

<table>
<thead>
<tr>
<th>Date</th>
<th>Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 1995</td>
<td>First edition</td>
</tr>
<tr>
<td>December 1996</td>
<td>Second edition</td>
</tr>
<tr>
<td>April 1998</td>
<td>Third edition</td>
</tr>
<tr>
<td>July 2004</td>
<td>Fourth edition</td>
</tr>
</tbody>
</table>

Electronic edition published 2007

QNX, Neutrino, Photon, Photon microGUI, Momentics, and Aviage are trademarks, registered in certain jurisdictions, of QNX Software Systems GmbH & Co. KG, and are used under license by QNX Software Systems International Corporation. All other trademarks belong to their respective owners.
About This Reference  xxxv
What you’ll find in this guide xxxvii
What’s new in Photon for QNX Neutrino 6.3x  xxxix
  New content in the docs  xxxix
What’s new in Photon for QNX Neutrino 6.3 Service Pack 1  xxxix
  New content in the docs  xxxix
What’s new in Photon for QNX Neutrino 6.3  xxxix
  New content in the docs  xxxix
  Corrections, clarifications, and other changes  xlvii
  Deprecated functions and data types  xlvii
What’s new in Photon for QNX Neutrino 6.2.1  xlviii
  New content in the docs  xlviii
  Deprecated functions and data types  lv
  Corrections  lv
What’s new in Photon for QNX Neutrino 6.2.0  lv
  New content in the docs  lv
  Deprecated functions and data types  xlv
  Corrections, clarifications, and other changes  lv
What’s new in Photon for QNX Neutrino 6.1.0  lvii
  Patch A  lvii
  New content in the docs  lvii
  Deprecated functions and data types  lvii
  Other changes  lviii
What’s new in Photon for QNX Neutrino 6.0  lviii
  New functionality  lviii
New content in the docs lix
Corrections lxxvi
Typographical conventions lxxvi
Note to Windows users lxxviii
Technical support lxxviii

1 **Summary of Entries**

Alpha blending 4
Background processing 4
Bitmaps and Images 4
Blitting 7
Characters, translating 7
Chroma key operations 8
Clipboard operations 8
Clipping 9
Colors, converting and parsing 9
Configuration files 10
Connections to other applications 12
Coordinates, translating 14
Cursors/pointers 14
Data chains 15
Direct mode 15
Drag and drop 16
Dragging 19
Draw contexts 19
Drawing attributes 19
  General attributes 19
  Fill attributes 20
  Line (stroke) attributes 20
  Text attributes 21
Events 22
Font handling 25
Geometry 27
Video overlay 48
Wide characters 49
Widgets 49
  Callbacks and hotkey handlers 49
  Class hierarchy 51
  Control surfaces 51
  Creating and destroying widgets 55
  Custom widgets 55
  Damaging widgets 60
  Databases 60
  Family hierarchy 61
  Finding widgets in an area 63
  Focus 64
  Geometry 65
  Library initialization 66
  Menus 66
  PtComboBox 67
  PtFileSel 67
  PtGenList 69
  PtGenTree 72
  PtList 74
  PtMTrend 75
  PtMultiText 76
  PtPanelGroup 77
  PtProgress 77
  PtTerminal 78
  PtText 79
  PtTree 79
  PtTrend 81
  PtTty 81
  PtWindow 82
Realizing and unrealizing widgets 82
Resources  82
Styles  84
Updates, forcing and holding off  85
Window Manager  85

2  Ab—PhAB-Generated Code  87
AbGetABW()  90

3  AI—PhAB Translation  93
AICloseTranslation()  96
AICloseDBase()  98
AIGetEntry()  99
AIGetSize()  101
AIOpenDBase()  103
AISaveTranslation()  107
AISetEntry()  109

4  Ap—PhAB  111
ApAddClass()  114
ApAddContext()  116
ApAppendTranslation()  118
ApCloseDBase()  121
ApCloseMessageDB()  123
ApCopyDBWidget()  124
ApCreateDBWidget()  126
ApCreateDBWidgetFamily()  129
ApCreateModule()  132
ApCreateWidget()  137
ApCreateWidgetFamily()  140
ApDeleteDBWidget()  143
ApError()  145
ApGetDBWidgetInfo()  148
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ApGetImageRes()</code></td>
<td>151</td>
</tr>
<tr>
<td><code>ApGetInstance()</code></td>
<td>154</td>
</tr>
<tr>
<td><code>ApGetItemText()</code></td>
<td>156</td>
</tr>
<tr>
<td><code>ApGetMessage()</code></td>
<td>158</td>
</tr>
<tr>
<td><code>ApGetTextRes()</code></td>
<td>160</td>
</tr>
<tr>
<td><code>ApGetWidgetPtr()</code></td>
<td>162</td>
</tr>
<tr>
<td><code>ApInfo_t</code></td>
<td>165</td>
</tr>
<tr>
<td><code>ApInstanceName()</code></td>
<td>167</td>
</tr>
<tr>
<td><code>ApLoadMessageDB()</code></td>
<td>169</td>
</tr>
<tr>
<td><code>ApModalWait()</code></td>
<td>171</td>
</tr>
<tr>
<td><code>ApModifyItemAccel()</code></td>
<td>173</td>
</tr>
<tr>
<td><code>ApModifyItemState()</code></td>
<td>175</td>
</tr>
<tr>
<td><code>ApModifyItemText()</code></td>
<td>177</td>
</tr>
<tr>
<td><code>ApModuleFunction()</code></td>
<td>179</td>
</tr>
<tr>
<td><code>ApModuleLocation()</code></td>
<td>181</td>
</tr>
<tr>
<td><code>ApModuleParent()</code></td>
<td>183</td>
</tr>
<tr>
<td><code>ApName()</code></td>
<td>185</td>
</tr>
<tr>
<td><code>ApOpenDBase()</code></td>
<td>187</td>
</tr>
<tr>
<td><code>ApOpenDBaseFile()</code></td>
<td>189</td>
</tr>
<tr>
<td><code>ApRemoveClass()</code></td>
<td>192</td>
</tr>
<tr>
<td><code>ApRemoveContext()</code></td>
<td>194</td>
</tr>
<tr>
<td><code>ApResClose()</code></td>
<td>196</td>
</tr>
<tr>
<td><code>ApSaveDBaseFile()</code></td>
<td>198</td>
</tr>
<tr>
<td><code>ApSetContext()</code></td>
<td>200</td>
</tr>
<tr>
<td><code>ApSetTranslation()</code></td>
<td>202</td>
</tr>
<tr>
<td><code>ApWidget()</code></td>
<td>205</td>
</tr>
</tbody>
</table>

5  *mbstr—Multibyte-Character*  207

6  *Pd—Draw Context*  211

  *PdCreateDirectContext()*  214
  *PdCreateOffscreenContext()*  216
  *PdCreateOffscreenLock()*  220
PdDestroyOffscreenLock()  222
PdDirectStart()  224
PdDirectStop()  226
PdDupOffscreenContext()  228
PdGetDevices()  231
PdGetOffscreenContextPtr()  233
PdGetOffscreenSurface()  237
PdIsOffscreenLocked()  238
PdLockOffscreen()  240
PdOffscreenContext_t  242
PdReleaseDirectContext()  243
PdSetOffscreenTranslation()  245
PdSetTargetDevice()  247
PdUnlockOffscreen()  250

7  Pf—Font Server  253
PfAllocDetailsCx()  256
PfAllocRenderCx()  258
PfAssignDllCx()  264
PfAttach(), PfAttachCx()  267
PfAttachDllCx()  270
PfAttachLocalDll()  274
PfAttachServerDll()  278
PfAttachSleuthMonitorDll()  281
PfConvertFontID(), PfConvertFontIDCx()  283
PfConvertPixelsToPointSizeCx()  287
PfDecomposeStemToID(), PfDecomposeStemToIDCx()  290
PfDefaultContext()  292
PfDetach(), PfDetachCx()  293
PfDetachLocalDll()  295
PfDynamicFontIDCx()  297
PfDynamicLoad(), PfDynamicLoadCx()  299
PfDynamicUnload(), PfDynamicUnloadCx()  306
PfExtent(), PfExtentCx() 309
PfExtentComponents(), PfExtentComponentsCx() 318
PfExtentFractTextCharPositions() 321
PfExtentText() 326
PfExtentTextCharPositions(), PfExtentTextCharPositionsCx() 329
PfExtentTextToRect() 341
PfExtentWideText() 347
PfFindFont(), PfFindFontCx() 350
PfFontBaseStem(), PfFontBaseStemCx() 355
PfFontDescription(), PfFontDescriptionCx() 357
PfFontFlags(), PfFontFlagsCx() 359
PfFontSize(), PfFontSizeCx() 361
PfFontTypeCx() 363
PfFractionalExtentText() 366
PfFreeFont(), PfFreeFontCx() 369
PfGenerateFontName(), PfGenerateFontNameCx() 371
PfGetGlyphIndexCx() 374
PfGetOutline(), PfGetOutlineCx() 379
PfGlyph(), PfGlyphCx() 388
PfLoadFont(), PfLoadFontCx() 391
PfLoadMetrics(), PfLoadMetricsCx() 393
PfQueryFontInfo(), PfQueryFontInfoCx() 395
PfQueryFonts(), PfQueryFontsCx() 398
PfRender(), PfRenderCx() 402
PfRestartServerDll() 410
PfSetOptionsDll() 412
PfSetRenderingDPICx() 414
PfTextWidthBytes() 419
PfTextWidthChars() 422
PfUnloadMetrics() 424
PfWaitOnServerDll() 426
PfWideTextWidthBytes() 428
8 Pf—Graphics 433
PgAlphaOff(), PgAlphaOffCx() 437
PgAlphaOn(), PgAlphaOnCx() 438
PgAlphaValue() 439
PgARGB() 441
PgBackgroundShadings() 443
PgBevelBox(), PgBevelBoxCx() 445
PgBlit(), PgBlitCx() 449
PgBlueValue() 451
PgCalcColorContrast() 453
PgChromaOff(), PgChromaOffCx() 455
PgChromaOn(), PgChromaOnCx() 456
PgClearDrawBuffer(), PgClearDrawBufferCx() 457
PgClearTranslation(), PgClearTranslationCx() 459
PgCMY() 460
PgColor_t 462
PgColorHSV_t 466
PgColorMatch() 467
PgConfigScalerChannel() 469
PgContextBlit(), PgContextBlitCx() 471
PgContextBlitArea(), PgContextBlitAreaCx() 474
PgContrastBevelBox(), PgContrastBevelBoxCx() 476
PgCreateDriverRegion() 480
PgCreateGC() 483
PgCreateLayerSurface() 485
PgCreateVideoChannel() 488
PgDefaultAlpha() 490
PgDefaultChroma() 491
PgDefaultFill() 492
PgDefaultGC() 493
PgDefaultMode() 494
PgDefaultStroke()  495
PgDefaultText()  496
PgDestroyGC()  497
PgDestroyVideoChannel()  498
PgDrawArc(), PgDrawArcCx()  499
PgDrawArrow(), PgDrawArrowCx()  504
PgDrawBevelBox(), PgDrawIBevelBox(), PgDrawBevelBoxCx(), PgDrawIBevelBoxCx()  506
PgDrawBeveled(), PgDrawBeveledCx()  510
PgDrawBezier(), PgDrawBezierv(), PgDrawBezierCx(), PgDrawBezierCxv()  515
PgDrawBitmap(), PgDrawBitmapv(), PgDrawBitmapCx(), PgDrawBitmapCxv()  519
PgDrawEllipse(), PgDrawEllipseCx()  523
PgDrawGradient(), PgDrawGradientCx()  527
PgDrawGradientBevelBox(), PgDrawGradientBevelBoxCx()  531
PgDrawGrid(), PgDrawGridCx()  535
PgDrawImage(), PgDrawImagev(), PgDrawImageCx(), PgDrawImageCxv()  538
PgDrawLine(), PgDrawILine(), PgDrawLineCx(), PgDrawILineCx()  542
PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()  544
PgDrawPhImage(), PgDrawPhImagev(), PgDrawPhImageCx(), PgDrawPhImageCxv()  550
PgDrawPhImageRectv(), PgDrawPhImageRectCxv()  553
PgDrawPixel(), PgDrawIPixel(), PgDrawPixelCx(), PgDrawIPixelCx()  555
PgDrawPixelArray(), PgDrawPixelArrayv(), PgDrawPixelArrayCx(), PgDrawPixelArrayCxv()  557
PgDrawPolygon(), PgDrawPolygonv(), PgDrawPolygonCx(), PgDrawPolygonCxv()  559
PgDrawRect(), PgDrawIRect(), PgDrawRectCx(), PgDrawIRectCx()  564
PgDrawRepBitmap(), PgDrawRepBitmapv(),
    PgDrawRepBitmapCx(), PgDrawRepBitmapCxv() 568
PgDrawRepImage(), PgDrawRepImagev(), PgDrawRepImageCx(),
    PgDrawRepImageCxv() 573
PgDrawRepPhImage*() 576
PgDrawRoundRect(), PgDrawRoundRectCx() 579
PgDrawSpan(), PgDrawSpanv(), PgDrawSpanCx(),
    PgDrawSpanCxv() 583
PgDrawString(), PgDrawStringv(), PgDrawStringCx(),
    PgDrawStringCxv() 587
PgDrawText*(), PgDrawTextChars*() 589
PgDrawTextArea(), PgDrawTextAreaCx() 595
PgDrawTImage(), PgDrawTImagev(), PgDrawTImageCx(),
    PgDrawTImageCxv() 598
PgDrawTrend(), PgDrawTrendv(), PgDrawTrendCx(),
    PgDrawTrendCxv() 601
PgExtentMultiText() 605
PgExtentText() 607
PgFlush(), PgFFlush(), PgFlushCx(), PgFFlushCx() 609
PgGetColorModel(), PgGetColorModelCx() 611
PgGetGC(), PgGetGCCx() 613
PgGetGraphicsHWCaps() 615
PgGetLayerCaps() 619
PgGetOverlayChromaColor() 621
PgGetPalette() 622
PgGetRegion(), PgGetRegionCx() 624
PgGetScalerCapabilities() 626
PgGetVideoMode() 628
PgGetVideoModeInfo() 630
PgGetVideoModeList() 634
PgGray() 636
PgGrayValue() 638
PgGreenValue() 640
"PgHSV() 642
PgHSV2RGB() 645

PgLayerCaps_t 647
PgLockLayer() 655

PgMap_t 657
PgNextVideoFrame() 658
PgReadScreen() 659
PgReadScreenSize() 662
PgRedValue() 664
PgRGB() 666
PgRGB2HSV() 668

PgScalerCaps_t 670

PgScalerProps_t 673

PgSetAlpha(), PgSetAlphaCx() 676
PgSetAlphaBlend(), PgSetAlphaBlendCx() 685
PgSetChroma(), PgSetChromaCx() 687
PgSetClipping(), PgSetClippingCx() 689
PgSetColorModel(), PgSetColorModelCx() 691

PgSetDPMSMode() 693
PgSetDrawBufferSize(), PgSetDrawBufferSizeCx() 695
PgSetDrawMode(), PgSetDrawModeCx() 697
PgSetFillColor(), PgSetFillColorCx() 701
PgSetFillDither(), PgSetFillDitherCx() 703
PgSetFillTransPat(), PgSetFillTransPatCx() 708
PgSetFillXORColor(), PgSetFillXORColorCx() 710

PgSetFont(), PgSetFontCx() 712
PgSetGC(), PgSetGCCx() 714

PgSetLayerArg() 716

PgSetLayerSurface() 723

PgSetMultiClip(), PgSetMultiClipCx() 726

PgSetPalette(), PgSetPaletteCx() 728

PgSetPlaneMask(), PgSetPlaneMaskCx() 731
PgSetRegion(), PgSetRegionCx()  733
PgSetStrokeCap(), PgSetStrokeCapCx()  735
PgSetStrokeColor(), PgSetStrokeColorCx()  737
PgSetStrokeDash(), PgSetStrokeDashCx()  739
PgSetStrokeDither(), PgSetStrokeDitherCx()  742
PgSetStrokeJoin(), PgSetStrokeJoinCx()  744
PgSetStrokeTransPat(), PgSetStrokeTransPatCx()  747
PgSetStrokeWidth(), PgSetStrokeFWidth(), PgSetStrokeWidthCx(),
   PgSetStrokeFWidthCx()  749
PgSetStrokeXORColor(), PgSetStrokeXORColorCx()  751
PgSetTextColor(), PgSetTextColorCx()  753
PgSetTextDither(), PgSetTextDitherCx()  755
PgSetTextTransPat(), PgSetTextTransPatCx()  757
PgSetTextXORColor(), PgSetTextXORColorCx()  759
PgSetTranslation(), PgSetTranslationCx()  761
PgSetUnderline(), PgSetUnderlineCx()  763
PgSetUserClip(), PgSetUserClipAbsolute(), PgSetUserClipCx(),
   PgSetUserClipAbsoluteCx()  765
PgSetVideoMode()  767
PgShmemAttach()  769
PgShmemCleanup()  771
PgShmemCreate()  773
PgShmemDestroy()  775
PgShmemDetach()  777
PgSwapDisplay(), PgSwapDisplayCx()  779
PgUnlockLayer()  781
PgVideoChannel_t  783
PgWaitDrawComplete()  785
PgWaitHWIdle()  786
PgWaitVSync(), PgWaitVSyncCx()  787

9  Ph—Photon  789
PhAddMergeTiles()  792
PhAllocPackType() 794
PhArea_t 797
PhAreaToRect() 798
PhAttach() 799
PhBlit() 803
PhBitmapCursorDescription_t 805
PhCancelDrag() 808
PhChannelAttach() 810
PhCharacterCursorDescription_t 814
PhClipboardCopyString() 816
PhClipboardHdr 818
PhClipboardPasteString() 819
PhClipboardRead() 821
PhClipboardWrite() 823
PhClipTilings() 826
PhCoalesceTiles() 828
PhCopyTiles() 830
PhCreateImage() 832
PhCreateTransportCtrl() 834
PhCursorDef_t 835
PhCursorDescription_t 837
PhDCCreate() 838
PhDCGetCurrent() 841
PhDCRelease() 842
PhDCSetCurrent() 844
PhDetach() 847
PhDeTranslateRect() 849
PhDeTranslateTiles() 850
PhDim_t 852
PhDragEvent_t 853
PhEmit() 857
PhEmitmx() 860
PhEvent_t 863
PhEventArm() 881
PhEventEmit() 883
PhEventEmitmx() 886
PhEventNext() 889
PhEventPeek() 891
PhEventRead() 894
PhEventRegion_t 898
PhFindTransportType() 900
PhFreeTiles() 901
PhFreeTransportType() 902
PhGetAllTransportHdrs() 904
PhGetConnectId() 905
PhGetConnectInfo() 906
PhGetData() 908
PhGetMsgSize() 909
PhGetNextInlineData() 911
PhGetNextTransportHdr() 913
PhGetRects() 915
PhGetTile() 917
PhGetTransportHdr() 919
PhGetTransportVectors() 920
PhImage_t 922
PhInitDrag() 928
PhInputGroup() 933
PhIntersectTilings() 934
PhKeyEvent_t 936
PhKeyToMb() 940
PhLibVersion() 941
PhLinkTransportData() 942
PhLocateTransHdr() 944
PhMakeGhostBitmap() 946
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhMakeTransBitmap()</td>
<td>948</td>
</tr>
<tr>
<td>PhMakeTransparent()</td>
<td>953</td>
</tr>
<tr>
<td>PhMallocUnpack()</td>
<td>957</td>
</tr>
<tr>
<td>PhMergeTiles()</td>
<td>959</td>
</tr>
<tr>
<td>PhMoveCursorAbs()</td>
<td>961</td>
</tr>
<tr>
<td>PhMoveCursorRel()</td>
<td>962</td>
</tr>
<tr>
<td>PhPackEntry()</td>
<td>963</td>
</tr>
<tr>
<td>PhPackType()</td>
<td>967</td>
</tr>
<tr>
<td>PhPoint_t</td>
<td>972</td>
</tr>
<tr>
<td>PhPointerEvent_t</td>
<td>973</td>
</tr>
<tr>
<td>PhQueryCursor()</td>
<td>976</td>
</tr>
<tr>
<td>PhQueryRids()</td>
<td>979</td>
</tr>
<tr>
<td>PhQuerySystemInfo()</td>
<td>982</td>
</tr>
<tr>
<td>PhReattach()</td>
<td>984</td>
</tr>
<tr>
<td>PhRect_t</td>
<td>986</td>
</tr>
<tr>
<td>PhRectIntersect()</td>
<td>987</td>
</tr>
<tr>
<td>PhRectsToTiles()</td>
<td>988</td>
</tr>
<tr>
<td>PhRectToArea()</td>
<td>990</td>
</tr>
<tr>
<td>PhRectUnion()</td>
<td>991</td>
</tr>
<tr>
<td>PhRegion_t</td>
<td>992</td>
</tr>
<tr>
<td>PhRegionChange()</td>
<td>997</td>
</tr>
<tr>
<td>PhRegionClose()</td>
<td>1001</td>
</tr>
<tr>
<td>PhRegionDataFindType()</td>
<td>1002</td>
</tr>
<tr>
<td>PhRegionDataHdr_t</td>
<td>1004</td>
</tr>
<tr>
<td>PhRegionInfo()</td>
<td>1006</td>
</tr>
<tr>
<td>PhRegionOpen()</td>
<td>1009</td>
</tr>
<tr>
<td>PhRegionQuery()</td>
<td>1014</td>
</tr>
<tr>
<td>PhRegisterTransportType()</td>
<td>1016</td>
</tr>
<tr>
<td>PhReleaseImage()</td>
<td>1018</td>
</tr>
<tr>
<td>PhReleaseTransportCtrl()</td>
<td>1020</td>
</tr>
<tr>
<td>PhReleaseTransportHdrs()</td>
<td>1021</td>
</tr>
<tr>
<td>PhSortTiles()</td>
<td>1022</td>
</tr>
</tbody>
</table>
PhSysInfo_t 1024
PhTile_t 1029
PhTilesBoundingRect() 1030
PhTilesToRects() 1032
PhTimerArm() 1034
PhTo8859_I() 1036
PhTranslateRect() 1037
PhTranslateTiles() 1039
PhTransportCtrl_t 1041
PhTransportFindLink() 1042
PhTransportLink_t 1043
PhTransportRegEntry_t 1044
PhTransportType() 1047
PhUnlinkTransportHdr() 1051
PhUnpack() 1052
PhWindowEvent_t 1054
PhWindowQueryVisible() 1058

10 Pi—Images 1061
PiCropImage() 1064
PiDuplicateImage() 1066
PiFlipImage() 1068
PiGetPixel() 1070
PiGetPixelFromData() 1072
PiGetPixelRGB() 1074
PiSetPixel() 1076
PiSetPixelInData() 1078

11 Pm—Memory 1081
PmMemCreateMC() 1084
PmMemFlush() 1089
PmMemReleaseMC() 1091
PmMemSetChunkSize() 1093
12 Pp—Printing 1101
PpContinueJob() 1104
PpCreatePC() 1107
PpEndJob() 1109
PpFreePrinterList() 1111
PpGetCanvas() 1112
PpGetPC() 1114
PpLoadDefaultPrinter() 1119
PpLoadPrinter() 1121
PpLoadPrinterList() 1123
PpPrintContext_t 1124
PpPrintNewPage() 1134
PpPrintWidget() 1136
PpReleasePC() 1139
PpSetCanvas() 1141
PpSetPC() 1143
PpStartJob() 1147
PpSuspendJob() 1149

13 Pt—Widget Toolkit 1151
PtAddCallback() 1154
PtAddCallbacks() 1157
PtAddClassStyle() 1159
PtAddData() 1161
PtAddEventHandler() 1163
PtAddEventHandlers() 1166
PtAddFilterCallback() 1168
PtAddFilterCallbacks() 1170
PtAddHotkeyHandler()  1172
PtAddResponseType()  1176
PtAlert()  1180
PtAllowExit()  1184
PtAppAddCallback()  1186
PtAppAddEventHandler()  1188
PtAppAddFd(), PtAppAddFdPri()  1190
PtAddFilterCallback()  1193
PtAppAddHotkeyHandler()  1195
PtAppAddInput()  1197
PtAppAddInputRemote()  1202
PtAppAddSignalProc()  1207
PtAppAddWorkProc()  1209
PtAppCreatePulse()  1212
PtAppDeletePulse()  1214
PtAppGetResource()  1216
PtAppGetResources()  1219
PtAppInit()  1221
PtAppPulseTrigger()  1223
PtAppRemoveCallback()  1225
PtAppRemoveEventHandler()  1227
PtAppRemoveFd()  1229
PtAppRemoveFilterCallback()  1231
PtAppRemoveHotkeyHandler()  1233
PtAppRemoveInput()  1235
PtAppRemoveSignal()  1236
PtAppRemoveWorkProc()  1238
PtAppSetFdMode()  1240
PtAppSetResource()  1242
PtAppSetResources()  1244
PtArg_t  1252
Pt_ARG()  1253
Contents
PtConnectionServerDestroy() 1314
PtConnectionServerGetUserData() 1315
PtConnectionServerSetError() 1316
PtConnectionServerSetUserData() 1318
PtConnectionTmpName() 1319
PtConnectionWaitForName() 1320
PtConnectorCreate() 1323
PtConnectorDestroy() 1325
PtConnectorGetId() 1326
PtConsoleSwitch() 1327
PtContainerBox() 1329
PtContainerFindFocus() 1331
PtContainerFocusNext() 1332
PtContainerFocusPrev() 1334
PtContainerGiveFocus() 1336
PtContainerHit() 1341
PtContainerHold() 1343
PtContainerNullFocus() 1345
PtContainerRelease() 1347
PtCRC() 1349
PtCRCValue() 1351
PtCreateActionSurface() 1353
PtCreateClassStyle() 1357
PtCreateSurface() 1359
PtCreateTransportCtrl() 1363
PtCreateWidget() 1364
PtDamageExtent() 1367
PtDamageSurface(), PtDamageSurfaceById() 1369
PtDamageSurfaceByAction() 1371
PtDestroyWidget() 1372
PtDestroyAllSurfaces() 1374
PtDestroySurface() 1375
PtDestroySurfaceById() 1376
PtDestroyWidget() 1378
PtDisableSurface(), PtDisableSurfaceById() 1380
PtDisableSurfaceByAction() 1382
PtDndFetch_t 1384
PtDndSelect() 1387
PtDupClassStyle() 1389
PtEnableSurface(), PtEnableSurfaceById() 1391
PtEnableSurfaceByAction() 1393
PtEndFlux() 1395
PtEnter() 1396
PtEventHandler() 1401
PtExit() 1402
PtExtentWidget() 1404
PtExtentWidgetFamily() 1405
PtFdProcF_t, PtFdProc_t 1406
PtFepCmd() 1407
PtFileSelection() 1409
PtFindChildClass() 1423
PtFindChildClassMember() 1426
PtFindClassStyle() 1428
PtFindContainer() 1430
PtFindData() 1431
PtFindDisjoint() 1433
PtFindFocusChild() 1435
PtFindFocusNextFrom() 1436
PtFindFocusPrevFrom() 1437
PtFindGuardian() 1438
PtFindNextData() 1440
PtFindSurface() 1442
PtFindSurfaceByAction() 1444
PtFlush() 1446
PtFontSelection() 1448
PtForwardWindowEvent() 1451
PtForwardWindowTaskEvent() 1453
PtGetAbsPosition() 1455
PtGetControlFlags() 1456
PtGetDndFetchIndex() 1457
PtGetParent() 1460
PtGetParentWidget() 1462
PtGetRcvidPid() 1463
PtGetRcvidPidNd() 1465
PtGetResource() 1467
PtGetResources() 1470
PtGetStyleMember() 1473
PtGetWidgetStyle() 1476
PtGiveFocus() 1478
PtGlobalFocusNext() 1480
PtGlobalFocusNextContainer() 1482
PtGlobalFocusNextFrom() 1484
PtGlobalFocusPrev() 1486
PtGlobalFocusPrevContainer() 1488
PtGlobalFocusPrevFrom() 1490
PtHelpQuit() 1492
PtHelpSearch() 1493
PtHelpTopic() 1496
PtHelpTopicRoot() 1498
PtHelpTopicTree() 1500
PtHelpUrl() 1502
PtHelpUrlRoot() 1504
PtHideSurface(), PtHideSurfaceById() 1506
PtHideSurfaceByAction() 1508
PtHit() 1509
PtHold() 1510
PtInflateBalloon() 1512
PtInit() 1514
PtInitDnd() 1516
PtInputCallbackProcF_t, PtInputCallbackProc_t 1519
PtInsertSurface(), PtInsertSurfaceById() 1520
PtIsFluxing() 1522
PtIsFocused() 1523
PtLeave() 1525
PtMainLoop() 1529
PtMakeModal() 1530
PtMessageBox() 1531
PtModalBlock() 1533
PtModalEnd() 1540
PtModalStart() 1541
PtModalUnblock() 1543
PtNextTopLevelWidget() 1545
PtNotice() 1546
PtPassword() 1550
PtPositionMenu() 1555
PtPreventExit() 1557
PtPrintPropSelect() 1559
PtPrintSelect() 1569
PtPrintSelection() 1571
PtProcessEvent() 1576
PtPrompt() 1578
PtPulseArm() 1583
PtQuerySystemInfo() 1585
PtQuitMainLoop() 1587
PtRealizeWidget() 1589
PtReattach() 1591
PtRelease() 1593
PtReleaseTransportCtrl() 1594
PtRemoveCallback() 1595
PtRemoveCallbacks() 1597
PtRemoveData() 1599
PtRemoveEventHandler() 1601
PtRemoveEventHandlers() 1603
PtRemoveFilterCallback() 1605
PtRemoveFilterCallbacks() 1607
PtRemoveHotkeyHandler() 1609
PtReparentWidget() 1611
PtReRealizeWidget() 1613
PtResizeEventMsg() 1615
PtSendEventToWidget() 1617
PtSetAreaFromCanvas() 1619
PtSetArg() 1621
PtSetClassStyleMethods() 1623
PtSetParentWidget() 1625
PtSetResource() 1627
PtSetResources() 1629
PtSetStyleMember() 1631
PtSetStyleMembers() 1635
PtSetWidgetStyle() 1637
PtShowSurface(), PtShowSurfaceById() 1639
PtShowSurfaceByAction() 1641
PtSignalProcF_t, PtSignalProc_t 1643
PtSpawn() 1644
PtSpawnDeleteCallback() 1648
PtSpawnSetCallback() 1649
PtSpawnWait() 1650
PtStartFlux() 1652
PtSurfaceActionId() 1653
PtSurfaceAddData(), PtSurfaceAddDataById() 1654
PtSurfaceBrotherBehind() 1656
PtSurfaceBrotherInFront() 1657
PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById() 1658
PtSurfaceExtent(), PtSurfaceExtentById() 1660
PtSurfaceGetData() 1662
PtSurfaceHit() 1664
PtSurfaceId() 1666
PtSurfaceInBack() 1667
PtSurfaceInFront() 1668
PtSurfaceIsDisabled() 1669
PtSurfaceIsEnabled() 1670
PtSurfaceIsHidden() 1671
PtSurfaceIsShown() 1672
PtSurfaceRect(), PtSurfaceRectById() 1673
PtSurfaceRemoveData(), PtSurfaceRemoveDataById() 1675
PtSurfaceTestPoint() 1677
PtSurfaceToBack(), PtSurfaceToBackById() 1678
PtSurfaceToFront(), PtSurfaceToFrontById() 1680
PtSyncWidget() 1682
PtTimerArm() 1683
PtTransportCtrl_t 1685
PtTransportRequestable() 1687
PtTransportType() 1693
PtUnblockWindows() 1697
PtUnlinkData() 1698
PtUnrealizeWidget() 1699
PtUpdate() 1701
PtValidParent() 1702
PtWidgetActiveSurface() 1704
PtWidgetArea() 1705
PtWidgetBrotherBehind() 1706
PtWidgetBrotherInFront() 1709
PtWidgetChildBack() 1710
PtWidgetChildFront() 1712
PtWidgetClass() 1714
PtWidgetClassFlags() 1716
PtWidgetDim() 1718
PtWidgetExtent() 1719
PtWidgetFamily() 1721
PtWidgetFlags() 1723
PtWidgetHelpHit() 1724
PtWidgetInsert() 1726
PtWidgetIsClass() 1728
PtWidgetIsClassMember() 1730
PtWidgetIsRealized() 1732
PtWidgetMinimumSize() 1733
PtWidgetOffset() 1734
PtWidgetParent() 1736
PtWidgetPreferredSize() 1738
PtWidgetRid() 1740
PtWidgetSkip() 1741
PtWidgetToBack() 1743
PtWidgetToFront() 1746
PtWidgetTree() 1748
PtWidgetTreeTraverse() 1750
PtWidgetVisibleExtent() 1754
PtWindowConsoleSwitch() 1755
PtWindowGetFrameSize() 1756
PtWorkProcF_t, PtWorkProc_t 1758

14  Px—Extended 1759
PxConfigClose(), PxConfigCloseCx() 1762
PxConfigDeleteEntry(), PxConfigDeleteEntryCx() 1764
PxConfigDeleteSection(), PxConfigDeleteSectionCx() 1766
PxConfigFirstSection(), PxConfigFirstSectionCx() 1768
PxConfigForceEmptySection(), PxConfigForceEmptySectionCx() 1770
PxConfigNextEntry(), PxConfigNextEntryCx() 1772
PxConfigNextSection(), PxConfigNextSectionCx() 1774
PxConfigNextString(), PxConfigNextStringCx() 1776
PxConfigOpen(), PxConfigOpenCx() 1778
PxConfigReadBool(), PxConfigReadBoolCx() 1783
PxConfigReadChar(), PxConfigReadCharCx() 1785
PxConfigReadDouble(), PxConfigReadDoubleCx() 1787
PxConfigReadInt(), PxConfigReadIntCx() 1789
PxConfigReadLLong(), PxConfigReadLLongCx() 1792
PxConfigReadLong(), PxConfigReadLongCx() 1795
PxConfigReadShort(), PxConfigReadShortCx() 1798
PxConfigReadString(), PxConfigReadStringCx() 1801
PxConfigSection(), PxConfigSectionCx() 1803
PxConfigWriteBool(), PxConfigWriteBoolCx() 1805
PxConfigWriteChar() 1808
PxConfigWriteDouble(), PxConfigWriteDoubleCx() 1811
PxConfigWriteInt(), PxConfigWriteIntCx() 1814
PxConfigWriteLLong(), PxConfigWriteLLongCx() 1817
PxConfigWriteLong(), PxConfigWriteLongCx() 1820
PxConfigWriteShort(), PxConfigWriteShortCx() 1823
PxConfigWriteString(), PxConfigWriteStringCx() 1826
PxGetImageExtensions() 1829
PxLoadImage() 1831
PxTerminalBuildCharsets() 1840
PxTerminalLoadCharsets() 1842
PxTerminalSaveCharsets() 1844
PxTerminate() 1846
PxTranslateFromUTF() 1847
PxTranslateList() 1850
PxTranslateSet() 1852
PxTranslateStateFromUTF() 1855
15 Rt—Realtime 1865
RtTimerCreate() 1868
RtTimerDelete() 1870
RtTimerGetTime() 1871
RtTimerSetTime() 1873

16 utf8—UTF-8 Character 1875
utf8len() 1878
utf8strbstrlen() 1880
utf8strchr() 1882
utf8strichr() 1884
utf8strirchr() 1886
utf8strlen() 1888
utf8strnchr() 1890
utf8strncmp() 1892
utf8strndup() 1894
utf8strnichr() 1896
utf8strnlen() 1898
utf8strrchr() 1900
utf8towc() 1903

17 wc—Wide-Character 1905
wctolower() 1908
wctoutf8() 1910

Glossary 1913

Index 1935
List of Figures

A sample dialog displayed by `AppError()`. 146
Text justification relative to the indicated positions. 591
Predefined dither patterns. 704
Styles for capping lines. 735
Styles for joining lines. 745
A sample dialog displayed by `PtAlert()`. 1182
Numbering for Photon’s virtual consoles. 1327
An example of the dialog created by `PtFileSelection()`. 1414
A font-selection dialog. 1448
A sample dialog displayed by `PtNotice()`. 1548
A sample dialog displayed by `PtPrompt()`. 1581
About This Reference
What you’ll find in this guide

If you’re familiar with earlier versions of Photon, you should read:

- “What’s new in Photon for QNX Neutrino 6.3x”
- “What’s new in Photon for QNX Neutrino 6.3 Service Pack 1”
- “What’s new in Photon for QNX Neutrino 6.3”
- “What’s new in Photon for QNX Neutrino 6.2.1”
- “What’s new in Photon for QNX Neutrino 6.2.0”
- “What’s new in Photon for QNX Neutrino 6.1.0”
- “What’s new in Photon for QNX Neutrino 6.0”

to find out how the libraries have changed in this release.

The Photon Library Reference accompanies the Photon Development System and is intended for application developers. It describes the data types, structures, and functions that are defined in the Photon API.

This reference contains the following chapters:

- Summary of Entries
- Ab—PhAB-Generated Code
- Al—PhAB Translation
- Ap—PhAB
- utf8—UTF-8 Character
- Pd—Draw Context
- Pf—Font Server
- Pg—Graphics
- Ph—Photon
What you’ll find in this guide

- Pi—Images
- Pm—Memory
- Pp—Printing
- Pt—Widget Toolkit
- Px—Extended
- Rt—Realtime
- wc—Wide-Character
- Glossary

- For functions that deal with specific widgets, see the Widget Reference.
- For functions that deal with creating widget classes, see Building Custom Widgets.
- In general, the Photon libraries aren’t thread-safe. For information on using Photon and threads, see “Threads” in the Parallel Operations chapter of the Photon Programmer’s Guide.

To use the datatypes and functions in an application:

- If you’re using the Photon Application Builder (PhAB), the appropriate header files are automatically included in your application.
- If you aren’t using PhAB but your application uses widgets, include `<Pt.h>`.
- If you aren’t using PhAB or widgets, include `<Ph.h>`.
What’s new in Photon for QNX Neutrino 6.3x
New content in the docs
  - PhCancelDrag()

What’s new in Photon for QNX Neutrino 6.3 Service Pack 1
New content in the docs
  - PhTilesBoundingRect()

What’s new in Photon for QNX Neutrino 6.3
New content in the docs
  - AbGetABW()
  - ApSetContext()
  - PdGetOffscreenSurface()
  - PfAllocDetailsCx()
  - PfAllocRenderCx()
  - PfAssignDllCx()
  - PfAttachCx()
  - PfAttachDllCx()
  - PfAttachLocalDll()
  - PfAttachServerDll()
  - PfAttachSleuthMonitorDll()
  - PfConvertFontIDCx()
  - PfConvertPixelsToPointSizeCx()
What's new in Photon for QNX Neutrino 6.3

- PfDecomposeStemToIDCx()
- PfDefaultContext()
- PfDetachCx()
- PfDetachLocalDll()
- PfDynamicFontIDCx()
- PfDynamicLoadCx()
- PfDynamicUnloadCx()
- PfExtentComponentsCx()
- PfExtentCx()
- PfExtentTextCharPositionsCx()
- PfFindFontCx()
- PfFontBaseStemCx()
- PfFontDescriptionCx()
- PfFontFlagsCx()
- PfFontSizeCx()
- PfFontTypeCx()
- PfFreeFontCx()
- PfGenerateFontNameCx()
- PfGetGlyphIndexCx()
- PfGetOutlineCx()
- PfGlyphCx()
- PfLoadFontCx()
- PfLoadMetricsCx()
What's new in Photon for QNX Neutrino 6.3

- `PfQueryFontInfoCx()`  
- `PfQueryFontsCx()`  
- `PfRenderCx()`  
- `PfRestartServerDll()`  
- `PfSetOptionsDll()`  
- `PfSetRenderingDPICx()`  
- `PfWaitOnServerDll()`  
- `PgAlphaOffCx()`  
- `PgAlphaOnCx()`  
- `PgBevelBoxCx()`  
- `PgBlit()`  
- `PgBlitCx()`  
- `PgChromaOffCx()`  
- `PgChromaOnCx()`  
- `PgClearDrawBufferCx()`  
- `PgClearTranslationCx()`  
- `PgContextBlitCx()`  
- `PgContextBlitAreaCx()`  
- `PgContrastBevelBoxCx()`  
- `PgDefaultAlpha()`  
- `PgDefaultChroma()`  
- `PgDrawArcCx()`  
- `PgDrawArrowCx()`
What's new in Photon for QNX Neutrino 6.3

- PgDrawBevelBoxCx(), PgDrawIBevelBoxCx()
- PgDrawBeveledCx()
- PgDrawBezierCx(), PgDrawBezierv(), PgDrawBezierCxv()
- PgDrawBitmapCx(), PgDrawBitmapv(), PgDrawBitmapCxv()
- PgDrawEllipseCx()
- PgDrawGradientCx()
- PgDrawGradientBevelBoxCx()
- PgDrawGridCx()
- PgDrawGridCx()
- PgDrawImageCx(), PgDrawImagev(), PgDrawImageCxv()
- PgDrawLineCx(), PgDrawILineCx()
- PgDrawMultiTextAreaCx()
- PgDrawPhImageCx(), PgDrawPhImagev(), PgDrawPhImageCxv()
- PgDrawPhImageRectv(), PgDrawPhImageRectCxv()
- PgDrawPixelCx(), PgDrawIPixelCx()
- PgDrawPixelArrayCx(), PgDrawPixelArrayv(), PgDrawPixelArrayCxv()
- PgDrawPolygonCx(), PgDrawPolygonv(), PgDrawPolygonCxv()
- PgDrawRectCx(), PgDrawIRectCx()
- PgDrawPixelArrayCx(), PgDrawPixelArrayv(), PgDrawPixelArrayCxv()
- PgDrawRepImageCx(), PgDrawRepImagev(), PgDrawRepImageCxv()
What’s new in Photon for QNX Neutrino 6.3

- `PgDrawRepPhImageCx()`, `PgDrawRepPhImagev()`, `PgDrawRepPhImageCxv()`
- `PgDrawRoundRectCx()`
- `PgDrawSpanCx()`, `PgDrawSpanv()`, `PgDrawSpanCxv()`
- `PgDrawStringCx()`, `PgDrawStringv()`, `PgDrawStringCxv()`
- `PgDrawTextCx()`, `PgDrawTextv()`, `PgDrawTextCxv()`, `PgDrawTextCharsCx()`
- `PgDrawTextAreaCx()`
- `PgDrawTImageCx()`, `PgDrawTImageCxv()`, `PgDrawTImagev()`
- `PgDrawTrendCx()`, `PgDrawTrendCxv()`, `PgDrawTrendv()`
- `PgFlushCx()`, `PgFFlushCx()`
- `PgGetColorModelCx()`
- `PgGetGCCx()`
- `PgGetRegionCx()`
- `PgSetAlphaCx()`
- `PgSetAlphaBlendCx()`
- `PgSetChromaCx()`
- `PgSetClippingCx()`
- `PgSetColorModelCx()`
- `PgSetDrawBufferSizeCx()`
- `PgSetDrawModeCx()`
- `PgSetFillColorCx()`
- `PgSetFillDitherCx()`
- `PgSetFillTransPatCx()`
What's new in Photon for QNX Neutrino 6.3

- PgSetFillXORColorCx()
- PgSetFontCx()
- PgSetGCCx()
- PgSetMultiClipCx()
- PgSetPaletteCx()
- PgSetPlaneMaskCx()
- PgSetRegionCx()
- PgSetStrokeCapCx()
- PgSetStrokeColorCx()
- PgSetStrokeDashCx()
- PgSetStrokeDitherCx()
- PgSetStrokeJoinCx()
- PgSetStrokeTransPatCx()
- PgSetStrokeWidthCx(), PgSetStrokeFWidthCx()
- PgSetStrokeXORColorCx()
- PgSetTextColorCx()
- PgSetTextDitherCx()
- PgSetTextTransPatCx()
- PgSetTextXORColorCx()
- PgSetTranslationCx()
- PgSetUnderlineCx()
- PgSetUserClipCx(), PgSetUserClipAbsoluteCx()
- PgSwapDisplayCx()
What's new in Photon for QNX Neutrino 6.3

- `PgWaitVSyncCtx()`
- `PgMap_t`
- `PhClipboardHdr`
- `PhClipboardRead()`
- `PhClipboardWrite()`
- `PhRegionInfo()`
- `PtAppAddCallback()`
- `PtAppAddEventHandler()`
- `PtAppAddFilterCallback()`
- `PtAppAddHotkeyHandler()`
- `PtAppGetResource()`
- `PtAppGetResources()`
- `PtAppRemoveCallback()`
- `PtAppRemoveEventHandler()`
- `PtAppRemoveFilterCallback()`
- `PtAppRemoveHotkeyHandler()`
- `PtAppSetResource()`
- `PtAppSetResources()`
- `PtGetDndFetchIndex()`
- `PtWindowGetFrameSize()`
- `PxConfigCloseCx()`
- `PxConfigDeleteEntryCx()`
- `PxConfigDeleteSectionCx()`
What's new in Photon for QNX Neutrino 6.3

- PxConfigFirstSection(), PxConfigFirstSectionCx()
- PxConfigForceEmptySectionCx()
- PxConfigNextEntry(), PxConfigNextEntryCx()
- PxConfigNextSectionCx()
- PxConfigNextStringCx()
- PxConfigOpenCx()
- PxConfigReadBoolCx()
- PxConfigReadCharCx()
- PxConfigReadDoubleCx()
- PxConfigReadIntCx()
- PxConfigReadLLong(), PxConfigReadLLongCx()
- PxConfigReadLongCx()
- PxConfigReadShortCx()
- PxConfigReadStringCx()
- PxConfigSectionCx()
- PxConfigWriteBoolCx()
- PxConfigWriteCharCx()
- PxConfigWriteDoubleCx()
- PxConfigWriteIntCx()
- PxConfigWriteLLong(), PxConfigWriteLLongCx()
- PxConfigWriteLongCx()
- PxConfigWriteShortCx()
- PxConfigWriteStringCx()
What's new in Photon for QNX Neutrino 6.3

- `PxGetImageExtensions()`
- `PxTerminate()`

**Corrections, clarifications, and other changes**

`PdCreateOffscreenContext()` and `PdDupOffscreenContext()`

- Added several new flags for optimization. In `PdCreateOffscreenContext()`, `ImageType` now supports several new values.

`PdSetTargetDevice()`

- Now takes a `void *` instead of a `PhDrawContext_t` to specify the draw context.

`PgCreateLayerSurface()`

- Added several new flags for optimization.

`PgGetVideoModeInfo()`

- Returns a new flag in `PgVideoModeInfo_t`.
  - `mode_capabilities2`: `PgVM_MODE_CAP2_SCALED_BLIT`

`PxLoadImage()`

- Added support for SGI image format.

**Deprecated functions and data types**

- `PfRenderText()` — use `PfRender()` instead.
- `PfRenderWideText()` — use `PfRender()` instead.
- `PfFractionalRenderText()` — use `PfRender()` instead.
- `PfRenderCtx()` — use `PfRenderCx()` instead.
- `PfGetCacheStats()` — no longer required.
- `PgDrawPolygonmx()` — use `PgDrawPolygonv()` instead.
- `PgDrawPixelArraymx()` — use `PgDrawPixelArrayv()` instead.
What's new in Photon for QNX Neutrino 6.2.1

- `PgDrawSpanmx()` — use `PgDrawSpanv()` instead.
- `PgDrawBeziermx()` — use `PgDrawBezierv()` instead.
- `PgDrawTrendmx()` — use `PgDrawTrendv()` instead.
- `PgDrawBitmapmx()` — use `PgDrawBitmapv()` instead.
- `PgDrawImagemx()` — use `PgDrawImagev()` instead.
- `PgDrawPhImagemx()` — use `PgDrawPhImagev()` instead.
- `PgDrawPhImageRectmx()` — use `PgDrawPhImageRectv()` instead.
- `PgDrawTImagemx()` — use `PgDrawTImagev()` instead.
- `PgDrawRepTImagemx()` — use `PgDrawRepTImagev()` instead.
- `PgDrawRepBitmapmx()` — use `PgDrawRepBitmapv()` instead.
- `PgDrawRepImagemx()` — use `PgDrawRepImagev()` instead.
- `PgDrawTextmx()` — use `PgDrawTextv()` instead.
- `PgDrawStringmx()` — use `PgDrawStringv()` instead.
- `PtWindowFrameSize()` — use `PtWindowGetFrameSize()` instead.
- `PhClipboardCopy()` — use `PhClipboardWrite()` instead.
- `PhClipboardPasteStart()`, `PhClipboardPasteType()`, `PhClipboardPasteTypeN()`, `PhClipboardPasteFinish()` — use `PhClipboardRead()` instead.

What’s new in Photon for QNX Neutrino 6.2.1

New content in the docs

`ApModifyItemAccel()`

Modify the keyboard shortcut for a menu item
What's new in Photon for QNX Neutrino 6.2.1

*PgCreateDriverRegion()*
Create a region that’s owned by the graphics driver

*PgCreateLayerSurface()*
Create an offscreen context for a layer

*PgGetLayerCaps()*
Query the capabilities of a layer

*PgLayerCaps_t*
Capabilities for a layer

*PgLockLayer()*
Lock a layer for exclusive use by an application

*PgSetLayerArg()*
Configure a layer argument

*PgSetLayerSurface()*
Display the offscreen context on the specified layer surface

*PgUnlockLayer()*
Unlock a layer

*PhKeyEvent_t*
Added more information about interpreting key events.

*PhLibVersion()*
Get the version number of the Photon libraries

*PtFepCmd()*
Control a Front-End Processor (FEP) from an application

*PtHelpSearch()*
New value for the *method* argument, HELP_SEARCH_METHOD_SUBSTRING_CASE.
What's new in Photon for QNX Neutrino 6.2.0

*PxLoadImage()*

Added a section on threads and *PxLoadImage()*.

**Deprecated functions and data types**

- *PtConnectionWaitForId()* — this function is unnecessary.

**Corrections**

*ApCreateModule()*

Corrected the description of creating a picture module.

*PdSetTargetDevice()*

Corrected the prototype.

*PhEventRead()*

Corrected the description and example.

**What’s new in Photon for QNX Neutrino 6.2.0**

**New content in the docs**

**Ap**

*ApAddContext()*

Add a PhAB context so you can use a PhAB application as a DLL.

*ApRemoveClass()*

Remove a widget class

*ApRemoveContext()*

Remove the PhAB context from a PhAB application that you’re using as a DLL.
What's new in Photon for QNX Neutrino 6.2.0

**Pd**

*PdCreateOffscreenLock()*
Create an offscreen lock

*PdDestroyOffscreenLock()*
Destroy an offscreen lock

*PdIsOffscreenLocked()*
Determine whether or not an offscreen context is locked

*PdLockOffscreen()*
Lock an offscreen context

*PdUnlockOffscreen()*
Unlock an offscreen context

**Pf**

*PfExtent(), PfExtentCx()*
Calculate the extent rectangle of a text string

*PfExtentTextCharPositionsCx()*
Calculate individual character positions, specifying a font context

*PfExtentWideText()*

The prototype has changed; the string is now of type `uint16_t*`.

*PfFindFont(), PfGenerateFontName()*
The prototypes have changed; they now use `char` instead of `uchar_t`.

*PfGenerateFontNameCx()*
Generate a font name.

*PfGetOutlineCx()*
Get individual point information for a glyph outline, specifying the font context.
What's new in Photon for QNX Neutrino 6.2.0

**Pf**

PfRenderCtx(), PfRenderCx()
Render a string via a user callback function

PfRenderWideText(), PFWideTextWidthBytes(), PFWideTextWidthChars()
The prototype has changed; the string is now of type `uint16_t`.

**Pg**

PgAlphaValue()
Extract the alpha component from a color value

PgARGB()
Convert alpha, red, green, and blue values to composite color format

PgContextBlit(), PgContextBlitArea()
The source data is now scaled to fit the destination rectangle.

PgGetColorModel()
Get the current color model

PgSetColorModel()
Set the current color model

**Ph**

PhImage_t
New image types:
- Pg_IMAGE_DIRECT_1555
- Pg_IMAGE_DIRECT_4444

**Pp**

PpPrintContext_t
Data structure describing a print context
What's new in Photon for QNX Neutrino 6.2.0

Pt

PtCreateClassStyle()
Create a class style

PtDupClassStyle()
Get a copy of a widget class style

PtFindClassStyle()
Find the style with a given name

PtGetStyleMember()
Get a member of a style

PtGetWidgetStyle()
Get the style that a widget is currently using

PtSetClassStyleMethods()
Set multiple members of a style from an array

PtSetStyleMember()
Set a member of a style

PtSetStyleMembers()
Set multiple members of a style from a variable-length argument list

PtSetWidgetStyle()
Set the current style for a widget

utf8

utf8len() Count the bytes in a UTF-8 character

utf8strblen() Find the number of UTF-8 characters in part of a string

utf8strchr() Search for a UTF-8 character in a string

utf8strichr() Search for a UTF-8 character in a string, ignoring case
What's new in Photon for QNX Neutrino 6.2.0

utf8strirchr() Search backwards for a UTF-8 character in a string, ignoring case
utf8strlen() Find the length of a UTF-8 character string
utf8strmchr() Search for a UTF-8 character in part of a string
utf8strncmp() Compare part of a UTF-8 character string
utf8strndup() Create a copy of part of a UTF-8 character string
utf8strnchr() Search for a UTF-8 character in part of a string, ignoring case
utf8strnlen() Find the number of bytes used by $n$ characters of a UTF-8 character string
utf8strrchr() Search backwards for a UTF-8 character in a string
utf8towc() Convert a UTF-8 character to a wide-character code

wc

wctou8() Convert a wide-character code into a UTF-8 character

Deprecated functions and data types
- Pg_MIX_COLOR — use dithering instead.
- PhRegionGetData() — use PhRegionQuery() and PhRegionDataFindType() instead.
- PtAppRemoveSignalProc() — use PtAppRemoveSignal() instead.

The mbstr* functions have been replaced:

<table>
<thead>
<tr>
<th>Instead of:</th>
<th>Use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbstrblen()</td>
<td>utf8strblen()</td>
</tr>
</tbody>
</table>

continued...
What's new in Photon for QNX Neutrino 6.2.0

### Instead of: Use:

<table>
<thead>
<tr>
<th>Instead of</th>
<th>Use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbstrchr()</td>
<td>utf8strchr()</td>
</tr>
<tr>
<td>mbstrichr()</td>
<td>utf8strichr()</td>
</tr>
<tr>
<td>mbstrirchr()</td>
<td>utf8strirchr()</td>
</tr>
<tr>
<td>mbstrlen()</td>
<td>utf8strlen()</td>
</tr>
<tr>
<td>mbstrnchr()</td>
<td>utf8strnchr()</td>
</tr>
<tr>
<td>mbstrncmp()</td>
<td>utf8strncmp()</td>
</tr>
<tr>
<td>mbstrndup()</td>
<td>utf8strndup()</td>
</tr>
<tr>
<td>mbstrnichr()</td>
<td>utf8strnichr()</td>
</tr>
<tr>
<td>mbstrnlen()</td>
<td>utf8strnlen()</td>
</tr>
<tr>
<td>mbstrrchr()</td>
<td>utf8strrchr()</td>
</tr>
</tbody>
</table>

### Corrections, clarifications, and other changes

**ApAddClass()**  
This function now makes a copy of the string pointed to by `class_name_string`; it no longer keeps a pointer to the string.

**PdGetOffscreenContextPtr()**  
This function can fail on certain hardware. You can use `PdGetOffscreenContextPtr()` on closed systems where you know that the graphics frame buffer is linear; don’t use it in applications that target generic hardware configurations.

**PfConvertFontID()**  
Now returns `char *` instead of `uchar_t *`.

**PfFontDescription()**  
Now returns `char const *` instead of `uchar_t const *`. 
**What's new in Photon for QNX Neutrino 6.2.0**

*PfQueryFonts()* If you don’t want to limit the search to fonts that support a specific character, pass `PHFONT_ALL_SYMBOLS` as the `symbol` argument.

*PfGetOutline()* The `pkucFont` argument is now of type `char const *` instead of `uchar_t const *`.

*PhEmit(), PhEmitmx(), PhEventEmit(), PhEventEmitmx()* If you set the collector ID for the event (`event->collector.rid`) to zero, the event is enqueued to every appropriately sensitive region that intersects with the event. If you set `collector.rid` to a region ID, only that region notices the event.

*PpPrintWidget()* `Pt_PP_RESIZE_PC` is a new value for the `resize` argument.

*PpSetPC()* Corrected the data types for `Pp_PC_INTENSITY`, `Pp_PC_PRINTER_RESOLUTION`, `Pp_PC_SOURCE_COLORS`, and `Pp_PC_SOURCE_RESOLUTION`.

*PtAddClassStyle()* Corrected the prototype.

*PtQuitMainLoop()* This function now returns an `int` that indicates whether or not the thread has already called `PtQuitMainLoop()`.

*PtSpawn()* By default, the new process inherits all of the parent’s valid file descriptors whose values are less than or equal to 9.

*PxLoadImage()* You don’t have to define the file formats that you want to support any more (e.g. `PX_GIF_SUPPORT`).
What’s new in Photon for QNX Neutrino 6.1.0

Patch A

New entries:
- `PgDrawArrow()`

New content in the docs
- `PdSetOffscreenTranslation()`
- `PfFontBaseStem()`
- `PgDrawPhImage()`
- `PgDrawRepPhImage()`
- `PgGetRegion()`
- `PhMakeTransparent()`
- `PtAllowExit()`
- `PtPassword()`
- `PtPreventExit()`
- `PtPrintPropSelect()`

Deprecated functions and data types
- `PdOpenGLContextSetRid()`
- `PdOpenGLContextSwapBuffers()`
- `PgCreateGradient()`
- `PgGetGradientColor()`
- `PgGradientControl_t`
- `PgGradientLinear()`
- `PgGradientMakeImage()`
What's new in Photon for QNX Neutrino 6.0

- `PgGradientPercent()`
- `PgGradientRotatePalette()`

Other changes

`PhWindowQueryVisible()`

`Ph_QUERY_CONSOLE` and `Ph_QUERY_WORKSPACE` are new bits for the `flags` argument. `Ph_QUERY_CONSOLE` is the default value.

What’s new in Photon for QNX Neutrino 6.0

New functionality

Font names

An API for handling font names in an organized, portable way has been added. Previously, we recommended you address a font by the “stem name” supplied by the font manager (e.g. `helv12`). This approach is no longer recommended, since stem names aren’t guaranteed to be static. This is especially true when new font technology is added to the Photon font subsystem. The new API provides an interface that eliminates the need to recode in the future. It includes:

- `PfConvertFontID()`
- `PfFindFont()`
- `PfFontDescription()`
- `PfFontFlags()`
- `PfFontSize()`
- `PfFreeFont()`
New content in the docs

AI—PhAB Translation

These new functions support language and message databases:

- AlClearTranslation()
- AlCloseDBase()
- AlGetEntry()
- AlGetSize()
- AlOpenDBase()
- AlReadTranslation()
- AlSaveTranslation()
- AlSetEntry()

Ap—PhAB

New:

- ApCloseMessageDB()
- ApCreateDBWidget()
- ApCreateDBWidgetFamily()
- ApDeleteDBWidget()
- ApGetDBWidgetInfo()
- ApGetMessage()
- ApLoadMessageDB()
- ApModalWait()

These functions have been renamed; you should use the new name, although applications that use the old name will still work:

- ApCopyWidget() — now called ApCopyDBWidget()
What's new in Photon for QNX Neutrino 6.0

Deprecated:

- `ApBitmap_t`
- `ApDeleteWidget()` — you can use `ApDeleteDBWidget()` instead, but note that it deletes the given widget and its children.
- `ApFreeBitmapRes()`
- `ApGetBitmapRes()`

The prototype has changed for:

- `ApCreateModule()`

**mbstr—Multibyte-Character**

New:

- `mbstrichr()` — see `utf8strichr()`
- `mbstrirchr()` — see `utf8strirchr()`
- `mbstrndup()` — see `utf8strndup()`
- `mbstrnichr()` — see `utf8strnichr()`

**Pd—Draw Context**

New:

- `PdCreateDirectContext()`
- `PdCreateOffscreenContext()`
- `PdDirectStart()`
- `PdDirectStop()`
- `PdDupOffscreenContext()`
- `PdGetDevices()`
- `PdGetOffscreenContextPtr()`
What’s new in Photon for QNX Neutrino 6.0

- `PdOffscreenContext_t`
- `PdReleaseDirectContext()`
- `PdSetTargetDevice()`

**Pf—Font Server**

New:
- `PfConvertFontID()`
- `PfDecomposeStemToID()`
- `PfDynamicLoad()`
- `PfDynamicUnload()`
- `PfExtentFractTextCharPositions()`
- `PfExtentTextCharPositions()`
- `PfExtentTextToRect()`
- `PfExtentWideText()`
- `PfFindFont()`
- `PfFontDescription()`
- `PfFontFlags()`
- `PfFontSize()`
- `PfFreeFont()`
- `PfGenerateFontName()`
- `PfGetCacheStats()`
- `PfGetOutline()`
- `PfQueryFontInfo() — replaces PfQueryFont()`
- `PfRenderWideText()`
What’s new in Photon for QNX Neutrino 6.0

- PfTextWidthBytes()
- PfTextWidthChars()
- PfWideTextWidthBytes()
- PfWideTextWidthChars()

Deprecated:
- PfQueryFont() — replaced by PfQueryFontInfo(), which returns a valid stem name when using a scalable font. The old function is still in the library, so applications that use it should still work.

Pg—Graphics

New:
- PgAlphaOff()
- PgAlphaOn()
- PgBevelBox()
- PgCalcColorContrast()
- PgChromaOff()
- PgChromaOn()
- PgConfigScalerChannel()
- PgContextBlit()
- PgContextBlitArea()
- PgContrastBevelBox()
- PgCreateVideoChannel()
- PgDestroyVideoChannel()
- PgDrawGradient()
- PgDrawGradientBevelBox()
What’s new in Photon for QNX Neutrino 6.0

- PgDrawMultiTextArea()
- PgDrawPhImageRec()mx
- PgDrawRepPhImage()mx
- PgExtentMultiText()
- PgGetGraphicsHWCaps()
- PgGetOverlayChromaColor()
- PgGetScalerCapabilities()
- PgGetVideoMode()
- PgGetVideoModeInfo()
- PgGetVideoModeList()
- PgNextVideoFrame()
- PgScalerCaps_t
- PgScalerProps_t
- PgReadScreen()
- PgReadScreenSize()
- PgSetAlpha()
- PgSetAlphaBlend()
- PgSetChroma()
- PgSetDPMSMode()
- PgSetVideoMode()
- PgSwapDisplay()
- PgVideoChannel_t
- PgWaitDrawComplete()
What’s new in Photon for QNX Neutrino 6.0

- PgWaitHWIdle()
- PgWaitVSync()

Other new features:

- PgSetDrawMode()

  New drawing modes.

Ph—Photon

New:

- PhAllocPackType()
- PhAreaToRect()
- PhCreateImage()
- PhCreateTransportCtrl()
- PhDCCreate()
- PhDCRelease()
- PhDeTranslateRect()
- PhEmit() — similar to PhEventEmit(), but with a cleaner API
- PhEmitmx() — similar to PhEventEmitmx(), but with a cleaner API
- PhFindTransportType()
- PhFreeTransportType()
- PhGetAllTransportHdrs()
- PhGetNextInlineData()
- PhGetNextTransportHdr()
- PhGetTransportHdr()
- PhGetTransportVectors()
What's new in Photon for QNX Neutrino 6.0

- `PhInputGroup()`
- `PhLinkTransportData()`
- `PhLocateTransHdr()`
- `PhMallocUnpack()`
- `PhPackEntry()`
- `PhPackType()`
- `PhRectIntersect()` — replaces `PtRectIntersect()`
- `PhRectToArea()`
- `PhRectUnion()` — replaces `PtRectUnion()`
- `PhRegionGetData()`
- `PhRegisterTransportType()`
- `PhReleaseTransportCtrl()`
- `PhReleaseTransportHdrs()`
- `PhTranslateRect()` — replaces `PtTranslateRect()`
- `PhTransportFindLink()`
- `PhTransportCtrl_t`
- `PhTransportLink_t`
- `PhTransportRegEntry_t`
- `PhTransportType()`
- `PhUnlinkTransportHdr()`
- `PhUnpack()`

Other changes:
What's new in Photon for QNX Neutrino 6.0

**PhEvent_t** Ph_EV_BUT_RELEASE events have a new subtype, Ph_EV_RELEASE_OUTBOUND.

**PhInitDrag()** The prototype has changed; the ptrpos and cursor arguments have been added.

**PhMakeTransBitmap()**

This function now supports all image types currently supported by Photon. The meaning of the trans_color argument depends on the image type.

**Pi—Images**

These new functions support image operations:

- *PiCropImage()*
- *PiDuplicateImage()*
- *PiFlipImage()*
- *PiGetPixel()*
- *PiGetPixelFromArray()*
- *PiGetPixelRGB()*
- *PiSetPixel()*
- *PiSetPixelIntoData()*

**Pp—Printing**

New functions:

- *PpContinueJob()* — replaces *PpPrintStart()*
- *PpCreatePC()* — replaces *PpPrintCreatePC()*
- *PpEndJob()* — replaces *PpPrintClose()*
- *PpFreePrinterList()*

Ixvi About This Reference
What’s new in Photon for QNX Neutrino 6.0

- `PpGetCanvas()`
- `PpGetPC()` — replaces `PpPrintGetPC()`
- `PpLoadDefaultPrinter()`
- `PpLoadPrinterList()`
- `PpReleasePC()` — replaces `PpPrintReleasePC()`
- `PpSetCanvas()`
- `PpSetPC()` — replaces `PpPrintSetPC()`
- `PpStartJob()` — replaces `PpPrintOpen()`
- `PpSuspendJob()` — replaces `PpPrintStop()`

Other changes:

`PpPrintWidget()`

You no longer need to call `PtFlush()` after calling this function.

**Pt—Widget toolkit**

New:

- `PtAddClassStyle()`
- `PtAddFilterCallback()`
- `PtAddFilterCallbacks()`
- `PtAddResponseType()`
- `PtAlert()`
- `PtAppSetFdMode()`
- `PtBlit()`
- `PtBlockAllWindows()`
- `PtBlockWindow()`
What’s new in Photon for QNX Neutrino 6.0

- PtCalcCanvas()
- PtCalcSurface()
- PtCalcSurfaceByAction()
- PtCalcSurfaceById()
- PtCancelDnd()
- PtCheckSurfaces()
- PtClippedBlit()
- PtCondTimedWait()
- PtCondWait()
- PtConnectionAddEventHandler()
- PtConnectionAddMsgHandler()
- PtConnectionClientGetUserData()
- PtConnectionClientSetError()
- PtConnectionClientSetUserData()
- PtConnectionFindId()
- PtConnectionFlush()
- PtConnectionFindName()
- PtConnectionNotify()
- PtConnectionReply(), PtConnectionReplymx()
- PtConnectionResizeEventBuffer()
- PtConnectionSend(), PtConnectionSendmx()
- PtConnectionServerDestroy()
- PtConnectionServerGetUserData()
What’s new in Photon for QNX Neutrino 6.0

- PtConnectionServerSetError()
- PtConnectionServerSetUserData()
- PtConnectionTmpName()
- PtConnectionWaitForName()
- PtConnectorCreate()
- PtConnectorDestroy()
- PtConnectorGetId()
- PtCRC() — replaces PxCRC()
- PtCRCValue()
- PtCreateActionSurface()
- PtCreateSurface()
- PtCreateTransportCtrl()
- PtDamageSurface(), PtDamageSurfaceById()
- PtDamageSurfaceByAction()
- PtDestroyAllSurfaces()
- PtDestroySurface()
- PtDestroySurfaceById()
- PtDisableSurface(), PtDisableSurfaceById()
- PtDisableSurfaceByAction()
- PtDndFetch_t
- PtDndSelect()
- PtEnableSurface(), PtEnableSurfaceById()
- PtEnableSurfaceByAction()
What's new in Photon for QNX Neutrino 6.0

- PtEnter()
- PtExit()
- PtFindFocusNextFrom()
- PtFindFocusPrevFrom()
- PtFindSurface()
- PtFindSurfaceByAction()
- PtGetResource()
- PtGiveFocus()
- PtHelpQuit() — replaces PxHelpQuit()
- PtHelpSearch() — replaces PxHelpSearch()
- PtHelpTopic() — replaces PxHelpTopic()
- PtHelpTopicRoot() — replaces PxHelpTopicRoot()
- PtHelpTopicTree() — replaces PxHelpTopicTree()
- PtHelpUrl() — replaces PxHelpUrl()
- PtHelpByUrlRoot() — replaces PxHelpByUrlRoot()
- PtHideSurface(), PtHideSurfaceById()
- PtHideSurfaceByAction()
- PtInitDnd()
- PtInsertSurface(), PtInsertSurfaceById()
- PtLeave()
- PtMakeModal()
- PtModalBlock()
- PtModalUnblock()
What's new in Photon for QNX Neutrino 6.0

- PtNotice()
- PtPrintSelect()
- PtPrompt()
- PtPulseArm()
- PtQuitMainLoop()
- PtReleaseTransportCtrl()
- PtRemoveFilterCallback()
- PtRemoveFilterCallbacks()
- PtSetAreaFromCanvas()
- PtSetResource()
- PtShowSurface(), PtShowSurfaceById()
- PtShowSurfaceByAction()
- PtSurfaceActionId()
- PtSurfaceAddData(), PtSurfaceAddDataById()
- PtSurfaceBrotherBehind()
- PtSurfaceBrotherInFront()
- PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById()
- PtSurfaceExtent(), PtSurfaceExtentById()
- PtSurfaceGetData(), PtSurfaceGetDataById()
- PtSurfaceHit()
- PtSurfaceId()
- PtSurfaceInBack()
- PtSurfaceInFront()
What's new in Photon for QNX Neutrino 6.0

- `PtSurfaceIsDisabled()`
- `PtSurfaceIsEnabled()`
- `PtSurfaceIsHidden()`
- `PtSurfaceIsShown()`
- `PtSurfaceRect()`, `PtSurfaceRectById()`
- `PtSurfaceRemoveData()`, `PtSurfaceRemoveDataById()`
- `PtSurfaceTestPoint()`
- `PtSurfaceToBack()`, `PtSurfaceToBackById()`
- `PtSurfaceToFront()`, `PtSurfaceToFrontById()`
- `PtTransportCtrl_t`
- `PtTransportRequestable()`
- `PtTransportType()`
- `PtUnblockWindows()`
- `PtWidgetActiveSurface()`
- `PtWidgetMinimumSize()`
- `PtWidgetPreferredSize()`
- `PtWindowFrameSize()`

Deprecated:

- `PtAskQuestion()`
- `PtBasicWidgetCanvas()`, `PtLabelWidgetCanvas()`, `PtWidgetCanvas()` — use `PtCalcCanvas()` instead.
- `PtDeTranslateRect()` — use `PhDeTranslateRect()`
- `PtFrameSize()` — use `PtWindowGetFrameSize()`
What’s new in Photon for QNX Neutrino 6.0

- `PtPulseArmFd()`, `PtPulseArmPid()` — use `PtPulseArm()` instead.
- `PtPulseDeliver()` — use `MsgDeliverEvent()` instead (see the QNX Neutrino Library Reference).
- `PtPulseDisarm()` — you don’t need this under QNX Neutrino.
- `PtSetAreaFromExtent()` — use `PhRectToArea()`
- `PtSetAreaFromWidgetCanvas()` — use `PtSetAreaFromCanvas()`
- `PtSyncPhoton()` — use `PtExit()` instead.
- `PtTranslateRect()` — use `PhTranslateRect()`

These functions have been renamed; you should use the new name, although applications that use the old name will still work:

- `PtReParentWidget()` — now called `PtReparentWidget()`

Other changes:

**PtCreateWidget()**

The `parent` argument has changed. It can now be a pointer to the parent widget or one of:

- `Pt_DEFAULT_PARENT` — use the default parent, which is the most recently created container.
- `Pt_NO_PARENT`

**PtFileSelection()**

This function can select directories as well as files. Enable directory selection with the `Pt_FSDIALOG_SELECT_DIRS` flag. Existing directories can be selected with `btn1` (the Open button).

`PtFileSelection()` can create and delete directories and delete files. You can create new directories at any time by pressing the New button. When the `PtFileSel` widget has focus, two new hotkeys...
are activated: the Insert key creates a new directory just like the New Directory button, and the Delete key removes the currently selected item.

The *info* structure has the following new members:

- `user_data`
- `confirm_display`
- `confirm_selection`
- `new_directory`

New flags:

- `Pt_FSR_NO_FCHECK` — replaces `Pt_FSDIALOG_NO_FCHECK`
- `Pt_FSR_NO_FSPEC` — replaces `Pt_FSDIALOG_NO_FSPEC`
- `Pt_FSR_NO_UP_BUTTON` — replaces `Pt_FSDIALOG_NO_UP_BUTTON`
- `Pt_FSR_NO_NEW`
- `Pt_FSR_NO_NEW_BUTTON`
- `Pt_FSR_NO_SELECT_FILES`
- `Pt_FSR_SELECT_DIRS`
- `Pt_FSR_CREATE_PATH`
- `Pt_FSR_NO_CONFIRM_CREATE_PATH`
- `Pt_FSR_NO_DELETE`
- `Pt_FSR_NO_CONFIRM_DELETE`
- `Pt_FSR_RECURSIVE_DELETE`
- `Pt_FSRCONFIRM_EXISTING`

`PtModalEnd()` The prototype has changed; this function no longer takes as an argument the value returned by `PtModalStart()`.

`PtModalStart()` No longer returns anything.

`PtRectIntersect()` Replaced by `PhRectIntersect()` —

`PtRectUnion()` Replaced by `PhRectUnion()` —
What's new in Photon for QNX Neutrino 6.0

Px—Extended

 PxCRC() Replaced by PtCRC().
 PxHelpQuit() Replaced by PtHelpQuit()
 PxHelpSearch() Replaced by PtHelpSearch()
 PxHelpTopic() Replaced by PtHelpTopic()
 PxHelpTopicRoot() Replaced by PtHelpTopicRoot()
 PxHelpTopicTree() Replaced by PtHelpTopicTree()
 PxHelpUrl() Replaced by PtHelpUrl()
 PxHelpUrlRoot() Replaced by PtHelpUrlRoot()
 PxLoadImage() If PX_TRANSPARENT is set in the flags member of the PxMethods_t structure, PxLoadImage() makes the image transparent, using the detected transparent color and the image’s chroma scheme. There’s no need to set the transparent member of this structure — it’s deprecated.
 PxTranslateUnknown() Control how unknown encodings are handled

You no longer need to include <photon/PxHelp.h> when using the help functions.

Rt—Realtime

 New:
 • RtTimerCreate()
 • RtTimerDelete()
 • RtTimerGetTime()
 • RtTimerSetTime()
Typographical conventions

Corrections
Pg—Graphics

\textit{PgDrawImage()}, \textit{PgDrawImagemx()}

Instead of using this function, we recommend using a \textit{PhImage_t} structure and calling \textit{PgDrawPhImagemx()}.  

\textit{PgDrawTImage()}, \textit{PgDrawTImagemx()}

Instead of using this function, we recommend using a \textit{PhImage_t} structure and calling \textit{PgDrawPhImagemx()}.  

Ph—Photon

\textit{PhMakeTransBitmap()}

Use \textit{PgDrawPhImagemx()} to draw a transparent image.  

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications. The following table summarizes our conventions:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code examples</td>
<td>\texttt{if ( stream == NULL )}</td>
</tr>
<tr>
<td>Command options</td>
<td>\texttt{-lR}</td>
</tr>
<tr>
<td>Commands</td>
<td>\texttt{make}</td>
</tr>
<tr>
<td>Environment variables</td>
<td>\texttt{PATH}</td>
</tr>
<tr>
<td>File and pathnames</td>
<td>\texttt{/dev/null}</td>
</tr>
<tr>
<td>Function names</td>
<td>\texttt{exit()}</td>
</tr>
<tr>
<td>Keyboard chords</td>
<td>\texttt{Ctrl-Alt-Delete}</td>
</tr>
</tbody>
</table>

\textit{continued…}
Typographical conventions

<table>
<thead>
<tr>
<th>Reference</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard input</td>
<td><em>something you type</em></td>
</tr>
<tr>
<td>Keyboard keys</td>
<td>Enter</td>
</tr>
<tr>
<td>Program output</td>
<td><em>login:</em></td>
</tr>
<tr>
<td>Programming constants</td>
<td>NULL</td>
</tr>
<tr>
<td>Programming data types</td>
<td><em>unsigned short</em></td>
</tr>
<tr>
<td>Programming literals</td>
<td><em>0xFF, &quot;message string&quot;</em></td>
</tr>
<tr>
<td>Variable names</td>
<td><em>stdin</em></td>
</tr>
<tr>
<td>User-interface components</td>
<td><em>Cancel</em></td>
</tr>
</tbody>
</table>

We use an arrow (→) in directions for accessing menu items, like this:

> You’ll find the **Other...** menu item under
                  **Perspective→Show View.**

We use notes, cautions, and warnings to highlight important messages:

---

**Notes** point out something important or useful.

---

**CAUTION:** Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

---

**WARNING:** Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.
Technical support

Note to Windows users

In our documentation, we use a forward slash (/) as a delimiter in all pathnames, including those pointing to Windows files.

We also generally follow POSIX/UNIX filesystem conventions.

Technical support

To obtain technical support for any QNX product, visit the Support + Services area on our website (www.qnx.com). You’ll find a wide range of support options, including community forums.
Chapter 1
Summary of Entries

In this chapter...

Alpha blending 4
Background processing 4
Bitmaps and Images 4
Blitting 7
Characters, translating 7
Chroma key operations 8
Clipboard operations 8
Clipping 9
Colors, converting and parsing 9
Configuration files 10
Connections to other applications 12
Coordinates, translating 14
Cursors/pointers 14
Data chains 15
Direct mode 15
Drag and drop 16
Dragging 19
Draw contexts 19
Drawing attributes 19
Events 22
Font handling 25
Geometry 27
Gradients 28
Graphical contexts 29
Input/Output events 29
Interprocess Communication (IPC) 30
Layers 31
Key events, translating 32
Memory contexts 32
Summary of Entries

Messages and questions 32
Modal dialogs 33
Modules 34
Online help 34
Photon Application Builder functions 35
Photon services, connecting and disconnecting 36
Power-saving modes 36
Primitive drawing routines 36
Printing 38
Processes 40
Realtime timers 40
Regions 41
Shared memory 41
Signals 42
Strings, translating 42
Synchronization 43
System information 44
Text 44
Threads 45
Tiles 45
UTF-8 character strings 46
Video modes 47
Video offscreen memory 47
Video overlay 48
Wide characters 49
Widgets 49
Window Manager 85
This chapter groups the datatypes and functions according to their purpose. You can use this chapter to determine what you need to perform a task.

The first two letters of a datatype’s or function’s name identify the chapter in which it’s described, as follows:

<table>
<thead>
<tr>
<th>If prefix is:</th>
<th>See the following chapter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ab</td>
<td>PhAB-Generated Code</td>
</tr>
<tr>
<td>Al</td>
<td>PhAB Translation</td>
</tr>
<tr>
<td>Ap</td>
<td>PhAB</td>
</tr>
<tr>
<td>utf8</td>
<td>UTF-8Character</td>
</tr>
<tr>
<td>Pd</td>
<td>Draw Context</td>
</tr>
<tr>
<td>Pf</td>
<td>Font Server</td>
</tr>
<tr>
<td>Pg</td>
<td>Graphics</td>
</tr>
<tr>
<td>Ph</td>
<td>Photon</td>
</tr>
<tr>
<td>Pi</td>
<td>Images</td>
</tr>
<tr>
<td>Pm</td>
<td>Memory</td>
</tr>
<tr>
<td>Pp</td>
<td>Printing</td>
</tr>
<tr>
<td>Pt</td>
<td>Widget Toolkit</td>
</tr>
<tr>
<td>Px</td>
<td>Extended</td>
</tr>
<tr>
<td>Rt</td>
<td>Realtime</td>
</tr>
<tr>
<td>wc</td>
<td>Wide-Character</td>
</tr>
</tbody>
</table>
Alpha blending

- For datatypes and functions that deal with specific widgets, see the Widget Reference.
- For functions that deal with creating widget classes, see Building Custom Widgets.

**Alpha blending**

- *PgAlphaOff*() Turn alpha blending operations off
- *PgAlphaOn*() Turn alpha blending operations on
- *PgSetAlpha*() Set the parameters for alpha blending in detail
- *PgSetAlphaBlend*() Set the parameters for alpha blending simply

**Background processing**

- *PtAppAddWorkProc()* Add a WorkProc (background) function
- *PtAppRemoveWorkProc()* Remove a WorkProc processing function
- *PtWorkProcF_t, PtWorkProc_t* Type for defining a work procedure function

**Bitmaps and Images**

- *PgDrawBitmap*() Draw a bitmap
- *PgDrawImage*() Draw an image
Bitmaps and Images

*PgDrawPhImage*()

Draw an image that’s contained in a PhImage_t structure

*PgDrawPhImageRect*()

Draw part of an image that’s contained in a PhImage_t structure

*PgDrawTI mage*()

Draw an image with a transparency mask

*PgDrawRepBitmap*()

Draw a bitmap several times

*PgDrawRepImage*()

Draw an image several times

*PgDrawRepPhImage*()

Repeatedly draw an image stored in a PhImage_t structure

*PgReadScreen*()  
Read an image from the screen

*PgReadScreenSize*()  
Determine the memory requirements for reading an image from the screen

*PhCreateImage*()

Create a new PhImage_t structure

*PhImage_t*  
Data and characteristics of an image

*PhMakeGhostBitmap*()

Create a ghost bitmap for an image

*PhMakeTransBitmap*()

Create a transparency mask for an image
**Bitmaps and Images**

*PhMakeTransparent()*

Make a given color transparent in an image, using chroma if possible

*PhReleaseImage()*

Release allocated members of an image structure

*PiCropImage()*

Crop an image to the specified boundary

*PiDuplicateImage()*

Duplicate an image

*PiFlipImage()*

Flip all or part of an image

*PiGetPixel()*

Retrieve the value of a pixel within an image

*PiGetPixelFromData()*

Retrieve a value from a run of pixels

*PiGetPixelRGB()*

Retrieve the RGB value of a pixel within an image

*PiSetPixel()*

Alter the value of a pixel within an image

*PiSetPixelInData()*

Set the value of a pixel in a run of pixels

*PtCRC()*

Calculate a CRC for a block of data

*PtCRCValue()*

Calculate a running CRC checksum

*PxGetImageExtensions()*

Get a list of supported image file extensions

*PxLoadImage()*

Read or query a graphic file
Blitting

- **PhBlit()**: Blit an area within a region
- **PgBlit()**: Blit an area within the region set for the current draw context
- **PgBlitCx()**: Blit an area within the region set for a specified draw context
- **PtBlit()**: Blit an area within a widget
- **PtClippedBlit()**: Blit areas within a widget, with clipping

Characters, translating

- **PtFepCmd()**: Control a Front-End Processor (FEP) from an application
- **PxTerminalBuildCharsets()**: Create character set tables based on translation tables
- **PxTerminalLoadCharsets()**: Load character-set information from a file
- **PxTerminalSaveCharsets()**: Save character-set information in a file
- **PxTranslateFromUTF()**: Translate characters from Unicode UTF-8
- **PxTranslateList()**: Create a list of all supported character translations
- **PxTranslateSet()**: Install a new character set translation
- **PxTranslateStateFromUTF()**: Translate characters from UTF-8, using an internal state buffer
Chroma key operations

PxTranslateStateToUTF()
Translates characters to UTF-8, using an internal state buffer

PxTranslateToUTF()
Translates characters to UTF-8

PxTranslateUnknown()
Controls how unknown encodings are handled

Chroma key operations

PgChromaOff*()
Turns chroma key operations off

PgChromaOn*()
Turns chroma operations on

PgSetChroma()
Sets the chroma color and operation

Clipboard operations

PhClipboardRead()
Reads data from the clipboard

PhClipboardWrite()
Copies data to the clipboard

PhClipboardCopyString()
Copies string-only data to the clipboard

PhClipboardPasteString()
Pastes string-only data from the clipboard

PhClipboardHdr
Clipbard header structure
Clipping

*PgSetClipping*( )
Limit the extent of drawing

*PgSetMultiClip*( )
Set a list of rectangles to clip drawing

*PgSetUserClip*( )
Restrict subsequent draws

*PgSetUserClipAbsolute*( )
Restrict subsequent draws

*PtWidgetVisibleExtent*( )
Calculate the visible portion of a widget

Colors, converting and parsing

*PgAlphaValue*( ) Extract the alpha component from a color value

*PgARGB*( ) Convert alpha, red, green, and blue values to composite color format

*PgBackgroundColorShadings*( )
Calculate top and bottom shading colors

*PgBlueValue*( ) Extract the blue component from a color value

*PgCMY*( ) Convert cyan, magenta, and yellow values to composite color format

*PgColor_t* Composite color value

*PgColorHSV_t* Hue-Saturation-Value color value

*PgColorMatch*( ) Query for best color matches

*PgGetColorModel*( )
Get the current color model
**Configuration files**

*PgGetPalette()* Query for current color palette  

*PgGray()* Generate the RGB value for a shade of gray  

*PgGrayValue()* Extract color brightness  

*PgGreenValue()* Extract the green component from a color value  

*PgHSV()* Convert hue, saturation, and value to composite color format  

*PgHSV2RGB()* Convert HSV colors to RGB  

*PgRedValue()* Extract the red component from a color  

*PgRGB()* Convert red, green, and blue values to composite color format  

*PgRGB2HSV()* Convert RGB colors to HSV  

*PgSetColorModel*() Set the current color model

**Configuration files**

*PxConfigClose*() Close a configuration file  

*PxConfigDeleteEntry*() Delete an entry from a configuration file  

*PxConfigDeleteSection*() Delete a section from a configuration file  

*PxConfigFirstSection*() Seek the beginning of the first section of a configuration file  

*PxConfigForceEmptySection*() Create an empty section in a configuration file
Configuration files

PxConfigNextEntry*()
Get the next entry in the current section

PxConfigNextSection*()
Seek the beginning of the next section of a configuration file

PxConfigNextString*()
Get the next entry in the current section

PxConfigOpen*()
Open a configuration file

PxConfigReadBool*()
Read a Boolean value from a configuration file

PxConfigReadChar*()
Read a character parameter from a configuration file

PxConfigReadDouble*()
Read a double-precision float parameter from a configuration file

PxConfigReadInt*()
Read an integer parameter from a configuration file

PxConfigReadLLong*()
Read a long long integer parameter from a configuration file

PxConfigReadLong*()
Read a long integer parameter from a configuration file

PxConfigReadShort*()
Read a short integer parameter from a configuration file

PxConfigReadString*()
Read a string parameter from a configuration file

PxConfigSection*()
Seek the start of a given section in a configuration file
Connections to other applications

PxConfigWriteBool*( )
Write a Boolean parameter in a configuration file

PxConfigWriteChar*( )
Write a character parameter in a configuration file

PxConfigWriteDouble*( )
Write a double-precision float parameter in a configuration file

PxConfigWriteInt*( )
Write an integer parameter in a configuration file

PxConfigWriteLLong*( )
Write a long long integer parameter in a configuration file

PxConfigWriteLong*( )
Write a long integer parameter in a configuration file

PxConfigWriteShort*( )
Write a short integer parameter in a configuration file

PxConfigWriteString*( )
Write a string parameter in a configuration file

Connections to other applications

PtConnectionAddEventHandlers()
Add a set of server event handlers to a client connection object

PtConnectionAddMsgHandlers()
Add a set of message handlers to a server connection object

PtConnectionClientDestroy()
Destroy a client connection object

PtConnectionClientGetUserData()
Get the client’s user data pointer from a connection object

Chapter 1 • Summary of Entries
Connections to other applications

\textit{PtConnectionClientSetError()}
Set the error-handler function for the client-side of a connection

\textit{PtConnectionClientSetUserData()}
Set the client’s user data pointer in a connection object

\textit{PtConnectionFindId()}
Find the connector with a given ID

\textit{PtConnectionFindName()}
Find the connector with a given name

\textit{PtConnectionFlush()}
Send all pending notifications to the client

\textit{PtConnectionNotify()}
Send a notification event to the client

\textit{PtConnectionReply(), PtConnectionReplymx()}
Reply to a message from a client

\textit{PtConnectionResizeEventBuffer()}
Resize the buffer used to store notifications

\textit{PtConnectionSend(), PtConnectionSendmx()}
Send a message to a server

\textit{PtConnectionServerDestroy()}
Destroy a server connection object

\textit{PtConnectionServerGetUserData()}
Get the server’s user data pointer from a connection object

\textit{PtConnectionServerSetError()}
Set the error-handler function for the server-side of a connection

\textit{PtConnectionServerSetUserData()}
Set the server’s user data pointer in a connection object
Coordinates, translating

- \texttt{PtConnectionTmpName()}
  Create a temporary name for a server

- \texttt{PtConnectionWaitForName()}
  Try to connect to the server with a given name

- \texttt{PtConnectorCreate()}
  Create a connector

- \texttt{PtConnectorDestroy()}
  Destroy a connector

- \texttt{PtConnectorGetId()}
  Get the ID of a connector

Coordinates, translating

- \texttt{PgClearTranslation*()}
  Restore the current translation to the default

- \texttt{PgSetTranslation*()}
  Translate draw commands horizontally and vertically

- \texttt{PhDeTranslateRect()}
  Detranslate a rectangle (subtract offset)

- \texttt{PhTranslateRect()}
  Translate a rectangle (add offset)

Cursors/pointers

- \texttt{PhCursorDef_t}
  Bitmap for the cursor

- \texttt{PhMoveCursorAbs()}
  Move the cursor to an absolute position
**Data chains**

*PhMoveCursorRel()*

Move the cursor to a relative position

*PhQueryCursor()*

Collect cursor information

**Data chains**

*PtAddData()*

Add data to the provided data chain

*PtFindData()*

Find the first data block of a given type and subtype

*PtFindNextData()*

Find the next data block of a given type and subtype

*PtRemoveData()*

Remove a link from a data chain

*PtUnlinkData()*

Remove the provided data link from the data chain

**Direct mode**

*PdCreateDirectContext()*

Create a direct-mode context

*PdDirectStart()*

Enter direct mode

*PdDirectStop()*

Leave direct mode

*PdReleaseDirectContext()*

Leave direct mode and release the direct-mode context

*PgWaitVSync*()

Wait for vertical synchronization
Drag and drop

*PhAllocPackType()*
Allocate a buffer and pack transport data into it

*PhCreateTransportCtrl()*
Allocate a `PhTransportCtrl_t` structure

*PhFindTransportType()*
Find a transport type in the transport registry

*PhFreeTransportType()*
Free data associated with a transport registry entry

*PhGetAllTransportHdrs()*
Extract all the headers from a buffer of packed transport data

*PhGetNextInlineData()*
Get the data for the next entry in a linked list of transport data

*PhGetNextTransportHdr()*
Get the next header from a buffer of packed transport data

*PhGetTransportHdr()*
Extract the header from a buffer of packed transport data

*PhGetTransportVectors()*
Build an I/O vector of data to be transported

*PhLinkTransportData()*
Add transport data to a linked list

*PhLocateTransHdr()*
Look for specific data in a linked list of transport headers
*PhMallocUnpack()*

Unpack transport data, using a custom memory-allocation function

*PhPackEntry()*

Pack transport data, given a transport registry entry

*PhPackType()*

Pack transport data, given the type of data

*PhRegisterTransportType()*

Add a new transport type to the transport registry

*PhReleaseTransportCtrl()*

Free a `PhTransportCtrl_t` structure

*PhReleaseTransportHdrs()*

Free a linked list of headers for packed transport data

*PhTransportCtrl_t*

Control structure for the Photon transport mechanism

*PhTransportFindLink()*

Search a linked list of transport data for some specific data

*PhTransportLink_t*

Entry in a linked list of transport data

*PhTransportRegEntry_t*

Data structure that describes data to be transported

*PhTransportType()*

Pack data into a `PhTransportCtrl_t` structure

*PhUnlinkTransportHdr()*

Remove an entry from a linked list of transport headers
Drag and drop

*PhUnpack()*  Unpack transport data

*PtAddResponseType()*  
Add data to the response chain

*PtCancelDnd()*  Cancel a drag-and-drop operation

*PtCreateTransportCtrl()*  
Create a transport control structure for use with drag and drop

*PtDndFetch_t*  Structure that defines data types a widget accepts from a drag-and-drop event

*PtDndSelect()*  Select drag-and-drop data

*PtGetDndFetchIndex()*  
Select drag-and-drop data

*PtInitDnd()*  Search for an entry in the data_array for an incoming drag and drop event

*PtReleaseTransportCtrl()*  
Release a transport control structure used with drag and drop

*PtTransportCtrl_t*  
Transport-control structure used in a drag-and-drop operation

*PtTransportRequestable()*  
Add an entry for requestable data to the drag-and-drop data

*PtTransportType()*  
Pack transport data
Dragging

PhCancelDrag()  
Cancel a drag operation

PhDragEvent_t  
Data associated with a drag event

PhInitDrag()  
Initiate a drag operation

Draw contexts

PdGetDevices()  
Get region IDs for the currently available draw devices

PdSetTargetDevice()  
Set the target device

PhDCCreate()  
Create and initialize a new draw context

PhDCGetCurrent()  
Get the currently active draw context

PhDCRelease()  
Release a draw context

PhDCSetCurrent()  
Set the currently active draw context

Drawing attributes

General attributes

PgSetDrawMode*()  
Set draw mode

PgSetPalette*()  
Set the color palette

PgSetPlaneMask*()  
Protect video memory from being modified
Drawing attributes

**Fill attributes**

*PgDefaultAlpha()*
- Reset alpha attributes to their default system values

*PgDefaultChroma()*
- Reset chroma attributes to their default system values

*PgDefaultFill()*
- Reset the fill attribute to its default value

*PgSetFillColor*()
- Set exact fill color

*PgSetFillDither*()
- Set specific dither pattern and colors

*PgSetFillTransPat*()
- Set draw transparency

*PgSetFillXORColor*()
- Set a color for XOR drawing

**Line (stroke) attributes**

*PgDefaultStroke()*
- Reset the stroke attribute to its system default

*PgSetStrokeCap*()
- Set what the ends of lines look like

*PgSetStrokeColor*()
- Set the color of subsequent outlines

*PgSetStrokeDash*()
- Set dashed lines

*PgSetStrokeDither*()
- Apply a color pattern to outlines
Drawing attributes

*PgSetStrokeFWidth*()
Set line thickness

*PgSetStrokeJoin*()
Set how lines are joined

*PgSetStrokeTransPat*()
Use a masking pattern to set draw transparency on outlines

*PgSetStrokeXORColor*()
Set the stroke color for XOR drawing

*PgSetStrokeWidth*()
Set line thickness

Text attributes

*PgDefaultText*()
Reset the text attribute to its system default

*PgSetFont*()  Set the text font

*PgSetTextColor*()  Set text color

*PgSetTextDither*()  Set text dither pattern

*PgSetTextTransPat*()  Set draw transparency

*PgSetTextXORColor*()  Set a color for XOR drawing

*PgSetUnderline*()  Set colors for underlining text
Events

PhEmit() Emit an event

PhEmitmx() Emit an event when the event-specific data isn’t contiguous in memory

PhEvent_t Data structure describing an event

PhEventArm() Arm the currently attached Photon channel

PhEventEmit() Emit an event — PhEmit() provides a cleaner API

PhEventEmitmx() Emit an event when the event-specific data isn’t contiguous in memory — PhEmitmx() provides a cleaner API

PhEventNext() Provide synchronous event notification

PhEventPeek() Check to see if an event is pending

PhEventRead() Provide asynchronous event notification

PhEventRegion_t Data structure describing the emitter and collector of an event

PhGetData() Get data for an event

PhGetMsgSize() Get message size

PhGetRects() Get an event’s rectangle set

PhInputGroup() Determine the input group associated with an event

PhKeyEvent_t Data structure describing a key event

PhPointerEvent_t Data associated with a pointer event
Events

PhTimerArm()  Arm a timer event

PhWindowEvent_t  
Data structure describing a window action

PtAddEventHandler()  
Add a single Pt_CB_RAW entry to a widget

PtAddEventHandlers()  
Add several Pt_CB_RAW entries to a widget

PtAddFilterCallback()  
Add a single Pt_CB_FILTER callback to a widget

PtAddFilterCallbacks()  
Add several Pt_CB_FILTER entries to a widget

PtAppAddEventHandler()  
Set an event handler resource for an application.

PtAppAddFilterCallback()  
Set a filter callback resource for an application.

PtAppAddHotkeyHandler()  
Set a hotkey handler resource for an application.

PtAppRemoveEventHandler()  
Remove an event handler from an application’s callback list.

PtAppRemoveFilterCallback()  
Remove a filter callback from an application’s callback list.

PtAppRemoveHotkeyHandler()  
Remove a hotkey handler from an application’s callback list.

Chapter 1 • Summary of Entries  23
Events

\textit{PtBkgdHandlerProcess()}
Process all outstanding Photon events

\textit{PtEventHandler()}
Determine the widgets involved in an event

\textit{PtForwardWindowEvent()}
Forward a window event to the window with a given region ID

\textit{PtMainLoop()}
Implement an application main loop

\textit{PtProcessEvent()}
Standard Photon event-handling function

\textit{PtQuitMainLoop()}
Cause \textit{PtMainLoop()} in the calling thread to return

\textit{PtRemoveEventHandler()}
Remove a single \textit{Pt\_CB\_RAW} entry from a widget

\textit{PtRemoveEventHandlers()}
Remove several \textit{Pt\_CB\_RAW} entries from a widget

\textit{PtRemoveFilterCallback()}
Remove a single \textit{Pt\_CB\_FILTER} entry from a widget

\textit{PtRemoveFilterCallbacks()}
Remove several \textit{Pt\_CB\_FILTER} entries from a widget

\textit{PtResizeEventMsg()}
Resize the event buffer

\textit{PtSendEventToWidget()}
Give an event to a widget

\textit{PtTimerArm()}
Arm a timer event on a widget
Font handling

- **PfAttach()**: Attach to a font server
- **PfConvertFontID()**: Convert a font ID to a font name for backwards compatibility
- **PfDecomposeStemToID()**: Convert a stem name into a font ID
- **PfDetach()**: Detach from a font server
- **PfDynamicLoad()**: Dynamically load a font
- **PfDynamicUnload()**: Unload a dynamically loaded font
- **PfExtent(), PfExtentCx()**: Calculate the extent rectangle of a text string
- **PfExtentComponents()**: Calculate the extent of a text string and invoke a callback
- **PfExtentFractTextCharPositions()**: Calculate individual character positions, using fractional scaling
- **PfExtentText()**: Calculate the extent rectangle of a text string
- **PfExtentTextCharPositions()**: Calculate individual character positions
- **PfExtentTextCharPositionsCx()**: Calculate individual character positions, specifying a font context
Font handling

\textit{PfExtentTextToRect()}  
Calculate the extent of a string, up to a given rectangle

\textit{PfExtentWideText()}  
Calculate the extent rectangle of a wide-character string

\textit{PfFindFont()}  
Generate an ID for a font

\textit{PfFontBaseStem()}  
Get the base stem associated with a given font ID

\textit{PfFontDescription()}  
Get the foundry name of a font

\textit{PfFontFlags()}  
Get the flags for a font

\textit{PfFontSize()}  
Get the point size of a font

\textit{PfFractionalExtentText()}  
Calculate the extent rectangle of a text string, using fractional scaling

\textit{PfFreeFont()}  
Release resources associated with a font

\textit{PfGenerateFontName(), PfGenerateFontNameCx()}  
Generate a font name

\textit{PfGetOutline(), PfGetOutlineCx()}  
Get individual point information for a glyph outline

\textit{PfGlyph()}  
Obtain the metrics and/or bitmap for the specified character

\textit{PfLoadFont()}  
Preload a font within the font server

\textit{PfLoadMetrics()}  
Load metric information for the given font
Geometry

*PfQueryFontInfo()*
Get information about a font

*PfQueryFonts()*
Construct a list of fonts

*PfRender(), PfRender()*
Render a string via a user callback function

*PfTextWidthBytes()*
Calculate the width of a *char* string of multibyte UTF-8 characters

*PfTextWidthChars()*
Calculate the width of a *char* string of multibyte UTF-8 characters

*PfUnloadMetrics()*
Unload metric information for the given font

*PfWideTextWidthBytes()*
Calculate the width of a *wchar_t* string of Unicode characters

*PfWideTextWidthChars()*
Calculate the width of a *wchar_t* string of Unicode characters

*PtFontSelection()*
Create a font-selection dialog

**Geometry**

*PhArea_t*
Position and dimensions of a rectangular area

*PhAreaToRect()*
Convert an area into a rectangle

*PhDim_t*
Dimensions of an area
Gradients

*PhPoint_t* Coordinates of a single point

*PhRect_t* Coordinates of a rectangle

*PhRectIntersect()*

Find the intersection of two rectangles

*PhRectToArea()*

Convert a rectangle into an area

*PhRectUnion()*

Determine a bounding box for two rectangles

**Gradients**

**Driver-level**

*PgDrawGradient*()  
Ask the graphics driver to render a gradient

**Application-level**

*PgBevelBox*()  
Draw a beveled box with gradients

*PgCalcColorContrast*()  
Compute light and dark colors to use for a gradient

*PgContrastBevelBox*()  
Draw a beveled box with gradients and a given level of contrast

*PgDrawGradientBevelBox*()  
Draw a beveled box with gradients and two flat colors
Graphical contexts

*PgClearDrawBuffer*()  
Reset the current draw buffer

*PgCreateGC*()  
Allocate a graphics context

*PgDefaultGC*()  
Reset all graphics context attributes to their default system values

*PgDefaultMode*()  
Reset draw mode and plane mask to their default values

*PgDestroyGC*()  
Release the resources of a graphics context

*PgFlush*(), *PgFFlush*()  
Explicitly flush the current draw buffer

*PgGetGC*()  
Get current graphics context

*PgGetRegion*()  
Get the ID of the region that emits draw events

*PgSetDrawBufferSize*()  
Resize a draw buffer

*PgSetGC*()  
Set current graphics context

*PgSetRegion*()  
Determine which region will emit draw events

Input/Output events

*PtAppAddFd*()  
Add a file-descriptor function

*PtAppAddFdPri*()  
Add a file-descriptor function, specifying a priority

*PtAppRemoveFd*()  
Remove a file-descriptor function
Interprocess Communication (IPC)

PtAppSetFdMode()
Change the mode that’s of interest to an FD handler

PtPulseArm()
Arm a Photon pulse for delivery

PtFileSelection()
Create a file-selection dialog

PtFdProcF_t, PtFdProc_t
Type for defining a file-descriptor function

Interprocess Communication (IPC)

PhChannelAttach()
Create or attach a Neutrino channel

PtAppAddInput()
Add an input processing function

PtAppAddInputRemote()
Add an input processing function for a process on a remote node

PtAppCreatePulse()
Create a Photon pulse

PtAppDeletePulse()
Delete a Photon pulse

PtAppPulseTrigger()
Deliver a Photon pulse to yourself

PtAppRemoveInput()
Remove an input processing entry

PtChannelCreate()
Make sure the widget library is using a channel
Layers

*PtGetRcvidPid()*

Get the process ID (PID) from the receive ID (RCVID)

*PtGetRcvidPidNd()*

Get the process ID (PID) and node descriptor (ND) from the receive ID (RCVID)

*PtInputCallbackProcF_t, PtInputCallbackProc_t*

Type for defining a input callback function

*PtPulseArm()*

Arm a Photon pulse for delivery

---

Layers

*PgCreateDriverRegion()*

Create a region that’s owned by the graphics driver

*PgCreateLayerSurface()*

Create an offscreen context for a layer

*PgGetLayerCaps()*

Query the capabilities of a layer

*PgLayerCaps_t*

Capabilities for a layer

*PgLockLayer()*

Lock a layer for exclusive use by an application

*PgSetLayerArg()*

Configure a layer argument

*PgSetLayerSurface()*

Display the offscreen context on the specified layer surface

*PgUnlockLayer()*

Unlock a layer
Key events, translating

**Key events, translating**

*PhKeyToMb()*  Get the UTF-8 value of a key event

*PhTo8859_1()*  Get the ISO859-1 value of a key event

**Memory contexts**

*PmMemCreateMC()*  
Create a memory context

*PmMemFlush()*  
Flush a memory context

*PmMemReleaseMC()*  
Release a memory context

*PmMemSetChunkSize()*  
Set the increment for growing a memory context’s draw buffer

*PmMemSetMaxBufSize()*  
Set the maximum size of a memory context’s draw buffer

*PmMemSetType()*  
Set the type of a memory context

*PmMemStart()*  
Make a memory context active

*PmMemStop()*  
Deactivate a memory context

**Messages and questions**

*ApError()*  Display an error message dialog

*PtAlert()*  Display a message and request a response

*PtMessageBox()*  
Pop up a message box
Modal dialogs

- **PtNotice()** Display a message and wait for acknowledgment
- **PtPassword()** Prompt for a password
- **PtPrompt()** Display a message and get textual input from the user

**Modal dialogs**

- **ApModalWait()** Process Photon events until a given widget is destroyed
- **PtBlockAllWindows()** Block all windows except the one with a given widget
- **PtBlockWindow()** Block a given window
- **PtMakeModal()** Block all of an application’s windows, except the one containing a given widget
- **PtModalBlock()** Start a modal loop
- **PtModalEnd()** Terminate modal-window processing
- **PtModalStart()** Initiate modal-window processing
- **PtModalUnblock()** Stop a modal loop
- **PtProcessEvent()** Standard Photon event-handling function
- **PtUnblockWindows()** Unblock a set of previously blocked windows
Modules

The following can be used only by applications made with the Photon Application Builder:

*AbGetABW()* Get the instance pointer for a widget

*ApCreateModule()* Create an instance of a module that was built with PhAB

*ApGetInstance()* Get the module link instance pointer for a widget

*ApGetWidgetPtr()* Get the instance pointer for a widget in a given module

*ApModuleFunction()* Specify the setup function for a PhAB internal link callback

*ApModuleLocation()* Specify the module location for a PhAB internal link

*ApModuleParent()* Specify the parent for a window or dialog module

*ApWidget()* Determine the widget that initiated a link callback

Online help

*PtWidgetHelpHit()* Find the first widget at a given position that has a help topic

*PtHelpQuit()* Exit the Helpviewer
Photon Application Builder functions

\(\text{PtHelpSearch()}\) \hspace{1em} \text{Search for text in help information}

\(\text{PtHelpTopic()}\) \hspace{1em} \text{Display the help text identified by the given topic path}

\(\text{PtHelpTopicRoot()}\)
\hspace{1em} \text{Specify the root of help topic paths}

\(\text{PtHelpTopicTree()}\)
\hspace{1em} \text{Load a new help topic tree}

\(\text{PtHelpUrl()}\) \hspace{1em} \text{Display the help text at the given URL}

\(\text{PtHelpUrlRoot()}\) \hspace{1em} \text{Display help text relative to the given URL}

**Photon Application Builder functions**

The following can be used only by applications made with the Photon Application Builder:

\(\text{ApAddContext()}\) \hspace{1em} \text{Add a PhAB context so you can use a PhAB application as a DLL}

\(\text{ApInfo_t}\) \hspace{1em} \text{Data structure for information passed to PhAB callbacks and setup functions}

\(\text{ApInstanceName()}\)
\hspace{1em} \text{Return the widget’s instance name string}

\(\text{ApName()}\) \hspace{1em} \text{Return a PhAB name value for the specified widget}

\(\text{ApRemoveContext()}\)
\hspace{1em} \text{Remove the PhAB context from a PhAB application that you’re using as a DLL}

\(\text{ApResClose()}\) \hspace{1em} \text{Close the file of module resource records}
Photon services, connecting and disconnecting

**Photon services, connecting and disconnecting**

*PhAttach()*  
Open a communications channel

*PhDetach()*  
Free all resources consumed by a Photon channel

*PhLibVersion()*  
Get the version number of the Photon libraries

*PhReattach()*  
Change the current Photon channel

*PtReattach()*  
Send an application to another Photon server

**Power-saving modes**

*PgSetDPMSMode()*  
Set the display power-saving mode

**Primitive drawing routines**

*PgDrawArrow*()  
Draw an arrow that fits inside a given rectangle

*PgBevelBox*()  
Draw a beveled box with gradients

*PgContrastBevelBox*()  
Draw a beveled box with gradients and a given level of contrast

*PgDrawArc*()  
Draw an arc, pie, or chord

*PgDrawBevelBox*(), *PgDrawIBevelBox()*  
Draw a beveled box

*PgDrawBeveled*()  
Draw a beveled rectangle or arrow
Primitive drawing routines

\( \text{PgDrawBezier*}() \)
Draw a stroked and/or filled Bézier curve

\( \text{PgDrawBitmap*}() \)
Draw a bitmap

\( \text{PgDrawEllipse*}() \)
Draw an ellipse

\( \text{PgDrawGradient*}() \)
Ask the graphics driver to render a gradient

\( \text{PgDrawGradientBevelBox*}() \)
Draw a beveled box with gradients and two flat colors

\( \text{PgDrawGrid*}() \)
Draw a grid

\( \text{PgDrawImage*}() \)
Draw an image

\( \text{PgDrawLine*}, \text{PgDrawILine}() \)
Draw a single line

\( \text{PgDrawMultiTextArea*}() \)
Draw multiline text in an area

\( \text{PgDrawPixel*}, \text{PgDrawIPixel*}() \)
Draw a single point

\( \text{PgDrawPixelArray*}() \)
Draw multiple points

\( \text{PgDrawPolygon*}() \)
Draw a stroked and/or filled polygon

\( \text{PgDrawRect*}, \text{PgDrawIRect*}() \)
Draw a rectangle
Printing

\textit{PgDrawRepBitmap} (*)
\begin{itemize}
  \item Draw a bitmap several times
\end{itemize}

\textit{PgDrawRepImage} (*)
\begin{itemize}
  \item Draw an image several times
\end{itemize}

\textit{PgDrawRoundRect} (*)
\begin{itemize}
  \item Draw a rounded rectangle
\end{itemize}

\textit{PgDrawSpan} (*)
\begin{itemize}
  \item Draw a list of spans
\end{itemize}

\textit{PgDrawString} (*)
\begin{itemize}
  \item Draw a string of characters
\end{itemize}

\textit{PgDrawText} (*)
\begin{itemize}
  \item Draw text
\end{itemize}

\textit{PgDrawTextArea} (*)
\begin{itemize}
  \item Draw text within an area
\end{itemize}

\textit{PgDrawTextChars} (*)
\begin{itemize}
  \item Draw the specified number of text characters
\end{itemize}

\textit{PgDrawTrend} (*)
\begin{itemize}
  \item Draw a trend graph
\end{itemize}

Printing

\textit{PpContinueJob} ()
\begin{itemize}
  \item Continue a suspended print job
\end{itemize}

\textit{PpCreatePC} ()
\begin{itemize}
  \item Create a print context
\end{itemize}

\textit{PpEndJob} ()
\begin{itemize}
  \item End a print job
\end{itemize}

\textit{PpFreePrinterList} ()
\begin{itemize}
  \item Free a list of available printers
\end{itemize}
Printing

PpGetCanvas() Get the size of the current drawing area of a print context

PpGetPC() Extract data from a print context

PpLoadDefaultPrinter() Initialize a print context with information for the default printer

PpLoadPrinter() Initialize a print context with information for a given printer

PpLoadPrinterList() Load a list of available printers

PpPrintContext_t Data structure describing a print context

PpPrintNewPage() Place a page break in the draw stream for a print context

PpPrintWidget() Print a widget

PpReleasePC() Release a print context

PpSetCanvas() Set the size of the current drawing area of a print context

PpSetPC() Set data in a print context

PpStartJob() Start a print job

PpSuspendJob() Suspend a print job

PtPrintPropSelect() Change the printing options for a selected printer via a modal dialog

PtPrintSelect() Display a custom modal dialog for selecting print options
Processes

*PtPrintSelection()*

Display a modal dialog for selecting print options

**Processes**

*PtAllowExit()*    
Allow a Photon application to exit

*PtExit()*    
Exit a Photon program

*PtPreventExit()*    
Prevent a Photon application from exiting

*PtSpawn()*    
Spawn a new process

*PtSpawnDeleteCallback()*

Remove a child-termination callback

*PtSpawnSetCallback()*

Change the callback in a PtSpawn() control structure

*PtSpawnWait()*    
Spawn a process and wait for its termination

**Realtime timers**

*RtTimerCreate()*

Create a realtime timer

*RtTimerDelete()*

Delete a realtime timer

*RtTimerGetTime()*

Get the time remaining on a realtime timer

*RtTimerSetTime()*

Set the expiration time for a realtime timer
Regions

*PhQueryRids() Get a list of regions*

*PhRegion_t Data structure that describes a region*

*PhRegionChange() Change the definition of a region*

*PhRegionClose() Remove a region*

*PhRegionDataFindType() Find a data type within a region’s data*

*PhRegionDataHdr_t Data that’s attached to a region*

*PhRegionInfo() Retrieve information about a region with multiple rectangles*

*PhRegionOpen() Open a region*

*PhRegionQuery() Retrieve information about a region*

*PhQuerySystemInfo() Query the system for information about a given region*

*PtWidgetRid() Get a widget’s region ID*

**Shared memory**

*PgShmemAttach() Record a shared memory reference*

*PgShmemCleanup() Remove shared memory references*
Signals

PtAppAddSignalProc()
   Add Photon signalling to a context

PtAppRemoveSignal()
   Remove specific signal handling from a context

PtSignalProcF_t, PtSignalProc_t
   Type for defining a signal-handling function

Strings, translating

AlClearTranslation()
   Clear all the translations in a language or message database

AlCloseDBase()
   Close a language or message database

AlGetEntry()   Get an entry from a language or message database

AlGetSize()   Get the number of records in a language or message database

AlOpenDBase()
   Load a language or message database
Synchronization

ApCloseMessageDB()
Close a message database

AlReadTranslation()
Read a translation file into a database

AlSaveTranslation()
Save translations from a language or message database

AlSetEntry()
Set the translated string for a database entry

ApAppendTranslation()
Append external translation files to an application’s translation list

ApCloseMessageDB()
Close a message database

ApGetTextRes() Get a translated text string from a widget database

ApGetMessage() Get a message from a message database

ApLoadMessageDB() Load a message database

ApSetTranslation() Change the current translation to another language

ApSetContext() Change the application’s current context. Language translations are context-specific.

Synchronization

PgWaitDrawComplete() Wait until all emitted draw streams have been processed
System information

*PgWaitHWIdle()*
Wait until the video driver is idle

*PgWaitVSync*() Wait for vertical synchronization

**System information**

*PhGetConnectId()*
Get the connection ID of the calling process

*PhGetConnectInfo()*
Get information about a Photon channel

*PhQueryCursor()*
Collect cursor information

*PhQuerySystemInfo()*
Query the system for information about a given region

*PhSysInfo_t*
Data structure for system information

*PtQuerySystemInfo()*
Query the system for information about a given widget

**Text**

*PgExtentMultiText()*
Calculate the extent of a multiline text string

*PgExtentText()*
Calculate the extent of a string of text

See also the **PtMultiText** and **PtText** convenience functions.
Threads

*PtCondTimedWait()*
Block a thread on a conditional variable, with a time limit

*PtCondWait()*
Block a thread on a conditional variable

*PtEnter()*
Lock the Photon library for use by a single thread

*PtLeave()*
Unlock the Photon library for use by other threads

Tiles

*PhAddMergeTiles()*
Merge two list tiles, eliminating overlap

*PhClipTilings()*
Clip one list of tiles from another

*PhCoalesceTiles()*
Combine a list of tiles

*PhCopyTiles()*
Copy a list of tiles

*PhDeTranslateTiles()*
Subtract x and y offsets from the vertices of a list of tiles

*PhFreeTiles()*
Return a list of tiles to the internal tile pool

*PhGetTile()*
Retrieve a tile from the internal tile pool

*PhIntersectTilings()*
Determine the intersection of two lists of tiles

*PhMergeTiles()*
Remove all overlap from a list of tiles

*PhRectsToTiles()*
Create a list of tiles from an array of rectangles

*PhSortTiles()*
Sort a list of tiles
UTF-8 character strings

`PhTile_t` A list of rectangles

`PhTilesBoundingRect()` Calculate the bounding box for a list of tiles

`PhTilesToRects()` Create an array of rectangles from a list of tiles

`PhTranslateTiles()` Add x and y offsets to the vertices of a list of tiles

UTF-8 character strings

`utf8len()` Count the bytes in a UTF-8 character

`utf8strlen()` Find the number of UTF8 characters in part of a string

`utf8strchr()` Search for a UTF8 character in a string

`utf8strichr()` Search for a UTF8 character in a string, ignoring case

`utf8strirchr()` Search backwards for a UTF8 character in a string, ignoring case

`utf8strchr()` Search backwards for a UTF8 character in a string

`utf8towc()` Convert a UTF-8 character to a wide-character code
Video modes

\texttt{wctoutf8()} \hspace{1em} Convert a wide-character code into a UTF-8 character

**Video modes**

\texttt{PgGetGraphicsHWCaps()}
  Determine the hardware capabilities

\texttt{PgGetVideoMode()}
  Get the current video mode

\texttt{PgGetVideoModeInfo()}
  Get information about a video mode

\texttt{PgGetVideoModeList()}
  Query a graphics driver for a list of its supported video modes

\texttt{PgSetVideoMode()}
  Set the current video mode

**Video offscreen memory**

\texttt{PdCreateOffscreenContext()}
  Create an offscreen context in video RAM

\texttt{PdCreateOffscreenLock()}
  Create a lock for an offscreen context

\texttt{PdDestroyOffscreenLock()}
  Destroy a lock for an offscreen context

\texttt{PdDupOffscreenContext()}
  Duplicate an offscreen context

\texttt{PdGetOffscreenContextPtr()}
  Create a shared memory object reference to an offscreen context

\texttt{PdGetOffscreenSurface()}
  Get a handle for an offscreen surface
Video overlay

\textit{PdIsOffscreenLocked()},
Determine whether or not an offscreen context is locked

\textit{PdLockOffscreen()},
Lock an offscreen context

\textbf{PdOffscreenContext\_t}
Data structure that describes an offscreen context

\textit{PdSetOffscreenTranslation()}
Set the translation for an offscreen context

\textit{PdUnlockOffscreen()}
Unlock an offscreen context

\textit{PgContextBlit\*()}
Copy data from a rectangle in one context to another context

\textit{PgContextBlitArea\*()}
Copy data from an area in one context to another context

\textit{PgSwapDisplay()}
Point the CRT of the video display at a given context

\textbf{Video overlay}

\textit{PgConfigScalerChannel()}
Configure a video overlay scaler channel

\textit{PgCreateVideoChannel()}
Create a channel for video streaming

\textit{PgDestroyVideoChannel()}
Destroy resources associated with a video channel

\textit{PgGetOverlayChromaColor()}
Return the color used for video overlay chroma-key operations
Wide characters

\textit{PgGetScalerCapabilities()}
Get the capabilities of a video overlay scaler

\textit{PgNextVideoFrame()}
Get the index of the next video buffer to fill

\textit{PgScalerCaps\_t}
Data structure that describes video overlay scaler capabilities

\textit{PgScalerProps\_t}
Data structure that describes video overlay scaler properties

\textit{PgVideoChannel\_t}
Data structure that describes a video overlay channel

Wide characters

\textit{utf8towc()}
Convert a UTF-8 character to a wide-character code

\textit{wctolower()}
Return the lowercase equivalent of a wide character

\textit{wctoutf8()}
Convert a wide-character code into a UTF-8 character

Widgets

Callbacks and hotkey handlers

\textit{PtAddCallback()}
Add a single callback entry to a callback list

\textit{PtAddCallbacks()}
Add several callback entries to a callback list

\textit{PtAddEventHandler()}
Add a single \textit{Pt\_CB\_RAW} entry to a widget

\textit{PtAddEventHandlers()}
Add several \textit{Pt\_CB\_RAW} entries to a widget
Widgets

\textit{PtAddFilterCallback()}

Add a single \textit{Pt\_CB\_FILTER} callback to a widget

\textit{PtAddFilterCallbacks()}

Add several \textit{Pt\_CB\_FILTER} entries to a widget

\textit{PtAddHotkeyHandler()}

Add a single hotkey handler entry to a widget

\textit{PtBalloonCallback\_t}

Balloon callback structure — see the Photon \textit{Widget Reference}

\textit{PtCallback\_t}

Regular callback structure — see the Photon \textit{Widget Reference}

\textit{PtCallbackInfo\_t}

Specific callback information — see the Photon \textit{Widget Reference}

\textit{PtHotkeyCallback\_t}

Hotkey handler structure — see the Photon \textit{Widget Reference}

\textit{PtRawCallback\_t}

Event handler structure — see the Photon \textit{Widget Reference}

\textit{PtRemoveCallback()}

Remove a single callback entry from a callback list

\textit{PtRemoveCallbacks()}

Remove several callback entries from a callback list

\textit{PtRemoveEventHandler()}

Remove a single \textit{Pt\_CB\_RAW} entry from a widget

\textit{PtRemoveEventHandlers()}

Remove several \textit{Pt\_CB\_RAW} entries from a widget

\textit{PtRemoveFilterCallback()}

Remove a single \textit{Pt\_CB\_FILTER} entry from a widget
\textit{PtRemoveFilterCallbacks()} \\
Remove several \textit{Pt\_CB\_FILTER} entries from a widget

\textit{PtRemoveHotkeyHandler()} \\
Remove a single hotkey handler entry from a widget

\textbf{Class hierarchy}

\textit{PtWidgetClass()} \\
Return the class of a widget

\textit{PtWidgetIsClass()} \\
Determine whether a widget is a specific class type

\textit{PtWidgetIsClassMember()} \\
Determine whether a widget belongs to a specified class

\textbf{Control surfaces}

\textit{PtCalcSurface()} \hspace{1em} Force a surface to calculate its geometry

\textit{PtCalcSurfaceByAction()} \\
Force all surfaces associated with an action to calculate their geometry

\textit{PtCalcSurfaceById()} \\
Force the control surface with a given ID to calculate its geometry

\textit{PtCheckSurfaces()} \\
Match an event with the control surfaces belonging to a widget

\textit{PtCreateActionSurface()} \\
Create a control surface within a widget, bound to a widget action
Widgets

*PtCreateSurface()*

Create a regular control surface within a widget

*PtDamageSurface(), PtDamageSurfaceById()*

Mark a control surface as damaged so that it will be redrawn

*PtDamageSurfaceByAction()*

Damage all surfaces that are associated with an action

*PtDestroyAllSurfaces()*

Destroy all of a widget’s control surfaces

*PtDestroySurface()*

Destroy a control surface

*PtDestroySurfaceById()*

Destroy the control surface with a given ID

*PtDisableSurface(), PtDisableSurfaceById()*

Disable a control surface

*PtDisableSurfaceByAction()*

Disable all control surfaces associated with an action

*PtEnableSurface(), PtEnableSurfaceById()*

Enable a control surface

*PtEnableSurfaceByAction()*

Enable all control surfaces associated with an action

*PtFindSurface()*

Find the control surface with a given ID

*PtFindSurfaceByAction()*

Find the control surface associated with a given action
Widgets

**PtHideSurface(), PtHideSurfaceById()**
Hide a control surface

**PtHideSurfaceByAction()**
Hide all control surfaces associated with an action

**PtInsertSurface(), PtInsertSurfaceById()**
Insert a control surface in front of or behind another

**PtShowSurface(), PtShowSurfaceById()**
Show a hidden control surface

**PtShowSurfaceByAction()**
Show all hidden control surfaces associated with an action

**PtSurfaceActionId()**
Get the action ID for a surface

**PtSurfaceAddData(), PtSurfaceAddDataById()**
Add data to a control surface

**PtSurfaceBrotherBehind()**
Get the control surface behind a given one

**PtSurfaceBrotherInFront()**
Get the control surface in front of a given one

**PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById()**
Calculate the bounding box for a control surface

**PtSurfaceExtent(), PtSurfaceExtentById()**
Calculate the extent of a control surface

**PtSurfaceGetData(), PtSurfaceGetDataById()**
Get data associated with a control surface

**PtSurfaceHit()**
Find the control surface hit by a given point

**PtSurfaceId()**
Get the ID of a control surface
Widgets

\textbf{PtSurfaceInBack()}
Get the backmost control surface belonging to a widget

\textbf{PtSurfaceInFront()}
Get the frontmost control surface belonging to a widget

\textbf{PtSurfaceIsDisabled()}
Determine if a control surface is disabled

\textbf{PtSurfaceIsEnabled()}
Determine if a control surface is enabled

\textbf{PtSurfaceIsHidden()}
Determine if a control surface is hidden

\textbf{PtSurfaceIsShown()}
Determine if a control surface is shown

\textbf{PtSurfaceRect(), PtSurfaceRectById()}
Get the bounding box of a control surface

\textbf{PtSurfaceRemoveData(), PtSurfaceRemoveDataById()}
Remove data from a control surface

\textbf{PtSurfaceTestPoint()}
Test whether or not a point is inside a control surface

\textbf{PtSurfaceToBack(), PtSurfaceToBackById()}
Move a control surface behind all other control surfaces belonging to a widget

\textbf{PtSurfaceToFront(), PtSurfaceToFrontById()}
Move a control surface in front of all other control surfaces belonging to a widget

\textbf{PtWidgetActiveSurface()}
Get a widget’s currently active control surface
Creating and destroying widgets

*Pt_ARG()*. Macro for creating statically initialized argument lists

*PtClearWidget()*
Destroy all widgets within a container

*PtCreateWidget()*
Create a widget

*PtDestroyWidget()*
Remove a widget from the widget hierarchy

*PtInflateBalloon()*
Create a balloon widget

*PtSetArg()*. Build argument lists for widgets

Custom widgets

The following are described in *Building Custom Widgets*:

*PhWindowChange()*
Modify the attributes of a window’s region

*PhWindowClose()*
Close a window

*PhWindowOpen()*
Create a window region

*PtAddWidgetData()*
Add data to the widget data chain

*PtAnchorDeregister()*
Deregister a widget from its parent for anchoring
Widgets

*PtAnchorRegister()*

Register a widget with its parent for anchoring

*PtAnchorWidget()*

Anchor the provided widget

*PtApplyAnchors()*

Anchor a widget and its children

*PtAttemptResize()*

Adjust the size of a widget

*PtCalcAnchorOffsets()*

Update the anchoring values (rules) for the given widget

*PtCalcRegion()*

Determine whether or not a widget needs a region

*PtChildBoundingBox()*

Calculate a widget’s canvas and its children’s bounding boxes

*PtClipAdd()*

Add a clipping rectangle to the stack

*PtClipRemove()*

Take a clipping rectangle off the stack

*PtCompoundRedirect()*

Redirect widgets to a parent

*PtCoreChangeRegion()*

Determine if a region is required

*PtCreateWidgetClass()*

Create a widget class

*PtDamageExposed()*

Damage the specified widgets
Widgets

PtDestroyCallbackList()
    Free the specified callbacks
PtDestroyHotkeyCallbacks()
    Free the specified hotkey callbacks
PtDestroyRawCallbacks()
    Free the specified raw callbacks
PtFindNextWidgetData()
    Find the next appropriate data block
PtFindResource()
    Find the record associated with a resource
PtFindWidgetData()
    Find the first data block of a given type and subtype
PtGetCallbackList()
    Get a callback list
PtGetStruct()
    Retrieve the specified resource
PtInvokeCallbackList()
    Invoke a callback list
PtInvokeCallbackType()
    Invoke a callback list of a specific type
PtInvokeResizeCallbacks()
    Invoke the resize callbacks of the specified container
PtMoveResizeWidget()
    Synchronize a widget’s extent
PtRemoveWidgetData()
    Remove data from the widget data chain
Widgets

\textit{PtResizeCanvas()}

Set the size of a widget’s canvas

\textit{PtResizePolicy()}

Determine whether a widget has a resize policy

\textit{PtSetExtentFromArea()}

Calculate the extent of a widget

\textit{PtSetStruct()}

Set the specified resource

\textit{PtSetValue()}

Set the value of a resource using \texttt{mod_f}

\textit{PtSuperClassCalcOpaque()}

Call the Calc Opaque Rect method of the specified superclass

\textit{PtSuperClassChildCreated()}

Invoke a Child Created method

\textit{PtSuperClassChildDestroyed()}

Invoke a Child Destroyed method

\textit{PtSuperClassChildGettingFocus()}

Invoke a Child Getting Focus method

\textit{PtSuperClassChildGettingResources()}

Invoke a Child Getting Resources method

\textit{PtSuperClassChildLosingFocus()}

Invoke a Child Losing Focus method

\textit{PtSuperClassChildMovedResized()}

Invoke a Child Moved/Resized method

\textit{PtSuperClassChildRealized()}

Invoke a Child Realized method

\textit{PtSuperClassChildSettingResources()}

Invoke a Child Setting Resources method
Widgets

PtSuperClassChildUnrealized()

Invoke a Child Unrealized method

PtSuperClassConnect(), PtSuperClassConnectFrom()

Invoke the Connection method of the specified widget class

PtSuperClassDraw()

Invoke the Draw method of the specified superclass

PtSuperClassExtent()

Invoke the Extent method of the specified superclass

PtSuperClassGetResources()

Get the specified resource

PtSuperClassGotFocus()

Invoke the Got Focus method of the specified superclass

PtSuperClassInit(), PtSuperClassInitFrom()

Invoke the Initialize method of the specified widget class

PtSuperClassLostFocus()

Invoke the Lost Focus method of the specified superclass

PtSuperClassRawEvent(), PtSuperClassRawEventFrom()

Invoke the raw callback list of the specified widget class

PtSuperClassRealized()

Invoke the Realization method of the specified widget class

PtSuperClassSetResources()

Set resources
Widgets

PtUpdateVisibility()
Tell the widget library about a change in visibility

PtWidgetAbove()
Get the widget that’s above a given widget in a family hierarchy

Damaging widgets

PtDamageExtent()
Mark an area of a widget as damaged so that it will be redrawn

PtDamageWidget()
Mark a widget as damaged so it will be redrawn

Databases

The following can be used only by applications made with the Photon Application Builder:

ApAddClass() Indicates the widgets likely to be encountered in a widget database

ApCloseDBase()
Close a widget database

ApCopyDBWidget() Copy a widget from a PhAB widget database

ApCreateDBWidget() Create a widget by copying it from a PhAB widget database, specifying a parent

ApCreateDBWidgetFamily() Create a widget family by copying it from a PhAB widget database, specifying a parent
ApCreateWidget()
Create a widget by copying it from a PhAB widget database

ApCreateWidgetFamily()
Create a widget family by copying it from a PhAB widget database

ApDeleteDBWidget()
Remove widgets from a widget database

ApGetDBWidgetInfo()
Get information about a widget in a widget database

ApGetImageRes()
Extract the image data from a widget in a widget database

ApGetTextRes()
Get a translated text string from a widget database

ApOpenDBase()
Open a picture module as a widget database

ApOpenDBaseFile()
Open an external module file as a widget database

ApSaveDBaseFile()
Save a widget database as an external file

ApRemoveClass()
Remove a widget class

**Family hierarchy**

PtChildType()
Determine the relationship between two widgets

PtCreateWidget()
Create a widget
Widgets

\textit{PtFindChildClass()} \\
Find the first descendant that matches the specified class

\textit{PtFindChildClassMember()} \\
Find the first descendant that’s a subclass of the specified class

\textit{PtFindContainer()} \\
Return the nearest container parent

\textit{PtFindDisjoint()} \\
Return the nearest disjoint parent widget

\textit{PtFindFocusChild()} \\
Find the closest focusable child widget

\textit{PtFindGuardian()} \\
Find the widget responsible for another widget’s actions

\textit{PtGetParent()} \\
Find the nearest parent widget that matches the specified class

\textit{PtGetParentWidget()} \\
Return the current default widget parent

\textit{PtNextTopLevelWidget()} \\
Get a pointer to the next top-level widget

\textit{PtReparentWidget()} \\
“Reparent” a widget to a new container

\textit{PtSetParentWidget()} \\
Set the current parent widget

\textit{PtValidParent()} \\
Identify a valid parent for a widget

\textit{PtWidgetBrotherBehind()} \\
Get the brother behind a widget
PtWidgetBrotherInFront()  
Get the brother in front of a widget

PtWidgetChildBack()  
Get the child that’s farthest back in a container

PtWidgetChildFront()  
Get the child at the very front of a container

PtWidgetFamily()  
Traverse the widget hierarchy from back to front

PtWidgetInsert()  
Insert a widget in the widget family hierarchy

PtWidgetParent()  
Get a widget’s parent

PtWidgetSkip()  
Skip to a widget in the next hierarchy

PtWidgetToBack()  
Move a widget behind all its brothers

PtWidgetToFront()  
Move a widget in front of all its brothers

PtWidgetTree()  
Walk the widget tree from front to back

PtWidgetTreeTraverse()  
Walk the widget family hierarchy from front to back

Finding widgets in an area

PtContainerBox()  
Find the next widget in an area

PtContainerHit()  
Find the nth widget in an area

PtHit()  
Identify a widget in the specified container
Widgets

Focus

\texttt{PtContainerFindFocus()}
Find the currently focused widget in the same family hierarchy as a widget

\texttt{PtContainerFocusNext()}
Give focus to the next Pt\_GETS\_FOCUS widget within the same container

\texttt{PtContainerFocusPrev()}
Give focus to the previous Pt\_GETS\_FOCUS widget within the same container

\texttt{PtContainerGiveFocus()}
Give focus to a widget

\texttt{PtContainerNullFocus()}
Truncate the focus chain at the specified widget

\texttt{PtFindFocusChild()}
Find the closest focusable child widget

\texttt{PtFindFocusNextFrom()}
Find the next widget that can get focus

\texttt{PtFindFocusPrevFrom()}
Find the previous widget that can get focus

\texttt{PtGiveFocus()}
Give focus to a widget

\texttt{PtGlobalFocusNext()}
Give focus to next widget

\texttt{PtGlobalFocusNextContainer()}
Give focus to another container’s widget

\texttt{PtGlobalFocusNextFrom()}
Give focus to the next widget behind the specified widget
Widgets

PtGlobalFocusPrev()
Give focus to previous widget

PtGlobalFocusPrevContainer()
Give focus to widget in previous container

PtGlobalFocusPrevFrom()
Give focus to widget previous to the specified widget

PtIsFocused()
Determine to what degree a widget is focused

Geometry

PtCalcAbsPosition()
Calculate the position of a widget based on a position and another widget

PtCalcCanvas()
Calculate the drawable canvas for a widget

PtExtentWidget()
Force a widget to calculate its extent

PtExtentWidgetFamily()
Force a widget and its children to calculate their extents

PtGetAbsPosition()
Get the absolute position of a widget

PtSetAreaFromCanvas()
Calculate an area based on the canvas and borders of a widget

PtWidgetArea()
Retrieve a copy of a widget’s area

PtWidgetDim()
Retrieve a copy of a widget’s dimension
Widgets

*PtWidgetExtent()*
Get a widget’s extent

*PtWidgetMinimumSize()*
Determine the minimum permissible size of a widget

*PtWidgetOffset()*
Find the offset of a widget’s origin from its disjoint parent

*PtWidgetPreferredSize()*
Retrieve the preferred size of a widget

*PtWidgetVisibleExtent()*
Calculate the visible portion of a widget

Library initialization

*PtAppInit()* Initialize an application and create the main window

*PtInit()* Initialize the widget library

Menus

The following can be used only with menu modules created in PhAB:

*ApGetItemText()*
Get the text for a menu item

*ApModifyItemAccel()*
Modify the keyboard shortcut for a menu item

*ApModifyItemState()*
Modify the state of menu items

*ApModifyItemText()*
Modify the text for a menu item

This function is to be used with a PtMenu widget:
Widgets

PtPositionMenu()

Set a menu’s position

PtComboBox

The following are described in the Photon Widget Reference:

PtComboBoxListOpen()

Open a combobox list

PtComboBoxListClose()

Close an open combobox list

PtFileSel

The following are described in the Photon Widget Reference:

PtFSAddAfter()

Insert an item after the specified item

PtFSAddFirst()

Add a root item to the widget

PtFSAllItems()

Fill a buffer with pointers to all items

PtFSAllocItem()

Create an item for a file-selector widget

PtFSClearSelection()

Clear the selection

PtFSDamageItem()

Redraw an item

PtFSExpandParents()

If any ancestors of the given item are collapsed, this function tries to expand them.
Widgets

*PtFSFolderCollapse()*
Collapse an expandable item (directory)

*PtFSFolderExpand()*
Expand an expandable item (directory)

*PtFSFreeAllItems()*
Unlink and free all items

*PtFSFreeItems()*
Free an unlinked item

*PtFSGetCurrent()*
Get the current item

*PtFSGetSelIndexes()*
Fill a buffer with indexes

*PtFSGoto()*
Set the current item

*PtFSItemIndex()*
Calculate the index of the specified item

*PtFSRemoveChildren()*
Unlink all the children of a given item

*PtFSRemoveItem()*
Unlink an item

*PtFSRemoveList()*
Unlink the root item

*PtFSRootItem()*
Return the first root item of the file selector

*PtFSSelect()*
Select the specified item

*PtFSSelectedItems()*
Fill a buffer with item pointers
Widgets

PtFSSetSelIndexes()
Set the selection indexes

PtFSShow()    Set the position so that the specified item is visible

PtFSUnselect()    Unselect the specified item

PtFSUnselectNonBrothers()    Unselect all items that aren’t siblings of the specified item

PtGenList
The following are described in the Photon Widget Reference:

PtGenListAddItems()    Add items to a list

PtGenListAllItems()    Get pointers to all the items in a list

PtGenListClearSelection()    Clear the selection

PtGenListCreateTextBalloon()    Create a popup balloon for an item in the list

PtGenListDamageItem()    Redraw an item when its data has been changed

PtGenListDrawBackground()    Draw the background of a list

PtGenListDrawString()    Draw a string

PtGenListFirstItem()    Return a pointer to the first item in a list
Widgets

`PtGenListGetCurrent()`
Return a pointer to the current item in a list

`PtGenListGetSelIndexes()`
Get the indexes of the selected items

`PtGenListGoto()`
Set the current item so that the new current item is visible

`PtGenListHold()`
Prevent visible repair of a list widget

`PtGenListItem_t`
`PtGenList` item structure

`PtGenListSetColumnBalloon()`
Adjust the balloon text to correspond to a given column

`PtGenListItemIndex()`
Find the index of an item

`PtGenListItemRealloc()`
Reallocate memory for an item

`PtGenListLastItem()`
Return a pointer to the last item in a list

`PtGenListLockItem()`
Lock an item so it can be resized

`PtGenListRelease()`
Release a hold on visible repairs of a list widget

`PtGenListRemoveItems()`
Remove items from a list

`PtGenListResize()`
Resize a list widget
Widgets

*PtGenListSelect()*
Select an item in a list

*PtGenListSelectedItems()*
Get pointers to the selected items

*PtGenListSetGflags()*
Modify the gflags field of the widget

*PtGenListSetSelIndexes()*
Set the selection indexes

*PtGenListShow()*
Set the current position so a given item is visible

*PtGenListUnlockItem()*
Unlock an item so it can be updated

*PtGenListUnselect()*
Unselect an item in a list

The following are described in *Building Custom Widgets*:

*PtSuperClassGenListDraw()*
Invoke the Draw List method in a superclass

*PtSuperClassGenListInflate()*
Invoke the List Inflate method in a superclass

*PtSuperClassGenListKey()*
Invoke the List Key method in a superclass

*PtSuperClassGenListMouse()*
Invoke the List Mouse method in a superclass

*PtSuperClassGenListSelect()*
Invoke the List Select method in a superclass
Widgets

PtGenTree

The following are described in the Photon Widget Reference:

PtGenTreeAddAfter()
Add items after a given item

PtGenTreeAddFirst()
Add items in front of any existing items

PtGenTreeAllItems()
Get pointers to all the items in the tree

PtGenTreeClearSelection()
Clear the selection

PtGenTreeCollapse()
Collapse a subtree

PtGenTreeDamageItem()
Redraw an item when its data has changed

PtGenTreeExpand()
Expand a given subtree

PtGenTreeExpandParents()
Expand any collapsed ancestors of a given item

PtGenTreeFreeAllItems()
Free all the items in a tree

PtGenTreeFreeItems()
Free the items in a subtree

PtGenTreeGetCurrent()
Get a pointer to the current item

PtGenTreeGetSelIndexes()
Get the indexes of the selected items
PtGenTreeGoto()
    Set the current item and position so that a given item is visible

PtGenTreeItem_t
    PtGenTree item structure

PtGenTreeItemIndex()
    Calculate the index of a given item

PtGenTreeItemRealloc()
    Reallocate an item

PtGenTreeItemResize()
    Resize an item

PtGenTreeRemoveChildren()
    Unlink all the children of a given item

PtGenTreeRemoveItem()
    Remove a given item and its children from its parents and siblings

PtGenTreeRemoveList()
    Remove a given items and its siblings from their parent

PtGenTreeResize()
    Resize many items

PtGenTreeRootItem()
    Get a pointer to the first root item

PtGenTreeSelect()
    Select a given item

PtGenTreeSelectedItems()
    Get pointers to the selected items

PtGenTreeSetSelIndexes()
    Set the selection indexes
Widgets

*PtGenTreeShow()*

Set the current position so that a given item is visible

*PtGenTreeUnselect()*

Unselect a given item

*PtGenTreeUnselectNonBrothers()*

Unselect all items that aren’t siblings of a given item

The following are described in *Building Custom Widgets*: 

*PtSuperClassGenTreeDrawItem()*

Invoke the Tree Draw Item method of a given superclass

*PtSuperClassGenTreeItemState()*

Invoke the Tree Item State method of a superclass

**PtList**

The following are described in the Photon *Widget Reference*: 

*PtListAddItems()*

Add one or more items to the list at a specified position

*PtListDeleteAllItems()*

Remove all the items from the list

*PtListDeleteItemPos()*

Delete a range of items by position

*PtListDeleteItems()*

Delete items in the list by name

*PtListGotoPos()*

Make the item at the specified position the current item and display it.
Widgets

\( PtListItemExists() \)
Determine whether or not an item exists within the list

\( PtListItemPos() \)
Determine the position of an item within the list

\( PtListRemovePositions() \)
Remove the items at the specified positions

\( PtListReplaceItemPos() \)
Replace items by position number

\( PtListReplaceItems() \)
Replace items by item text

\( PtListSelectPos() \)
Select the item at the specified position

\( PtListShowPos() \)
Display the item at the specified position

\( PtListUnselectPos() \)
Unselect the item at the specified position

\textbf{PtMTrend}

The following are described in the Photon \emph{Widget Reference}:

\( PtMTrendChangeData() \)
Replace some samples in a trend

\( PtTrendAddData() \)
Add some samples to a trend
Widgets

**PtMultiText**

The following are described in the Photon *Widget Reference*:

- **PtMultiLines_t**
  - Structure for setting multiline text and attributes

- **PtMultiTextAttributes_t**
  - Attributes for multiline text

- **PtMultiTextCallback_t, PtMultiTextControl_t**
  - Information passed to PtMultiText callbacks

- **PtMultiTextCreateAttributes()**
  - Initialize a multiten attribute structure

- **PtMultiTextGetAttributes()**
  - Get the attributes of a PtMultiText widget

- **PtMultiTextInfo()**
  - Get character/line information from a PtMultiText widget

- **PtMultiTextInfo_t**
  - Information passed to PtMultiText callbacks

- **PtMultiTextLine_t**
  - Information about a line of text in a PtMultiText

- **PtMultiTextModifyAttributes()**
  - Modify the attributes of a PtMultiText widget

- **PtMultiTextModifyText()**
  - Modify the contents of a PtMultiText widget

- **PtMultiTextQuery_t**
  - Structure for getting information about a line or character

- **PtMultiTextSegment_t**
  - Information about a segment of text in a PtMultiText
PtPanelGroup

The following are described in the Photon Widget Reference:

*PtPGCreatePopup()*
  Create an empty copy of a panel group as a popup window

*PtPGFindIndexByPanel()*
  Get the index for a panel, given a pointer to the panel

*PtPGFindIndexByTitle()*
  Get the index of a panel, given its title

*PtPGFindPanelByIndex()*
  Get a pointer to the panel widget with a given index

*PtPGFindPanelByTitle()*
  Get a pointer to the panel widget with a given title

*PtPGFindTitleByIndex()*
  Get the title of the panel with a given index

PtProgress

The following are described in the Photon Widget Reference:

*PtProgressEntireSegment()*
  Get the entire segment of a progress bar

*PtProgressFirstSegment()*
  Get the first segment of a progress bar

*PtProgressNextSegment()*
  Get the next segment of a progress bar

*PtProgressTextRect()*
  Get the text area of a progress bar
Widgets

PtTerminal

The following are described in the Photon Widget Reference:

PtTerminalCharset_t, PtTerminalCharsets_t

Character sets used by PtTerminal

PtTerminalCopy()

Copy the current selection to the clipboard

PtTerminalCreateCsXlat()

Create a translation table for PtTerminal’s character sets

PtTerminalDefaultCharsets()

Get the default character sets used by PtTerminal

PtTerminalFontInfo()

Examine a font

PtTerminalGetKeys()

Get the terminal line-editing keys

PtTerminalGetSelection()

Get a copy of the current selection

PtTerminalName()

Get the terminal’s termcap/terminfo name

PtTerminalPasteClipboard()

Paste the contents of the clipboard into the terminal

PtTerminalPasteSelection()

Paste the current selection into the terminal

PtTerminalPut(), PtTerminalPutc(), PtTerminalPuts()

Output text to the terminal

PtTerminalSelectWord()

Select a word
Widgets

**PtText**

The following are described in the Photon *Widget Reference*:

- **PtTextCallback_t, PtTextControl_t, PtTextControlInfo_t**
  Information passed to PtText callbacks

- **PtTextGetSelection()**
  Get the selected range from a PtText widget

- **PtTextModifyText()**
  Modify the contents of a PtText widget

- **PtTextSetSelection()**
  Set the selected range for a PtText widget

**PtTree**

The following are described in the Photon *Widget Reference*:

- **PtTreeAddAfter()**
  Insert an item after the specified item

- **PtTreeAddFirst()**
  Add a root item to the widget, or add an item as the first child of a specified item

- **PtTreeAddImages()**
  Add images to the PtTree’s widgets image list

- **PtTreeAllItems()**
  Fill a buffer with pointers to all items

- **PtTreeAllocItem()**
  Allocate a new item

- **PtTreeClearSelection()**
  Clear the selection
Widgets

\begin{itemize}
  \item \texttt{PtTreeCollapse()}
    \begin{itemize}
      \item Collapse an expandable item
    \end{itemize}
  \item \texttt{PtTreeExpand()}
    \begin{itemize}
      \item Expand an expandable item
    \end{itemize}
  \item \texttt{PtTreeFreeAllItems()}
    \begin{itemize}
      \item Unlink and free all items
    \end{itemize}
  \item \texttt{PtTreeFreeItems()}
    \begin{itemize}
      \item Free an unlinked item
    \end{itemize}
  \item \texttt{PtTreeGetCurrent()}
    \begin{itemize}
      \item Get the current item
    \end{itemize}
  \item \texttt{PtTreeGetSelIndexes()}
    \begin{itemize}
      \item Fill a buffer with indexes of selected items
    \end{itemize}
  \item \texttt{PtTreeGoto()}
    \begin{itemize}
      \item Set the current item
    \end{itemize}
  \item \texttt{PtTreeItem_t PtTreeItemIndex()}
    \begin{itemize}
      \item PtTree item structure
    \end{itemize}
  \item \texttt{PtTreeModifyItem()}
    \begin{itemize}
      \item Change item resources
    \end{itemize}
  \item \texttt{PtTreeModifyItemString()}
    \begin{itemize}
      \item Change the string for a PtTree item
    \end{itemize}
  \item \texttt{PtTreeRemoveChildren()}
    \begin{itemize}
      \item Unlink the children of the specified item
    \end{itemize}
  \item \texttt{PtTreeRemoveItem()}
    \begin{itemize}
      \item Unlink an item
    \end{itemize}
  \item \texttt{PtTreeRemoveList()}
    \begin{itemize}
      \item Unlink the given item and any siblings that follow
    \end{itemize}
\end{itemize}
Widgets

**PtTreeRootItem()**

Return the first root item of the tree

**PtTreeSelect()**

Select the specified item

**PtTreeSelectedItems()**

Fill a buffer with pointers to the selected items

**PtTreeSetSelIndexes()**

Set the selection indexes

**PtTreeShow()**

Set the position so that the specified item is visible

**PtTreeUnselect()**

Unselect the specified item

**PtTreeUnselectNonBrothers()**

Unselect all items that aren’t siblings of the specified item

---

**PtTrend**

The following are described in the Photon Widget Reference:

**PtTrendChangeData()**

Replace some samples for all trends

**PtTrendChangeTrendData()**

Replace some samples for one trend

---

**PtTty**

This function is described in the Photon Widget Reference:

**PtTtyShell()**

Return the default user’s shell
Widgets

**PtWindow**

The following are described in the Photon Widget Reference:

*PtWindowFocus()*
  
  Give a window focus

*PtWindowGetState()*
  
  Return the current state of a window

*PtWindowToBack()*
  
  Move a window to the back of the workspace

*PtWindowToFront()*
  
  Bring a window to the front and gives it focus

**Realizing and unrealizing widgets**

*PtDestroyWidget()*
  
  Remove a widget from the widget hierarchy

*PtRealizeWidget()*
  
  Make a widget and its children visible and possibly interactive

*PtReRealizeWidget()*
  
  Force a widget to unrealize and then rerealize itself

*PtUnrealizeWidget()*
  
  Unrealize a widget

*PtWidgetIsRealized()*
  
  Determine whether a widget is realized

**Resources**

*PtAppAddCallback()*
  
  Set a callback resource for an application.

82  Chapter 1 • Summary of Entries
Widgets

*PtAppGetResource()*
Retrieve one resource value for an application.

*PtAppGetResources()*
Retrieve one or more resource values for an application.

*PtAppRemoveCallback()*
Remove a callback resource from an application.

*PtAppSetResource()*
Set one resource for an application.

*PtAppSetResources()*
Set one or more resources for an application.

*PtArg_t*
Argument structure used for getting and setting widget resources

*Pt_ARG()*
Macro for creating statically initialized argument lists

*PtGetControlFlags()*
Get the flags from the _Pt_ control structure

*PtGetResource()*
Retrieve one resource value for a widget

*PtGetResources()*
Retrieve one or more resource values for a widget

*PtSetArg()*
Build argument lists for widgets

*PtSetResource()*
Set one resource for a widget

*PtSetResources()*
Set one or more resources for a widget
Widgets

PtWidgetClassFlags()
   Retrieve a widget’s class structure flags

PtWidgetFlags()
   Retrieve a widget’s flags

Styles

PtAddClassStyle()
   Add a style to a widget class

PtCreateClassStyle()
   Create a class style

PtDupClassStyle()
   Get a copy of a widget class style

PtFindClassStyle()
   Find the style with a given name

PtGetStyleMember()
   Get a member of a style

PtGetWidgetStyle()
   Get the style that a widget is currently using

PtSetClassStyleMethods()
   Set multiple members of a style from an array

PtSetStyleMember()
   Set a member of a style

PtSetStyleMembers()
   Set multiple members of a style from a variable-length argument list

PtSetWidgetStyle()
   Set the current style for a widget
**Updates, forcing and holding off**

*PtContainerHold()*

Prevent repairs to a container widget and its children

*PtContainerRelease()*

Decrement the flux count for a container, potentially damaging the container

*PtEndFlux()*

Decrement the flux count for a container

*PtFlush()*

Immediately repair widget damage

*PtHold()*

Increment the hold count to prevent the visible repair of all widgets

*PtRelease()*

Decrement the hold count, potentially permitting all widgets to be repaired

*PtStartFlux()*

Prevent repairs to a container widget and its children

*PtSyncWidget()*

Synchronize a widget

*PtUpdate()*

Decrement the hold count

**Window Manager**

*PhWindowQueryVisible()*

Query a visible extent

*PtConsoleSwitch()*

Switch to another virtual console

*PtForwardWindowEvent()*

Forward a window event

*PtForwardWindowTaskEvent()*

Forward a window event to the task with a given Photon connection ID
Window Manager

*PtWindowConsoleSwitch()*

Switch to the console a given window’s displayed on

*PtWindowGetFrameSize()*

Determine the size of a window’s frame

See also the list of *PtWindow* convenience functions.
Chapter 2
Ab—PhAB-Generated Code
This chapter describes the functions and data structures that are generated by the Photon Application Builder (PhAB).

These functions and data structures can be used only by applications built with PhAB.
AbGetABW()  
Return the widget’s instance pointer

Synopsis:

PtWidget_t *AbGetABW( int wgt_name );

Arguments:

wgt_name       The ABN name of the widget that you want to find. PhAB automatically generates these name values for you when you generate your code.

Description:

This macro returns the widget’s instance pointer based on its ABN name. It is similar to ApGetWidgetPtr(), but it doesn’t require a link instance (pointer to the widget’s parent module). Instead of searching the widget hierarchy, AbGetABW() looks up the widget’s ABW pointer, which means it’s much more efficient than ApGetWidgetPtr().

All the limitations that apply to the ABW manifests also apply to AbGetABW(). This means that if you create multiple instances of a module, the ABWs either point to the last instance you created, or are invalid pointers if that instance has already been destroyed. If you create multiple instances of your widgets, you should use ApGetWidgetPtr() rather than AbGetABW().

Returns:

A pointer to the widget wgt_name, or NULL if it wasn’t found.

Classification:

Photon

Safety

Interrupt handler       No
Signal handler           No

continued...
AbGetABW()

Safety

| Thread | No |

See also:

ApGetInstance(), ApGetWidgetPtr()

“Handling multiple instances of a window” in the Working with Code chapter of the Photon Programmer’s Guide
Chapter 3

AI—PhAB Translation
This chapter describes the functions and data structures that let you manipulate translation files (for PhAB apps or message databases) without using the translation editor.

You can use these functions to create your own language editor, or to convert a language database to a different file format (for example, so you can send the file to a non-Photon or non-QNX system for translation).
**AlClearTranslation()**

Clear all the translations in a language or message database

**Synopsis:**

```
#include <photon/Al.h>

void AlClearTranslation( AlDataBase_t *db );
```

**Arguments:**

`db` A pointer to a `AlDataBase_t` structure for the database, returned by `AlOpenDBase()`.

**Library:**

`phexlib`

**Description:**

This function clears all the translations in the given language or message database. It frees the translated strings and sets all the `str_translated` fields to NULL.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

International Language Support chapter of the Photon *Programmer’s Guide*
AlCloseDBase()
Close a language or message database

Synopsis:
#include <photon/Al.h>

void AlCloseDBase( AlDataBase_t *db );

Arguments:
db A pointer to a AlDataBase_t structure for the database, returned by AlOpenDBase().

Library:
phexlib

Description:
AlCloseDBase() closes the language or message database specified by db, releasing all the associated resources.

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

International Language Support chapter of the Photon Programmer’s Guide
AlGetEntry()

Get an entry from a language or message database

Synopsis:

```c
#include <photon/Al.h>

const AlTextEntry_t *AlGetEntry(
    AlDataBase_t const *db,
    unsigned n);
```

Arguments:

db A pointer to a AlDatabase_t structure for the database, returned by AlOpenDBase().
n The number of the record to return.

Library:

phexlib

Description:

AlGetEntry() returns the nth record from the language or message database specified by db.

The AlTextEntry_t structure contains at least the following members:

- **unsigned long res_flags**
  - Bits include:
    - Al_ISMESSAGE — this record describes a message rather than a widget’s resource.
    - Al_MULTILINE — multiple lines are allowed in the translation.
    - Al_ACCELERATOR — this record is a menu item accelerator.

- **const char *wgt_name**
  - The widget name or message tag.
AIAssign()

unsigned long res_value
The resource (0 for messages).

const char *res_descr
A description of the resource or message.

unsigned long res_index
For list items, it’s the index (starting from 0); otherwise, 0.

const char *str_original
The original message/resource.

const char *str_translated
The translated message/resource (initially NULL).

Returns:
A pointer to the record retrieved, or NULL if there isn’t a record
   corresponding to n.

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:
AIAssign(), AIAssignDBase(), AIAssignSize(), AIAssignDBase(),
AIAssignTranslation(), AIAssignTranslation(), AIAssignEntry(),

International Language Support chapter of the Photon Programmer’s
Guide
AlGetSize()

Get the number of records in a language or message database

Synopsis:

```c
#include <photon/Al.h>

unsigned AlGetSize( AlDataBase_t const *db );
```

Arguments:

- `db` A pointer to a `AlDataBase_t` structure for the database, returned by `AlOpenDBase()`.

Library:

- `phexlib`

Description:

This function returns the number of records in the database identified by `db`.

Returns:

The number of records.

Classification:

- Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**AlgGetSize()**

**See also:**


International Language Support chapter of the Photon *Programmer's Guide*
AlOpenDBase()
Load a language or message database

Synopsis:

```c
#include <photon/Al.h>

AlDataBase_t *AlOpenDBase( const char *path );
```

Arguments:

`path`  The pathname of the database that you want to open.

Library:

`phexlib`

Description:

`AlOpenDBase()` loads into memory the PhAB language database or message database stored in the file named by `path`.

This function doesn’t search for the file in any special directories or use any environment variables — the `path` is given directly to `open()`.

Returns:

A pointer to a `AlDataBase_t` structure that describes the database, which you’ll need to pass to functions that work with the database, or `NULL` if an error occurred.

Classification:

`Photon`

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
AlOpenDBase()

See also:

AlClearTranslation(), AlCloseDBase(), AlGetEntry(), AlGetSize(),
AlReadTranslation(), AlSaveTranslation(), AlSetEntry(),

International Language Support chapter of the Photon Programmer’s Guide
AlReadTranslation()
Read a translation file into a database

Synopsis:
#include <photon/Al.h>

int AlReadTranslation( AlDataBase_t *db,
                        const char *path );

Arguments:

  db   A pointer to a AlDataBase_t structure for the database,
        returned by AlOpenDBase().

  path The pathname of the translation file to load into the database.

Library:

  phexlib

Description:

This function reads the translation file with the given path into the
str_translated fields of the database, db.

This function doesn’t search for the file in any special directories or
use any environment variables — the path is given directly to open().

Returns:

  0  Success.

  -1  The file couldn’t be opened.

  A positive number

  The file contained entries that had to be ignored because they
didn’t match anything in the database.
**AlReadTranslation()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


International Language Support chapter of the Photon *Programmer’s Guide*
**AlSaveTranslation()**

*Save translations from a language or message database*

**Synopsis:**

```c
#include <photon/Al.h>

int AlSaveTranslation( AlDataBase_t const *db,
                         const char *path );
```

**Arguments:**

- `db` A pointer to a `AlDataBase_t` structure for the database, returned by `AlOpenDBase()`.
- `path` The pathname of the file in which to save the translations.

**Library:**

`phexlib`

**Description:**

`AlSaveTranslation()` saves the current translations in the database specified by `db` into a translation file named by `path`.

This function doesn’t locate the file in any special directories or use any environment variables — the `path` is given directly to `open()`.

**Returns:**

- 0  Success.
- -1  An error occurred.

**Classification:**

Photon
**AlSaveTranslation()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*AlClearTranslation(), AlCloseDBase(), AlGetEntry(), AlGetSize(),
AlOpenDBase(), AlReadTranslation(), AlSetEntry(),

International Language Support chapter of the Photon *Programmer’s Guide*
**AlSetEntry()**

Set the translated string for a database entry

**Synopsis:**
```
#include <photon/Al.h>

int AlSetEntry( AlDataBase_t *db,
                unsigned n,
                const char *string );
```

**Arguments:**
- **db** A pointer to a `AlDataBase_t` structure for the database, returned by `AlOpenDBase()`.
- **n** The number of the entry in the database to set.
- **string** The string to save in the entry.

**Library:**

`phexlib`

**Description:**
This function sets the `str_translated` field in the `n`th entry of the language or message database specified by `db`. For information about the members of the entry, see `AlGetEntry()`.

**Returns:**
- 0 Success.
- -1 An error occurred.

**Classification:**

Photon
AlSetEntry()

**Safety**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

AlClearTranslation(), AlCloseDBase(), AlGetEntry(), AlGetSize(),
AlOpenDBase(), AlReadTranslation(), AlSaveTranslation(),

International Language Support chapter of the Photon Programmer’s Guide
This chapter describes the functions and data structures that are associated with the Photon Application Builder (PhAB).

These functions and data structures can be used only by applications built with PhAB.
ApAddClass()

Indicate the widgets likely to be encountered in a widget database

Synopsis:

```c
#include <Ap.h>

int ApAddClass(
    char const * class_name_string,
    PtWidgetClassRef_t * const * wgt_class);
```

Arguments:

- `class_name_string`
  The name of the class (for example, "PtButton").

- `wgt_class`
  A predefined widget class; specify the name of the class preceded by an ampersand (for example, &PtButton).

Library:

`Ap`

Description:

This function lets you indicate which widget classes you’re likely to encounter when you call `ApOpenDBaseFile()`.

When you link your application, only those widgets it needs are linked into it. If you access widgets that aren’t in your application because they’re in an external database, you must add them to your internal class table so that they can be linked in at compile time.

Any widgets used when you build a PhAB application are automatically included in the internal class table (see the generated `abmain.c` file). When you use widgets from an external database, PhAB doesn’t know which widgets you need to access, so you’ll have to use this function to include them.

If you’re loading a DLL that adds classes, you should call `ApRemoveClass()` to remove the classes before you unload the DLL.
ApAddClass()

Returns:

0  Success.
-1  There isn’t enough memory to add the widget class, or you’ve already added a different widget class with the same name.

Examples:

```c
base_setup ( ... )
{
    ApAddClass ("PtProgress", &PtProgress);
    return (Pt_CONTINUE);
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

ApOpenDBaseFile(), ApRemoveClass()

Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
**ApAddContext()**

*Add a PhAB context so you can use a PhAB application as a DLL*

**Synopsis:**

```c
#include <Ap.h>

int ApAddContext( ApContext_t *context,
                  const char *exe_path );
```

**Arguments:**

- `context` A pointer to the context to add. This argument should be the address of `AbContext`, a global data structure that PhAB puts in `abmain.c`.
- `exe_path` The full path of the DLL, suitable for passing to `open()`.

**Library:**

`Ap`

**Description:**

This function adds a PhAB context so you can use a PhAB application as a DLL.

**Returns:**

0 on success, or -1 if there wasn’t enough memory or `ApAddClass()` failed.


**Classification:**

Photon

**Safety**

Interrupt handler No

continued...
ApAddContext()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApAddClass(), ApRemoveContext()

“Making a DLL out of a PhAB application” in the Generating, Compiling, and Running Code chapter of the Photon Programmer’s Guide
**ApAppendTranslation()**

Append external translation files to an application’s translation list

**Synopsis:**

```c
#include <Ap.h>

int ApAppendTranslation( char const *filename,
                        char const *lang_extension );
```

**Arguments:**

- `filename`  The name of the translation file that you want to append.
- `lang_extension`  NULL, or the language extension to append to the file name.

**Library:**

`Ap`

**Description:**

This function is used to append external translation files to the application’s translation list. It takes the translation file identified by `filename`, appends the `lang_extension` and looks for the translation file in the directory defined by the `ABLPATH` environment variable. If `lang_extension` is NULL, the current language extension is used.

This is useful when you want to share common text strings among many different applications. Essentially, you create a standalone application that contains a single picture module (widget database) of text strings. Then you use the PhAB Language Editor to translate the strings in this module. Once the database is created and translated, you can access it from another application.

Keep in mind when calling this function from a DLL that this function performs the language extension search based on the location of the executable or DLL associated with the current context, and appends the language translation to the current context. You must make sure that there is a current context. You most likely want your DLL to append a language translation to its own context, rather than the main program’s context:
ApAppendTranslation()

ApContext_t *old = ApSetContext( &AbContext );
ApAppendTranslation( filename, NULL );
ApSetContext( old ); // Restore the program’s context

For more information about creating DLLs from PhAB applications, see “Making a DLL out of a PhAB application” in the Generating, Compiling, and Running Code chapter of the Photon Programmer’s Guide.

Returns:

0 Success.
-1 Unable to read the translation file.

Examples:

Assuming ABLPATH has been set to /usr/photon/translations:

ApDBase_t *mytext_db;

/* Open the text database. */
mytext_db = ApOpenDBaseFile( "fullpath/mytext_db.wgtp" );

/* Set the translation to German. */
ApSetTranslation( "de_DE" );

/* Append the external German translation file to the current list. This will read the translation file "fullpath/mytext_db.wgtp" and append it to the application’s current translation list. */
ApAppendTranslation( "mystrings", "de_DE" );

/* Get a translated text string. */
text = ApGetTextRes( mytext_db, "msg001" );

Classification:

Photon
**ApAppendTranslation()**

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:


International Language Support chapter of the Photon *Programmer’s Guide*
ApCloseDBase()

Close a widget database

Synopsis:

```c
#include <Ap.h>

int ApCloseDBase( ApDBase_t *db );
```

Arguments:

- `db` A pointer to a PhAB picture database structure, returned by `ApOpenDBase()` or `ApOpenDBaseFile()`.

Library:

Ap

Description:

`ApCloseDBase()` closes a widget database that has been opened with `ApOpenDBase()` or `ApOpenDBaseFile()`.

Closing a widget database deallocates its memory. If you’re finished using a widget database, close it to free up the memory.

If you use a widget database to create widgets that have `PhImage_t` data attached to them, don’t close the database until those widgets are destroyed. Closing the database frees the memory used by the image. If you must close the database, make sure to copy the image data within your application code and to reset the image data resource to point to your new copy.

Returns:

- 0 Successful completion

Examples:

```c
ApDBase_t *mydbase;
mydbase = ApOpenDBase( ABM_mypicture );
ApCreateWidget( mydbase, "this_widget", 10, 10, 0, NULL );
```
ApCloseDBase()

ApCreateWidget(mydbase, "that_widget", 50, 10, 0, NULL);

ApCloseDBase(mydbase);

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApOpenDBase(), ApOpenDBaseFile(), PhImage_t

Accessing PhAB Modules from Code chapter of the Photon
Programmer’s Guide
ApCloseMessageDB()
Close a message database

Synopsis:

```c
void ApCloseMessageDB( ApMsgDBase_t *db );
```

Arguments:

- `db`: A pointer to a message database, returned by `ApLoadMessageDB()`. 

Library:

`Ap`

Description:

This function closes the message database specified by `db`.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`AlClearTranslation(), AlCloseDBase(), AlGetEntry(), AlGetSize(), AlOpenDBase(), AlReadTranslation(), AlSaveTranslation(), AlSetEntry(), ApGetMessage(), ApLoadMessageDB()`

International Language Support chapter of the Photon Programmer’s Guide
ApCopyDBWidget()

Copy a widget from a PhAB widget database

Synopsis:

```c
#include <Ap.h>

#include <Ap.h>

int ApCopyDBWidget( ApDBase_t const *from_dbase,
                     char const *from_name,
                     ApDBase_t *to_dbase,
                     char const *to_name );
```

Arguments:

- `from_dbase`: A pointer to a PhAB widget database structure, returned by `ApOpenDBase()` or `ApOpenDBaseFile()`, for the database that you want to copy a widget from.
- `from_name`: The name of the widget that you want to copy.
- `to_dbase`: A pointer to a PhAB widget database structure for the database that you want to copy the widget to.
- `to_name`: The name to use for the copy of the widget.

Library:

Ap

Description:

`ApCopyDBWidget()` copies a widget from one PhAB widget database to another. The `from_name` parameter indicates which widget to copy from the database, and `to_name` lets you rename the copy. Only one widget can be copied at a time. If you copy a container-class widget, only the container widget is copied, not its children.

This function was previously called `ApCopyWidget()`. You should use the new name, although applications that use the old name will still work.
ApCopyDBWidget()

Returns:

0  Success.
-1  Failure.

Examples:

ApDBase_t *from_dbase, *to_dbase;

from_dbase = ApOpenDBaseFile( "~/home/me/mydbase.wgtp" );
to_dbase  = ApOpenDBaseFile( "~/home/joe/his_dbase.wgtp" );
ApCopyDBWidget( from_dbase, "my_icon", to_dbase, "his_icon" );
ApSaveDBaseFile( to_dbase, "~/home/joe/his_dbase.wgtp" );

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:


Accessing PhAB Modules from Code chapter of the Photon
Programmer’s Guide
ApCreateDBWidget()
Create a widget by copying it from a PhAB widget database, specifying a parent

Synopsis:
#include <Ap.h>

PtWidget_t *ApCreateDBWidget(
    ApDBase_t const *db,
    char const *wgt_name,
    PtWidget_t *parent,
    PhPoint_t const *pos,
    int nargs,
    PtArg_t const *args);

Arguments:

    db          A pointer to a widget database that you opened with either ApOpenDBase() or ApOpenDBaseFile().
    wgt_name    The instance name of one of the widgets inside the database.
    parent      The parent for the widget. If parent is NULL, the widget has no parent. (Contrast this with ApCreateWidget(), which makes the widget a child of the current default parent.)
    pos         A pointer to a PhPoint_t structure that specifies the position of the widget when it’s created. If pos is NULL, the widget’s original position is used.
    nargs and args
                The standard resource argument counter and argument list (of type PtArg_t) used by PtCreateWidget().

Library:

    Ap
ApCreateDBWidget()

Description:

This function creates a widget by copying a widget from a PhAB widget database. This is very useful when you need to create many instances of the same widget. For example, a file manager may want to draw a file folder for each directory that it displays.

Before loading widgets from an external database, you should call ApAddClass() for each widget class that you’ll likely encounter in it. This will add the widget classes to the internal widget class table.

ApCreateDBWidget() creates only the widget named by wgt_name regardless of its class. ApCreateDBWidgetFamily() creates the named widget and, for container class widgets, any children of the widget.

If the widget returned by this function contains images, the images reference data in the widget database. Therefore, don’t close the widget database while you’re using the widget. If you need to close the database, you must remove all references to image data. You can do this by destroying the widgets, unsetting images in the widgets, or changing them into images that don’t reference the database by using PiDuplicateImage() to copy the images from the database.

Returns:

A pointer to the widget created for wgt_name, or NULL on failure.

Examples:

```c
ApDBase_t *mydbase;
PtArg_t args[2];
PhPoint_t pos;
PtWidget_t *my_label;

mydbase = ApOpenDBase( ABM_mypicture );

PtSetArg( &args[0], Pt_ARG_TEXT_STRING,
   "This Widget", 0 );
pos.x = 10;
pos.y = 10;
ApCreateDBWidget( mydbase, "my_label_widget", 127 );
```
ApCreateDBWidget()

    my_window, &pos, 1, args );
PtSetArg( &args[0], Pt_ARG_TEXT_STRING,
    "That Widget", 0 );
PtSetArg( &args[1], Pt_ARG_FILL_COLOR, Pg_WHITE, 0 );
pos.y = 30;
my_label = ApCreateDBWidget( mydbase, "my_label_widget",
    my_window, &pos, 1, args );
if (my_label != NULL)
{
    PtRealizeWidget( my_label );
}
ApCloseDBase( mydbase );

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApOpenDBaseFile(), ApSaveDBaseFile(), PhPoint_t, PtArg_t,
PtCreateWidget()  

Accessing PhAB Modules from Code chapter of the Photon
Programmer’s Guide
ApCreateDBWidgetFamily()

Create a widget family by copying it from a PhAB widget database, specifying a parent

Synopsis:

```c
#include <Ap.h>

PtWidget_t *ApCreateDBWidgetFamily(
    ApDBase_t const *db,
    char const *wgt_name,
    PtWidget_t *parent,
    PhPoint_t const *pos,
    int nargs,
    PtArg_t const *args);
```

Arguments:

- `db`: A pointer to a widget database that you opened with either `ApOpenDBase()` or `ApOpenDBaseFile()`.
- `wgt_name`: The instance name of one of the widgets inside the database.
- `parent`: The parent for the widget. If `parent` is NULL, the widget has no parent. (Contrast this with `ApCreateWidgetFamily()`, which makes the widget a child of the current default parent.)
- `pos`: A pointer to a `PhPoint_t` structure that specifies the position of the widget when it’s created. If `pos` is NULL, the widget’s original position is used.
- `nargs` and `args`: The standard resource argument counter and argument list (of type `PtArg_t`) used by `PtCreateWidget()`.

Library:

`Ap`
**ApCreateDBWidgetFamily()**

**Description:**
This function creates widgets by copying a widget family from a PhAB widget database. This is very useful when you need to create many instances of the same widget family.

Before loading widgets from an external database, you should call *ApAddClass()* for each widget class that you’ll likely encounter in it. This will add the widget classes to the internal widget class table.

*ApCreateDBWidgetFamily()* creates the named widget and, for container class widgets, any children of the widget.

The pointers of the widget’s children aren’t directly available using this function. If you need access to the container’s children, you’ll need to call *ApCreateDBWidget()* for the container and each widget inside it. If you create them in the same hierarchical order as defined in the database, the parent-child relationship will be maintained.

If any of the widgets returned by this function contain images, the images reference data in the widget database. Therefore, don’t close the widget database while you’re using these widgets. If you need to close the database, you must remove all references to image data. You can do this by destroying the widgets, unsetting images in the widgets, or changing them into images that don’t reference the database by using *PiDuplicateImage()* to copy the images from the database.

**Returns:**
A pointer to the widget created for *wgt_name*, or NULL on failure.

**Classification:**
Photon
ApCreateDBWidgetFamily()

Safety

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
**ApCreateModule()**

Create an instance of a module that was built with PhAB

**Synopsis:**

```c
#include <Ap.h>

PtWidget_t * ApCreateModule(
    ApEventLink_t const *link_callback,
    PtWidget_t *widget,
    PtCallbackInfo_t *cbinfo);
```

**Arguments:**

- `link_callback` The manifest that PhAB created for the module.
- `widget` A pointer to a widget. The way that the function uses this argument depends on whether or not the module is a picture; for more information, see “Usage with window, dialog, menu, and other modules” and “Usage with picture modules,” below.
- `cbinfo` NULL, or a pointer to the `PtCallbackInfo_t` structure (see the Photon Widget Reference) that was passed to the callback. It’s used only if you’re creating a module that’s positioned relative to the pointer.

**Library:**

`Ap`

**Description:**

You can use `ApCreateModule()` to manually create instances of modules built with PhAB. Its behavior depends on the type of module that you’re creating, as described below.

**Usage with window, dialog, menu, and other modules**

Before you can create an instance of a module, you must create an internal link to the module using the Internal Links dialog accessible from PhAB’s Application menu. For every internal link in the list,
PhAB generates a manifest that you can use with *ApCreateModule()* to create the module.

This function and PhAB’s link callbacks behave in very similar ways. If you define a location and a setup function for the internal link, the module appears at the specified location, and the setup function is called. All link callbacks that are attached to widgets inside the module are handled properly.

This function can be very handy in situations where you can’t use a regular link callback. For example, a menu item may need to display one dialog or another, depending on certain conditions in your application code. In this case, you can attach a code link callback to the menu item and in the code function you can create the appropriate module.

A module created with *ApCreateModule()* becomes a standard Photon widget (e.g. *PtWindow, PtMenu*) that you can destroy later using *PtDestroyWidget()* if you want to close the module.

*ApCreateModule()* is also useful when you need to display modules without direct user interaction.

This function uses the *widget* argument only if you’re creating a module with a location relative to another widget. Otherwise, you can set it to NULL. It’s passed to the module’s setup function as *apinfo->widget*.

The *cbinfo* argument is a pointer to the *PtCallbackInfo_t* structure (see the Photon Widget Reference) that was passed to the callback. It’s used only if you’re creating a module relative to the pointer position. Otherwise, you can set it to NULL.

This code creates one of two dialogs:

```c
int mycallback( PtWidget_t *widget,
                ApInfo_t *apinfo,
                PtCallbackInfo_t *cbinfo )
{
    /* check conditions */
    if ( condition1 ) {
        ApCreateModule( ABM_mydialog1, widget, cbinfo );
```
ApCreateModule()

} else {
    ApCreateModule( ABM_mydialog2, widget, cbinfo );
}

return (Pt_CONTINUE);
}

To specify the parent for a window or dialog module, use
ApModuleParent().

Usage with picture modules

ApCreateModule() is the only way to create picture modules because
pictures don’t have a direct link callback. You can’t attach a link
callback from a widget to a picture module. Instead, design your
picture module as you would a window or dialog module, and then
add the picture to the internal callbacks list. PhAB generates the
necessary manifests for you to access the picture from within your
application code.

For a more detailed description of picture modules and how to use
them, see “Picture modules” in the Working with Modules chapter of
the Programmer’s Guide.

ApCreateModule() uses the widget argument in a different way when
you’re creating pictures instead of other modules. Since pictures don’t
have an associated location, you use the widget argument to specify
the picture module’s container widget:

• If widget is non-NULL, the children of the picture module become
children of widget, and the picture module itself isn’t created. For
example:

    int mycallback( PtWidget_t *widget,
                    ApInfo_t *apinfo,
                    PtCallbackInfo_t *cbinfo )
    {
        /* Clear the container widget. */
        PtClearWidget( ABW_mycontainer );

        /* Create the picture’s children inside mycontainer. */
        ApCreateModule( ABM_mypicture, ABW_mycontainer,
                        cbinfo );

        /* Force the container to be updated. */
    }
ApCreateModule()

PtReRealizeWidget( ABW_mycontainer );

return (Pt_CONTINUE);
}

- If `widget` is NULL, the picture module becomes a child of the current parent widget, and the picture module’s children become grandchildren of the current parent widget. You can call `PtSetParentWidget()` to change the current parent. For example:

```c
int mycallback( PtWidget_t *widget,
    ApInfo_t *apinfo,
    PtCallbackInfo_t *cbinfo )
{
    /* Clear the container widget. */
    PtClearWidget( ABW_mycontainer );

    /* Create the picture inside mycontainer. */
    PtSetParentWidget( ABW_mycontainer );
    ApCreateModule( ABM_mypicture, NULL, cbinfo );

    /* Display the picture. */
    PtRealizeWidget( ABM_mypicture );

    return (Pt_CONTINUE);
}
```

Returns:

A pointer to the instance of the created module, or NULL if an error occurred or a setup function aborted the creation. There are some special cases:

- If the module is a dialog that’s already instantiated, a pointer to the existing instance is returned.

- If the module is a picture and the `widget` argument to `ApCreateModule()` isn’t NULL, a pointer to that widget (which now contains the contents of the picture module) is returned. If `widget` is NULL, a pointer to module created is returned.
ApCreateModule()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:


PtCallbackInfo_t in the Photon Widget Reference

**ApCreateWidget()**

Create a widget by copying it from a PhAB widget database

### Synopsis:

```c
#include <Ap.h>

PtWidget_t *ApCreateWidget( ApDBase_t const *db,
                           char const *wgt_name,
                           int x,
                           int y,
                           int nargs,
                           PtArg_t const *args);
```

### Arguments:

- **db**: A pointer to a widget database that you opened with either `ApOpenDBase()` or `ApOpenDBaseFile()`.
- **wgt_name**: The instance name of one of the widgets inside the database.
- **x and y**: Convenience arguments for specifying the position of the widget when it’s created. If y is -1, the widget’s original position is used.
- **nargs and args**: The standard resource argument counter and argument list (of type `PtArg_t`) used by `PtCreateWidget()`.

### Library:

Ap

### Description:

This function is used to create widgets by copying a widget from a PhAB widget database. This is very useful when you need to create many instances of the same widget. For example, a file manager may want to draw a file folder for each directory it displays.
**ApCreateWidget()**

You can use `ApCreateDBWidget()` instead of this function. `ApCreateDBWidget()` lets you specify the position without having to worry about the case when \( y \) happens to be -1.

The widget is created as a child of the default parent, which is usually the most recently created container. To change the default parent, call `PtSetParentWidget()`.

Before loading widgets from an external database, you should call `ApAddClass()` for each widget class that you’ll likely encounter in it. This will add the widget classes to the internal widget class table.

`ApCreateWidget()` creates only the widget named by `wgt_name` regardless of its class. `ApCreateWidgetFamily()` creates the named widget and, for container class widgets, any children of the widget.

If the widget returned by this function contains images, the images reference data in the widget database. Therefore, don’t close the widget database while you’re using the widget. If you need to close the database, you must remove all references to image data. You can do this by destroying the widgets, unsetting images in the widgets, or changing them into images that don’t reference the database by using `PiDuplicateImage()` to copy the images from the database.

**Returns:**

A pointer to the widget created for `wgt_name`, or NULL on failure.

**Examples:**

```c
ApDBase_t *mydbase;
PtArg_t args[2];
PtWidget_t *my_label;

mydbase = ApOpenDBase( ABM_mypicture );

PtSetArg( &args[0], Pt_ARG_TEXT_STRING, "This Widget", 0 );
ApCreateWidget( mydbase, "my_label_widget", 10, 10, 1, args );
```
ApCreateWidget()

PtSetArg( &args[0], Pt_ARG_TEXT_STRING, "That Widget", 0 );
PtSetArg( &args[1], Pt_ARG_FILL_COLOR, Pg_WHITE, 0 );
my_label = ApCreateWidget( mydbase, "my_label_widget", 10, 30, 2, args );
if (my_label != NULL)
{
    PtRealizeWidget( my_label );
}

ApCloseDBase( mydbase );

Classification:
Photon

Safety

Interrupt handler  No
Signal handler  No
Thread  No

See also:

Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
ApCreateWidgetFamily()
Create a widget family by copying it from a PhAB widget database

Synopsis:

```c
#include <Ap.h>

PtWidget_t *ApCreateWidgetFamily(
    ApDBase_t const *db,
    char const *wgt_name,
    int x,
    int y,
    int nargs,
    PtArg_t const *args);
```

Arguments:

- `db` A pointer to a widget database that you opened with either `ApOpenDBase()` or `ApOpenDBaseFile()`.
- `wgt_name` The instance name of one of the widgets inside the database.
- `x` and `y` Convenience arguments for specifying the position of the widget when it’s created.
  
  If `y` is -1, the widget’s original position is used.

- `nargs` and `args` The standard resource argument counter and argument list (of type `PtArg_t`) used by `PtCreateWidget()`.

Library:

- **Ap**

Description:

This function is used to create widgets by copying a widget family from a PhAB widget database. This is very useful when you need to create many instances of the same widget. For example, a file manager may want to draw a file folder for each directory it displays.
ApCreateWidgetFamily()

You can use ApCreateDBWidgetFamily() instead of this function. ApCreateDBWidgetFamily() lets you specify the position without having to worry about the case when y happens to be -1.

The root of the widget family is created as a child of the default parent, which is usually the most recently created container. To change the default parent, call PtSetParentWidget().

Before loading widgets from an external database, you should call ApAddClass() for each widget class that you’ll likely encounter in it. This will add the widget classes to the internal widget class table.

ApCreateWidgetFamily() creates the named widget and, for container class widgets, any children of the widget.

The pointers of the widget’s children aren’t directly available using this function. If you need access to the container’s children, you’ll need to call ApCreateWidget() for the container and each widget inside it. If you create them in the same hierarchical order as defined in the database, the parent-child relationship will be maintained.

If any of the widgets returned by this function contain images, the images reference data in the widget database. Therefore, don’t close the widget database while you’re using these widgets. If you need to close the database, you must remove all references to image data. You can do this by destroying the widgets, unsetting images in the widgets, or changing them into images that don’t reference the database by using PiDuplicateImage() to copy the images from the database.

Returns:

A pointer to the widget created for wgt_name, or NULL on failure.
ApCreateWidgetFamily()

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
ApDeleteDBWidget()
Remove widgets from a widget database

Synopsis:
#include <Ap.h>

int ApDeleteDBWidget( ApDBase_t *db,
                      char const *wgt_name );

Arguments:

  db          A pointer to a widget database that you opened with
              either ApOpenDBase() or ApOpenDBaseFile().
  wgt_name    The instance name of the widget that you want to
              delete.

Library:
Ap

Description:
ApDeleteDBWidget() removes the widget named wgt_name from the
widget database indicated by db. If the widget is a container, its
children are also removed.

Returns:

0      Success.
-1     Failure.

Examples:

ApDBase_t *my_dbase;

my_dbase = ApOpenDBaseFile( "~/home/me/mydbase.wgtp" );

ApDeleteDBWidget( my_dbase, "my_icon" );

ApSaveDBaseFile( my_dbase, "~/home/me/mydbase.wgtp" );

Chapter 4 • Ap—PhAB  143
**ApDeleteDBWidget()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


Accessing PhAB Modules from Code chapter of the Photon *Programmer’s Guide*
ApError()
Display an error message dialog

Synopsis:

```c
#include <Ap.h>

void ApError( PtWidget_t *widget,
    int errnum,
    char const *app_prefix,
    char const *error_message,
    char const *location );
```

Arguments:

- `widget` A pointer to the parent widget for the error message dialog.
- `errnum` The standard `errno` value set by the operation that failed. The `errno` variable and its values are defined in `<errno.h>`, and are described in the QNX Neutrino Library Reference.
- `app_prefix` A string of prefix letters for identifying the application. For example, `PhAB` is the prefix for the Photon Application Builder.
- `error_message` The error text to be displayed.
- `location` The location in the code where the error occurred. If you don’t want the location displayed, specify `NULL`.

Library:

`Ap`

Description:

`ApError()` displays an error message dialog on the screen. This is a modeless dialog that doesn’t wait for a user response. It displays a formatted error message and a single OK button for acknowledgment.
ApError()

A sample dialog displayed by AppError().

ApError() builds the message, in order, from the following:

- *app_prefix*
- *error_message*
- the string associated with the value of *errnum*
- *location*, if specified.

Examples:

If you call ApError() as follows, specifying the location:

```c
ApError( ABW_base, errno, "PhAB", "Unable to save file", __FILE__ );
```

then the error dialog is formatted as:

**PhAB: Unable to save file (Permission denied)**

In the example above, __FILE__ is a compiler directive to insert the name of the source file, and (Permission denied) is the string associated with the current value of *errno*.

If you make the same call, but omit the location:

```c
ApError( ABW_base, errno, "PhAB", "Unable to save file", NULL );
```

then the error dialog is formatted as:

**PhAB: Unable to save file**

146  Chapter 4  •  Ap—PhAB
ApError()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAlert(), PtNotice(), PtPrompt()

“Dialog modules” in the Working with Modules chapter of the Photon Programmer’s Guide
**ApGetDBWidgetInfo()**

Get information about a widget in a widget database

**Synopsis:**

```c
ApDBWidgetInfo_t *ApGetDBWidgetInfo(
    ApDBase_t const *dbase,
    unsigned index,
    ApDBWidgetInfo_t *info );
```

**Arguments:**

- `dbase` A pointer to a widget database that you opened with either `ApOpenDBase()` or `ApOpenDBaseFile()`.
- `index` The index of the widget that you want to get information about. The index of the first widget is 0.
- `info` A pointer to an `ApDBWidgetInfo_t` structure that the function fills with information about the widget; see below.

**Library:**

```
Ap
```

**Description:**

This function extracts information about the widget with a given index in the database specified by `dbase`.

If `index` is greater than or equal to the number of widgets in the database, the function returns NULL. Otherwise, it puts information about a widget in the buffer pointed to by `info`, and returns `info`.

The `ApDBWidgetInfo_t` structure contains at least the following members:

- `const char *wgt_name`
  - The widget’s name.
- `const char *wgt_class`
  - The name of the widget’s class.
ApGetDBWidgetItem() 

int parent_index

The index of the widget’s parent in the database, or -1 if it has no parent.

int level

How deep in the hierarchy the widget is. The top level in the hierarchy is 1.

Returns:

The same pointer as info, or NULL if there isn’t a widget with the given index.

Examples:

ApDBase_t *dbase = ApOpenDBase( ABM_db );

int i;
ApDBWidgetItem_t wi;

for ( i=0; ApGetDBWidgetItem( dbase, i, &wi ); ++i )
printf("#%d: '%s' is a %s, child of #%d, at level %d.\n", i, wi.wgt_name, wi.wgt_class, wi.parent_index, wi.level );

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApGetDBWidgetInfo()

Accessing PhAB Modules from Code chapter of the Photon
Programmer’s Guide
ApGetImageRes()
Extract the image data from a widget in a widget database

Synopsis:

```c
#include <Ap.h>

PhImage_t *ApGetImageRes(
    ApDBase_t const *dbase,
    char const *wgt_name);
```

Arguments:

- `dbase` The widget database pointer returned from `ApOpenDBase()` or `ApOpenDBaseFile()`.
- `wgt_name` The name of widget within the database that has the image that you want to extract.

Library:

**Ap**

Description:


This function is mainly used to perform simple animation. You can create a series of tiles, using any widget that supports images, in a PhAB widget database; to create the animation, cycle through the tiles by pulling out the images in sequence, updating another widget that is visible within the application window.

Returns:

A pointer to a `PhImage_t` structure, or NULL if the widget or image data couldn’t be found.
ApGetImageRes()

This function returns a pointer into the widget database; don’t close the database while still using the image. If you must close the widget database, you should first use PiDuplicateImage() to make a copy of the image and associated structures.

1. Make a copy of the image data, such as image pixel data, alpha channel, palette, etc, using PiDuplicateImage().

2. Update the PhImage_t structure to point to the new copies of the data.

Examples:

```c
PhImage_t *image;

mydbase = ApOpenDBase( ABM_mypicture );
image = ApGetImageRes( mydbase, "myimage" );

/* update the label widget with the new image */
if ( image ) {
    PtSetResource( ABW_label_wgt, Pt_ARG_LABEL_IMAGE,
                  image, 0 );
}
```

Classification:

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No
ApGetImageRes()

See also:

ApOpenDBase(), ApOpenDBaseFile(), PgDrawPhImage*(), PhCreateImage(), PhImage_t, PhMakeGhostBitmap(), PhMakeTransBitmap(), PhMakeTransparent(), PhReleaseImage(), PmMemCreateMC(), PmMemFlush(), PxLoadImage()

“Images” and “Animation” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
ApGetInstance()

Get the module link instance pointer for a widget

Synopsis:

```c
#include <Ap.h>

PtWidget_t *ApGetInstance( PtWidget_t *widget );
```

Arguments:

- `widget` A pointer to the widget that you want to get the module link instance pointer for.

Library:

Ap

Description:

`ApGetInstance()` is used to obtain the widget’s module link instance pointer. For most modules, PhAB generates manifests that let you access the widgets within that module directly, provided only one instance of the module exists at a time.

Because window modules allow you to have multiple instances, you can’t always use the direct access manifests; the manifest will only be valid for the last instance of the window. For windows, you may need either to save the module link_instance pointer when creating the window, or to use this function within the callback to determine the origin of the callback.

Returns:

The module link instance pointer for the widget, or NULL if it can’t be determined.

Examples:

```c
PtArg_t    args[1];

mywindow_callback( PtWidget_t *widget, ... )
{
    PtWidget_t *window;

    /* from which window did this come? */
```
ApGetInstance()

```c
if ( window = ApGetInstance( widget ) ) {
    /* set the widget selected to red */
    PtSetArg( &args[0], Pt_ARG_FILL_COLOR, Pg_RED, 0 );
    PtSetResources( ApGetWidgetPtr( window, ApName( widget ) ), 1, args );
}
return( Pt_CONTINUE );
```

An interesting observation about this callback example is that it doesn’t know which window is being modified or which widget gets changed. It acts on whatever widget the user selects.

**Classification:**

**Photon**

**Safety**

<table>
<thead>
<tr>
<th></th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

ApGetWidgetPtr()

“Handling multiple instances of a window” in the Working with Code chapter of the Photon *Programmer’s Guide*
**ApGetItemText()**

**Get the text for a menu item**

**Synopsis:**

```c
#include <Ap.h>

char * ApGetItemText( ApMenuLink_t *menu, 
                      int item_name);
```

**Arguments:**

- `menu` A pointer to a PhAB menu link structure.
- `item_name` The ABN name of the menu item, as generated by PhAB.

**Library:**

Ap

**Description:**

`ApGetItemText()` is used to extract the text of a menu item in a PhAB menu module. If a language translation is in effect, the translated string is returned rather than the default text built into the application.

**Returns:**

A pointer to a text string or translated text, or NULL if the ABN name is invalid.

---

**CAUTION:** Don’t free the returned text string, or your application will crash.

If you call `ApModifyItemText()` after calling `ApGetItemText()` for the same menu item, the string returned by `ApGetItemText()` becomes invalid.
ApGetItemText()

Examples:

```c
    text = ApGetItemText( &mymenu, ABN_item1 );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


“Changing menu-item text” in the Working with Code chapter of the Photon Programmer’s Guide
ApGetMessage()
Get a message from a message database

Synopsis:

```c
char *ApGetMessage( ApMsgDBase_t *db,
const char *tag );
```

Arguments:

- `db` A pointer to a message database, returned by `ApLoadMessageDB()`.
- `tag` The tag for the message that you want to get.

Library:

`Ap`

Description:

A message database is a file containing textual messages. `ApGetMessage()` can be used to retrieve the message with the given `tag` from the database specified by `db`.

Returns:

The message corresponding to the tag, or NULL if the tag wasn’t found in the database.

Don’t change the message string.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
</tbody>
</table>

`continued...`
ApGetMessage()

**Safety**

| Thread | No |

**See also:**

AlClearTranslation(), AlCloseDBase(), AlGetEntry(), AlGetSize(), AlOpenDBase(), AlReadTranslation(), AlSaveTranslation(), AlSetEntry(), ApCloseMessageDB(), ApLoadMessageDB()

International Language Support chapter of the Photon Programmer’s Guide
ApGetTextRes()
Get a translated text string from a widget database

Synopsis:
#include <Ap.h>

char *ApGetTextRes( ApDBase_t const *db,
                   char const *name );

Arguments:
dbase A pointer to a widget database that you opened with either
       ApOpenDBase() or ApOpenDBaseFile().
name The instance name of the text string that you want to
       translate.

Library:
Ap

Description:
ApGetTextRes() extracts the translated text string from the database
identified by db using name as the widget instance name identifier.

Returns:
The translated text string if successful, NULL if not found, or name or
db if invalid.

Examples:
ApDBase_t *mydbase;
char *string;

mydbase = ApOpenDBase( ABM_mypicture);
string = ApGetTextRes( mydbase, "msg1" );

Classification:
Photon
ApGetTextRes()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Accessing PhAB Modules from Code and International Language Support chapters of the Photon Programmer’s Guide
ApGetWidgetPtr()
Get the instance pointer for a widget in a given module

Synopsis:

```c
#include <Ap.h>

PtWidget_t *ApGetWidgetPtr( PtWidget_t *link_instance,
                            int wgt_name );
```

Arguments:

- `link_instance` - The link instance of the module that you want to search for the widget.
- `wgt_name` - The ABN_name of the widget that you want to find. PhAB automatically generates these name values for you when you generate your code.

Library:

Ap

Description:

ApGetWidgetPtr() is used to obtain the widget’s instance pointer within the specified `link_instance`. For most modules, PhAB generates manifests that let you access the widgets within the module directly, provided only one instance of the module exists at a time.

You can also use AbGetABW() to get the widget instance pointer more efficiently, though AbGetABW() does not work when there are multiple instances of the widget’s window module.

Because window modules allow you to have multiple instances you can’t always use the direct access manifests; the manifest will only be valid for the last instance of the window. For windows, you may need either to save the `link_instance` pointer when creating the window, or to use this function to find the correct `link_instance` for the callback.

Once you determine the module link instance, you’ll need to extract the widget pointer from it.
ApGetWidgetPtr()

Returns:

A pointer to the widget within the module link_instance, or NULL if it wasn’t found.

Examples:

```c
PtArg_t args[1];

mywindow_callback( PtWidget_t *widget, ... )
{
    PtWidget_t *window;

    /* from which window did this come? */
    if ( window = ApGetInstance( widget ) ) {
        /* set the widget selected to red */
        PtSetArg( &args[0], Pt_ARG_FILL_COLOR,
                  Pg_RED, 0 );
        PtSetResources( ApGetWidgetPtr( window, ABN_mywidget ),
                        1, args );
    }

    return( Pt_CONTINUE );
}
```

If you compare this example with the one for ApGetInstance(), you’ll notice a subtle difference. This example modifies a specific widget identified by ABN_mywidget, while the other example modifies any widget affected by the callback.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
ApGetWidgetPtr()

See also:

AbGetABW(), ApGetInstance()

“Handling multiple instances of a window” in the Working with Code chapter of the Photon Programmer’s Guide
ApInfo_t
Data structure for information passed to PhAB callbacks and setup functions

Synopsis:

```c
#include <Ap.h>

typedef struct {
    short reason;
    PtWidget_t *widget;
} ApInfo_t;
```

Description:

This structure is used as the second argument to most functions generated by PhAB, including code callbacks and module-setup functions.

The possible values for `reason` are:

- `ABR_PRE_REALIZE`  
  Pre-realize setup function
- `ABR_POST_REALIZE`  
  Post-realize setup function
- `ABR_CODE`  
  Code-type callback
- `ABR_DONE`  
  Done-type callback
- `ABR_CANCEL`  
  Cancel-type callback

The `widget` argument is a pointer to the widget that invoked the callback function. This is very useful in setup functions to determine which widget initiated the link callback.

Classification:

Photon
ApInfo_t

See also:

`ApWidget()`

`PtCallback_t` in the Photon Widget Reference

“Module setup functions” and “Code-callback functions” in the Working with Code chapter of the Photon Programmer’s Guide
ApInstanceName()
Return the widget’s instance name string

Synopsis:

```c
#include <Ap.h>

char * ApInstanceName( PtWidget_t *widget );
```

Arguments:

- `widget` A pointer to the widget whose instance name you want to get.

Library:

Ap

Description:

`ApInstanceName()` returns a widget’s instance string name if it’s defined. The string name is the name given to the widget in the Widget Instance Name field of the Control Panel. By default, string names are not saved with the widget, in order to save memory.

To embed string names in the widget, you must enable this feature:

1. In PhAB, select the Startup Info/Modules command from the Application menu.
2. In the Application Startup Information dialog, click the Store Names for ApInstanceName() button.

Returns:

The widget’s instance name string, or NULL if the widget doesn’t have string name data attached.

Examples:

```c
my_callback( PtWidget_t *widget, ... )
{
    if ( strcmp( ApInstanceName( widget ),
               "done_button" ) == 0 ) {
        /* done button processing */
```
ApInstanceName()


Classification:

Photon


Safety

Interrupt handler    No
Signal handler       No
Thread               No

See also:

“Including instance names” in the Working with Applications chapter of the Photon Programmer’s Guide
ApLoadMessageDB()

Load a message database

Synopsis:

ApMsgDBase_t *ApLoadMessageDB( ApMsgDBase_t *db,
                    const char *name );

Arguments:

    db    NULL, or a pointer to a message database that’s already open to which you want to add the loaded messages.

    name   The name of the message database that you want to load.

Library:

Ap

Description:

This function loads the message database with the given name. It searches for the file based on the value of the ABLPATH and the current language:

- If a language isn’t defined, the message database is loaded. It must have a name of name.mdb.

- If a language is defined, ApLoadMessageDB() looks for a translation file called name.language. Translation files can be created using PhAB translation editor — it can handle message databases.

Note that in an application that uses PhAB DLLs, this function performs the language search based on the location of the executable or DLL associated with the current context. If there’s no current context, a lookup that would search the directory of an executable or DLL is skipped.

Returns:

A pointer to the new database, or NULL if it couldn’t be opened.
ApLoadMessageDB()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

AlClearTranslation(), AlCloseDBase(), AlGetEntry(), AlGetSize(), AlOpenDBase(), AlReadTranslation(), AlSaveTranslation(), AlSetEntry(), ApGetMessage()

International Language Support chapter of the Photon Programmer’s Guide
**ApModalWait()**

*Process Photon events until a given widget is destroyed*

**Synopsis:**

```c
int ApModalWait( PtWidget_t *widget,
                 unsigned flags );
```

**Arguments:**

- **widget**
  A pointer to the widget whose destruction you want to wait for.

- **flags**
  Any combination of the following:
  - Ap_MODAL_BLOCK_WINDOWS — block all of the application’s windows, except the one that contains `widget`, using `PtMakeModal()` while this function is running.
  - Pt_EVENT_PROCESS_PREVENT — temporarily turn your thread into a nonreader: `PtModalBlock()` blocks on a condvar rather than processing events.
  - Pt_EVENT_PROCESS_ALLOW — make sure that `PtModalBlock()` processes events rather than blocking on a condvar.

**Library:**

```
Ap
```

**Description:**

`ApModalWait()` processes Photon events until the given widget is destroyed.

**Returns:**

0 on success, or -1 if `PtModalBlock()` fails.
**ApModalWait()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PtBlockAllWindows(), PtBlockWindow(), PtMakeModal(), PtModalBlock(), PtUnblockWindows()

ApModifyItemAccel()
Modify the keyboard shortcut for a menu item

Synopsis:

```c
int ApModifyItemAccel( ApMenuLink_t *menu,
                        int item_no,
                        const char *new_text,
                        int new_flags );
```

Arguments:

- `menu` A pointer to a PhAB menu link structure.
- `item_num` The number of the menu item, as generated by PhAB.
- `new_text` A pointer to the new shortcut for the menu item.
- `new_flags` Zero, or the following:
  - AB_ITEM_ACCEL_STRDUP — duplicate the string and store the copy in the menu item. If memory was allocated for the former keyboard shortcut, the function frees the space.

Library:

Ap

Description:

`ApModifyItemAccel()` modifies the keyboard shortcut for a menu item in a PhAB menu module.

If you don’t set AB_ITEM_ACCEL_STRDUP in the `new_flags` argument, `ApModifyItemAccel()` stores the address given by `new_text` instead of making a copy of the string pointed to by `new_text`. In this case, don’t modify the string after calling this function.

You can call `ApModifyItemAccel()` at any time to set the menu item’s shortcut, and the effect will be seen when the menu is next displayed.
ApModifyItemAccel()

Returns:

0  The item number isn’t valid.
1  Success.
-1  An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:


“Changing menu-item text” in the Working with Code chapter of the Photon Programmer’s Guide
**ApModifyItemState()**

*Modify the state of menu items*

**Synopsis:**

```c
#include <Ap.h>

int ApModifyItemState( ApMenuLink_t *menu,
                        int state,
                        int item_no,
                        ...,
                        NULL);
```

**Arguments:**

- `menu` A pointer to a PhAB menu link structure.
- `state` The state you want to set for the menu items:
  - AB_ITEM_DIM — disabled.
  - AB_ITEM_NORMAL — enabled and not set.
  - AB_ITEM_SET — set on (toggle item).
- `item_no, item_no, ...`, NULL
  A list of menu items, followed by NULL to terminate the list. The menu items are values that are generated by PhAB for each menu item in a menu module.

**Library:**

*Ap*

**Description:**

*ApModifyItemState()* modifies the state of menu items in a PhAB menu module.

You can call *ApModifyItemState()* at any time to set the menu item states, and the effect will be seen when the menu is displayed. This lets you set menu item states as soon as conditions within your application change.
ApModifyItemState()

Returns:

1 Successful completion

Examples:

In this example, mymenu is a pointer to the address of the menu name, which is equivalent to the instance name for the menu module.

    /* Dim the ABN_opt1, ABN_opt2, and ABN_opt3 menu items. */
    ApModifyItemState( &mymenu, AB_ITEM_DIM, ABN_opt1, ABN_opt2,
                       ABN_opt3, NULL );

Classification:

    Photon

    Safety
    | Interrupt handler | No |
    | Signal handler    | No |
    | Thread            | No |

See also:


“Enabling, disabling, or toggling menu items” in the Working with Code chapter of the Photon Programmer’s Guide
ApModifyItemText()
Modify the text for a menu item

Synopsis:

```c
#include <Ap.h>
int ApModifyItemText( ApMenuLink_t *menu,
                      int item_num,
                      char const *new_text );
```

Arguments:

- `menu` A pointer to a PhAB menu link structure.
- `item_num` The number of the menu item, as generated by PhAB.
- `new_text` A pointer to the replacement menu item text.

Library:

- Ap

Description:

`ApModifyItemText()` modifies the text for a menu item in a PhAB menu module.

`ApModifyItemText()` stores the address given by `new_text`; it doesn’t make a copy of the string pointed to by `new_text`. Don’t modify the string after calling this function.

You can call `ApModifyItemText()` at any time to set the menu item text, and the effect will be seen when the menu is displayed. This allows to you set menu item text as soon as conditions within your application change.

Returns:

- 0 The item number isn’t valid.
- 1 Success.
ApModifyItemText()

Examples:

In this example, mymenu is a pointer to the address of the menu name, which is equivalent to the instance name for the menu module.

```c
/* Change ABN_opt1 to say "New Option 1 Text" */
ApModifyItemText( &mymenu, ABN_opt1, "New Option 1 Text" );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


“Changing menu-item text” in the Working with Code chapter of the Photon Programmer’s Guide
ApModuleFunction()
Specify the setup function for a PhAB internal link callback

Synopsis:
#include <Ap.h>

void ApModuleFunction( ApEventLink_t *link_callback,
        int (* function)(),
        int realize_flags );

Arguments:
link_callback  The ABM_link callback that you want to specify a setup function.
function       The setup function that you want to register for the link callback.
realize_flags  A flag that indicates when the setup function should be called; one of:
                 • AB_FUNC_PRE_REALIZE
                 • AB_FUNC_POST_REALIZE
                 • AB_FUNC_BOTH

Library:
Ap

Description:
ApModuleFunction() is used to specify the setup function for a PhAB internal link callback.

When you create an internal link callback, PhAB lets you specify the setup function. ApModuleFunction() lets you change that setup function. The new function is retained until changed again.

Examples:
ApModuleFunction( ABM_mydialog, setup_module, AB_FUNC_BOTH );
ApCreateModule( ABM_mydialog, NULL, NULL );

setup_module( PtWidget_t *widget, ... ) {
        /* setup processing for module */
    }
ApModuleFunction()

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

ApCreateModule(), ApModuleLocation()

Accessing PhAB Modules from Code chapter of the Photon
Programmer’s Guide
ApModuleLocation()

Specify the module location for a PhAB internal link

Synopsis:

```c
#include <Ap.h>

void ApModuleLocation( ApEventLink_t *link_callback,
                       int loc_type,
                       int x_offset,
                       int y_offset );
```

Arguments:

- `link_callback`: The ABM_link callback for the module whose location you want to specify.
- `loc_type`: Where to place the module; one of the following:
  - `AB_LOC_BELOW_WGT`
  - `AB_LOC_ABOVE_WGT`
  - `AB_LOC_RIGHT_WGT`
  - `AB_LOC_LEFT_WGT`
  - `AB_LOC_TOP_LEFT`
  - `AB_LOC_TOP_RIGHT`
  - `AB_LOC_BOT_LEFT`
  - `AB_LOC_BOT_RIGHT`
  - `AB_LOC_CENTER`
  - `AB_LOC_REL_CURSOR`
  - `AB_LOC_REL_MODULE`
  - `AB_LOC_DEFAULT` — let the window manager determine the position. The offsets are ignored.
  - `AB_LOC_ABSOLUTE` — `x_offset` and `y_offset` are the absolute coordinates for the module.

- `x_offset`, `y_offset`: The horizontal and vertical offsets for the module, relative to the given location.


**ApModuleLocation()**

**Library:** Ap

**Description:**

`ApModuleLocation()` is used to specify the module location for a PhAB internal link.

The `x_offset` and `y_offset` pixel values are applied to the location determined from the `loc_type`. For example, if you want a dialog to appear 100 pixels from the top right edge of the screen, set the `x_offset` to -100 and the `loc_type` to `AB_LOC_TOP_RIGHT`.

When you create an internal link in PhAB, you can set the location. This function lets you change that value if required. The new location will remain in effect until changed again.

**Examples:**

```c
/* place the module in the center of the screen */
ApModuleLocation( ABM_mydialog, AB_LOC_CENTER, 0, 0 );
ApCreateModule( ABM_mydialog, NULL, NULL );
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`ApModuleFunction()`

Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
**ApModuleParent()**

Specify the parent for a window or dialog module

**Synopsis:**

```c
#include <Ap.h>

void ApModuleParent( ApEventLink_t *link_callback, 
                       int parent, 
                       PtWidget_t *widget );
```

**Arguments:**

- `link_callback`  The ABM_link callback for the module whose parent you want to specify.
- `parent`  One of the following:
  - `AB_PARENT` — the `widget` argument points to the module’s new parent.
  - `AB_NO_PARENT` — the module should have no parent. The `widget` argument is ignored.
- `widget`  A pointer to the widget that you want to be the parent of the module.

**Library:**

Ap

**Description:**

`ApModuleParent()` lets you specify the parent module for a window or dialog module.

**Examples:**

Change `my_window` to have no parent at all:

```c
ApModuleParent( ABM_my_window, AB_NO_PARENT, NULL );
ApCreateModule( ABM_my_window, NULL, NULL );
```

Change `new_window` to have the base window as its parent:

```c
ApModuleParent( ABM_new_window, AB_PARENT, ABW_base );
ApCreateModule( ABM_new_window, NULL, NULL );
```
ApModuleParent()

Classification:
Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

Accessing PhAB Modules from Code chapter of the Photon

Programmer’s Guide
ApName()

Return a PhAB name value for the specified widget

Synopsis:

```c
#include <Ap.h>

int ApName( PtWidget_t *widget );
```

Arguments:

- `widget` A pointer to the widget whose PhAB name you want to get.

Library:

- `Ap`

Description:

`ApName()` returns a PhAB name value for the specified widget. You can compare this name value with the global name values that PhAB generates for your application code. These name values make it easier to access and compare widgets in callback functions. It also lets you use the same callback function for more than one widget.

Returns:

A PhAB name value, or -1 if the widget doesn’t have a name.

Examples:

```c
my_callback( PtWidget_t *widget, ... )
{
    if ( ApName( widget ) == ABN_widget1 ) {
        /* do widget1 processing */
    } else {
        if ( ApName( widget ) == ABN_widget2 ) {
            /* do widget2 processing */
        } else {
            /* do something else? */
        }
    }
}
```
ApName()

**ABN_widget1** and **ABN_widget2** are name values generated by PhAB for widgets in your application with instance names of *widget1* and *widget2*.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**
“Variables and manifests” in the Working With Code chapter of the Photon Programmer’s Guide
ApOpenDBase()
Open a module as a widget database

Synopsis:
#include <Ap.h>

ApDBase_t *ApOpenDBase(
   ApEventLink_t const *link_callback);

Arguments:
link_callback The ABM_internal link manifest, generated by PhAB, of the module that you want to open.

Library:
Ap

Description:
ApOpenDBase() opens the given module as a widget database. Typically, the module is a picture, but it can also be a window or dialog.

ApOpenDBaseFile() lets you open an external module file as a widget database.

Returns:
A pointer to a PhAB picture database structure, or NULL for failure.

Examples:
ApDBase_t *mydbase;
PtArg_t args[2];

mydbase = ApOpenDBase( ABM_mypicture );

PtSetArg( &args[0], Pt_ARG_TEXT_STRING, "This Widget", 0 );
ApCreateWidget( mydbase, "my_label_widget", 10, 10, 1, args );

PtSetArg( &args[0], Pt_ARG_TEXT_STRING, "That Widget", 0 );
PSetArg( &args[1], Pt_ARG_FILL_COLOR, Pg_WHITE, 0 );
ApCreateWidget( mydbase, "my_label_widget", 10, 30, 2, args );
ApOpenDBase()

ApCloseDBase( mydbase );

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
Synopsis:

```c
#include <Ap.h>

ApDBase_t *ApOpenDBaseFile( char const *path );
```

Arguments:

- `path` The path of the external module file that you want to open.

Library:

- `Ap`

Description:

`ApOpenDBaseFile()` opens an external module file as a widget database. Typically, the module is a picture, but it can also be a window or dialog. External module files supported have a `.wgtp`, `.wgti`, `.wgtd`, or `.wgtw` extension.

`ApOpenDBase()` opens the module specified by a `ABM` internal link manifest that’s generated by PhAB.

Before calling `ApOpenDBaseFile()`, you should call `ApAddClass()` for each widget class that you’ll likely encounter in the database. This function adds the widget classes to the internal widget class table.

Note that this function associates the widget database with the current PhAB context. Only language translations from that context are applied when you create widgets using the database. If you don’t want any language translations to be applied to an external database, set the current context to NULL when you open the database:

```c
ApSetContext( NULL );
ApDBase_t *db = ApOpenDBaseFile( fname );
ApSetContext( &AbContext ); // Restore the program’s context
```

A typical PhAB application has only one PhAB context; only applications that load PhAB-created DLLs have to deal with multiple contexts.
ApOpenDBaseFile()

contexts. For more information about creating DLLs from PhAB applications, see “Making a DLL out of a PhAB application” in the Generating, Compiling, and Running Code chapter of the Photon Programmer’s Guide.

Returns:
A pointer to a PhAB picture database structure, or NULL for failure.

Examples:

```c
ApDBase_t *mydbase;
PtArg_t args[2];

/* Add the widget classes that are in the database
to the internal table. */
ApAddClass("PtLabel", &PtLabel);
mydbase = ApOpenDBaseFile( "mypicture.wgtp" );

PtSetArg( &args[0], Pt_ARG_TEXT_STRING,
    "This Widget", 0 );
ApCreateWidget( mydbase, "my_label_widget", 10, 10, 1,
    args );

PtSetArg( &args[0], Pt_ARG_TEXT_STRING,
    "That Widget", 0 );
PtSetArg( &args[1], Pt_ARG_FILL_COLOR, Pg_WHITE, 0 );
ApCreateWidget( mydbase, "my_label_widget", 10, 30, 2,
    args );

ApCloseDBase( mydbase );
```

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
</tbody>
</table>

continued...
ApOpenDBaseFile()

Safety

| Thread | No |

See also:


Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
ApRemoveClass()  
Remove a widget class

Synopsis:
```
#include <Ap.h>

int ApRemoveClass(
    char const * class_name_string,
    PtWidgetClassRef_t * const * wgt_class);
```

Arguments:
- `class_name_string`  
The name of the class that you want to remove (for example, "PtButton").
- `wgt_class`  
A predefined widget class; specify the name of the class preceded by an ampersand (for example, &PtButton).

Library:
Ap

Description:
This function lets you remove widget classes that you previously added by calling `ApAddClass()`.

If you’ve loaded a DLL that adds classes, you should remove the classes before you unload the DLL.

Returns:
- 0  Success.
- -1  The widget wasn’t registered with `ApAddClass()`, or the `wgt_class` pointer isn’t the same as when you registered the class.
ApRemoveClass()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApAddClass(), ApRemoveContext()

“Making a DLL out of a PhAB application” in the Generating, Compiling, and Running Code chapter, Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
**ApRemoveContext()**

Remove the PhAB context from a PhAB application that you’re using as a DLL

**Synopsis:**

```c
int ApRemoveContext( ApContext_t *context );
```

**Arguments:**

- `context` A pointer to the context to remove. This argument should be the address of `AbContext`, a global data structure that PhAB puts in `abmain.c`.

**Library:**

`Ap`

**Description:**

This function removes a PhAB context from a PhAB application that you’re using as a DLL. Call this function from the DLL’s cleanup function.

You must call `ApRemoveContext()` as many times as you successfully called `ApAddContext()`. After you’ve called `ApRemoveContext()`, your DLL must not call any PhAB functions.

**Returns:**

0 on success, or -1 on failure.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

194 Chapter 4 • Ap—PhAB
ApRemoveContext()

See also:

ApAddContext(), ApRemoveClass()

“Making a DLL out of a PhAB application” in the Generating, Compiling, and Running Code chapter of the Photon Programmer’s Guide
ApResClose()
Close the file of module resource records

Synopsis:
void ApResClose ( void );

Library:
Ap

Description:
All PhAB applications have module resource records bound into the executable. A PhAB application opens its own binary file to access these records, and keeps the file open for better performance until the application loses focus. Once the file is closed, it isn’t reopened unless a module record is required.

If your application has only a single base window or dialogs that are infrequently used, you can force the binary file to be closed by calling this function. This reduces the number of file descriptors in use, as well as freeing resources used for accessing other nodes in a networking environment.

Examples:
In the post-realize callback of the base window:

if (apinfo->reason == ABR_POST_REALIZE) {
    ApResClose ();
}

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
ApResClose()
ApSaveDBaseFile()
Save a widget database as an external file

Synopsis:
#include <Ap.h>

int ApSaveDBaseFile( ApDBase_t const *db,
   char const *path );

Arguments:
dbase A pointer to a widget database that you opened with either
ApOpenDBase() or ApOpenDBaseFile().
path The name of the file in which you want to save the
database.

Library:
Ap

Description:
ApSaveDBaseFile() saves a widget database as an external file. Both
internally bound widget databases and previously loaded external
widget databases can be saved as external files.

Returns:
0 Success.
-1 Failure.

Examples:
/* Open a PhAB picture database, delete my_icon,
   and save the database again. */
ApDBase_t *my_dbase;

my_dbase = ApOpenDBaseFile( "/home/me/mydbase.wgtp" );

ApDeleteDBWidget( my_dbase, "my_icon" );

ApSaveDBaseFile( my_dbase, "/home/me/mydbase.wgtp" );
ApSaveDBaseFile()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:


Accessing PhAB Modules from Code chapter of the Photon Programmer’s Guide
**ApSetContext()**

*Set the PhAB context*

**Synopsis:**

```c
#include <Ap.h>

ApContext_t *ApSetContext( ApContext_t *context );
```

**Arguments:**

- `context`  
  A pointer to the context you want to set the current PhAB context to. This argument should be the address of `AbContext`, a global data structure that PhAB puts into `abmain.c`, or NULL for no context.

**Library:**

*Ap*

**Description:**

`ApSetContext()` makes the given `context` current, and returns the previous current context. Both can be NULL; but if you pass a non-NULL pointer, it must point to a registered context.

At program startup, the program’s PhAB context `AbContext` is made current. You can unset it or change it to a different context by calling `ApSetContext()`. You may want to do this in a DLL which calls `Ap*()` functions that use the current context.

In an application that doesn’t involve DLLs, you can use this function to set the context to NULL when you’re calling `ApOpenDBaseFile()`, to prevent the database from using your application’s language translations if the file happens to contain widgets with the same name as one of your widgets.

**Returns:**

A pointer to the previous current context.
ApSetContext()

Examples:

This example sets the program’s current context to NULL so that the widget database opened by ApOpenDBaseFile() doesn’t use the current language translation:

```c
ApContext_t *old = ApSetContext( NULL );
ApDBase_t *db = ApOpenDBaseFile( fname );
ApSetContext( old ); // Restore the program’s context
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

“Making a DLL out of a PhAB application” in the Generating, Compiling, and Running Code chapter of the Photon Programmer’s Guide
**ApSetTranslation()**

Change the current translation to another language

**Synopsis:**

```c
#include <Ap.h>

int ApSetTranslation( char const *lang_ext );
```

**Arguments:**

`lang_ext` NULL, or the extension for the language that you want to switch to (e.g. `ja_JP` for Japanese).

**Library:**

`Ap`

**Description:**

`ApSetTranslation()` changes the current translation file to the one defined by `lang_ext`. If `lang_ext` is NULL, the translation is set to the default (i.e. the original language used in the application). If the extension pointed to by `lang_ext` is invalid, the translation file isn’t changed.

When you run your application outside of PhAB, it looks for the translation files as follows:

1. in the directories listed in the `ABLPATH` environment variable, if defined. This list takes the form:
   ```bash
   dir:dir:dir:dir
   ```
   Unlike the `PATH` environment variable, the current directory must be indicated by a period, not an empty string. An empty string in `ABLPATH` indicates the directory where the executable is.

2. in the same directory as the executable, if the `ABLPATH` environment variable isn’t defined
ApSetTranslation()

In an application that loads PhAB DLLs, this function changes the current language for all registered contexts (i.e. the program and all DLLs). It first unloads any existing language translations for all the contexts, and then attempts to find a translation file for each context, using the name and location of the executable or DLL to perform the ABLPATH search described above.

Returns:

0  Successful completion.

-1  The translation extension is invalid.

Examples:

/* Set the current translation to German: */

ApSetTranslation( "de_DE");

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApSetTranslation()

International Language Support chapter of the Photon *Programmer’s Guide*
ApWidget()
Determine the widget that initiated a link callback

Synopsis:
#include <Ap.h>

PtWidget_t *ApWidget( PtCallbackInfo_t *cbinfo );

Arguments:

  cbinfo  A pointer to the PtCallbackInfo_t structure (see the
          Photon Widget Reference) that was passed to the callback.

Library:

  Ap

Description:

ApWidget() is used within module setup functions to determine the
widget that initiated the link callback.

Returns:

A pointer to the initiating widget.

Examples:

mysetup_function( ..., PtCallbackInfo_t cbinfo ) {
  if (ApName(ApWidget(cbinfo)) == ABN_widget1) {
    /* setup based on widget1 */
  } else {
    if (ApName(ApWidget(cbinfo)) == ABN_widget2) {
      /* setup based on widget2 */
    } else {
      /* common setup */
    }
  }
  return( Pt_CONTINUE );
}
ApWidget()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

For another method of determining the widget that initiated a link callback, see the ApInfo_t structure.

PtcCallbackInfo_t in the Photon Widget Reference

“Module setup functions” in the Working With Code chapter of the Photon Programmer’s Guide
Chapter 5

mbstr—Multibyte-Character
The `mbstr*` functions have been replaced:

<table>
<thead>
<tr>
<th>Instead of</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbstrblen()</td>
<td>utf8strblen()</td>
</tr>
<tr>
<td>mbstrchr()</td>
<td>utf8strchr()</td>
</tr>
<tr>
<td>mbstrichr()</td>
<td>utf8strichr()</td>
</tr>
<tr>
<td>mbstrirchr()</td>
<td>utf8strirchr()</td>
</tr>
<tr>
<td>mbstrlen()</td>
<td>utf8strlen()</td>
</tr>
<tr>
<td>mbstrnchr()</td>
<td>utf8strnchr()</td>
</tr>
<tr>
<td>mbstrncmp()</td>
<td>utf8strncmp()</td>
</tr>
<tr>
<td>mbstrndup()</td>
<td>utf8strndup()</td>
</tr>
<tr>
<td>mbstrnichr()</td>
<td>utf8strnichr()</td>
</tr>
<tr>
<td>mbstrnlen()</td>
<td>utf8strnlen()</td>
</tr>
<tr>
<td>mbstrstrchr()</td>
<td>utf8strstrchr()</td>
</tr>
</tbody>
</table>
This chapter describes functions that change the draw context.
**PdCreateDirectContext()**

Create a direct-mode context

**Synopsis:**

```
PdDirectContext_t *PdCreateDirectContext( void );
```

**Library:**

ph

**Description:**

This function creates a direct-mode context. The context isn’t activated at this point, and the graphics driver is still operating normally (i.e. it’s still Reply Blocked on Photon).

When an application enters direct mode, it’s requesting that the graphics driver receive draw streams and service messages directly from the application, instead of from Photon. The driver blocks on the application, which is now responsible for telling the graphics driver what to do.

You must target this function at a specific card by calling `PdSetTargetDevice()`.

**Returns:**

A pointer to a `PdDirectContext_t` structure if successful, or NULL on failure.

**Examples:**

```
PdDirectContext_t *DirectMode=NULL;
PhDrawContext_t *Olddc=NULL;
PhRid_t rid_array[10];

DirectMode=PdCreateDirectContext();
if (DirectMode == NULL)
{
    // error code
}

if( PdGetDevices(rid_array, 10) > 0)
{
    PdSetTargetDevice( (PhDrawContext_t *) DirectMode,
```

214 Chapter 6 • Pd—Draw Context
PdCreateDirectContext()

    rid_array[0] );
Olddc=PdDirectStart(DirectMode);
PgSetFillColor(Pg_PURPLE);
PgDrawRect(0,0,300,300,Pg_DRAW_FILL);
PgFlush(); // Draw the purple rect
PdDirectStop(DirectMode);

    // When the driver leaves direct mode, an expose event
    // is emitted, which will erase our rectangle, so we
    // sleep for a bit so we can see that the rectangle
    // was drawn.
    sleep(5);
}

PdReleaseDirectContext(DirectMode);
PhDCSetCurrent(Olddc);

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdDirectStart(), PdDirectStop(), PdReleaseDirectContext(), PdSetTargetDevice()

“Direct mode” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Create an offscreen context

Synopsis:

PdOffscreenContext_t *PdCreateOffscreenContext(
    unsigned long ImageType,
    unsigned short width,
    unsigned short height,
    unsigned long flags);

Arguments:

ImageType The type of image. Can be one of:

- Pg_IMAGE_DIRECT_1555
- Pg_IMAGE_DIRECT_555
- Pg_IMAGE_DIRECT_565
- Pg_IMAGE_DIRECT_888
- Pg_IMAGE_DIRECT_8888
- Pg_IMAGE_PALETTE_BYTE
- 0 (the image type is defined by the current video mode)

For more information about these image types, see PhImage_t.

width, height The dimensions of the context, in pixels.

flags Flags that indicate how you want to create the offscreen context. Set to 0 if there are no restrictions on where the context can be allocated. Can be a combination of:

- Pg_OSC_CRTC_SAFE — tell the driver that you want to be able to point the CRT at this context at a later date (using PgSwapDisplay()).
  The driver may change the dimensions and/or image type of the context in order to comply with this flag.
**PdCreateOffscreenContext()**

- **Pg OSC_MAIN_DISPLAY** — create an offscreen context from the currently displayed screen. It doesn’t make a new offscreen context in video ram, but just wraps the displayed screen with a `PdOffscreenContext_t` structure.

- **Pg OSC_MEM_PAGE_ALIGN** — ensure that the offscreen context that’s created is aligned to `__PAGESIZE` (4K on an x86). You need to specify this flag in order to use `PdGetOffscreenContextPtr()` with this context.

- **Pg OSC_MEM_SYS_ONLY** — create an offscreen context in system RAM only. If there is not enough system RAM to create the context, the function fails.

- **Pg OSC_MEM_2D_WRITABLE** — create an offscreen context that the graphic card’s 2D engine can write to. The offscreen context is created in video RAM only to support hardware acceleration. If there is not enough video RAM to create the context, the function fails.

- **Pg OSC_MEM_2D_READABLE** — create an offscreen context that is readable by the graphic card’s 2D engine. The offscreen context is created in video RAM only to support hardware acceleration. If there is not enough video RAM to create the context, the function fails.

- **Pg OSC_MEM_HINT_CPU_READ** — create an offscreen context that is optimized by the driver for fast reading by the CPU. Use this flag if you are doing unaccelerated rendering or using `PdGetOffscreenContextPtr()`. This flag has a lower priority than other flags. For example, if conditions conflict for satisfying the requirements of this flag and `Pg OSC_MEM_2D_READABLE`, this flag is ignored.
**PdCreateOffscreenContext()**

- **PgOSC_MEM_HINT_CPU_WRITE** — create an offscreen context that is optimized by the driver for fast writing by the CPU. Use this flag if you are doing unaccelerated rendering or using `PdGetOffscreenContextPtr()`. This flag has a lower priority than other flags. For example, if conditions conflict for satisfying the requirements of this flag and `PgOSC_MEM_2D_WRITABLE`, this flag is ignored.

**Library:**

`ph`

**Description:**

This function creates an offscreen context.

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PdCreateOffscreenContext()` blocks until the operation is complete.

If there isn’t enough space in video RAM to create the offscreen context, the behavior of this function depends on the driver; most drivers allocate space in system memory instead (if none of `SYS_ONLY`, `2D_WRITABLE`, or `2D_READABLE` are set in `flags`).

**Returns:**

A pointer to a `PdOffscreenContext_t`, or NULL if an error occurred.

**Classification:**

Photon
PdCreateOffscreenContext()

Safety

<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdDupOffscreenContext(), PdGetOffscreenContextPtr(),
PdOffscreenContext_t, PdSetOffscreenTranslation(),
PdSetTargetDevice(), PgContextBlit(), PgSwapDisplay(),
PhDCCreate(), PhDCRelease()

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PdCreateOffscreenLock()**

Create a lock for an offscreen context

**Synopsis:**

```c
int PdCreateOffscreenLock(
    PdOffscreenContext_t *osc,
    PdOSCCreateLockParams_t *params);
```

**Arguments:**

- `osc` A pointer to the `PdOffscreenContext_t` structure for the offscreen context.
- `params` A pointer to a `PdOSCCreateLockParams_t` structure that defines the parameters for the lock; see below.

**Library:**

ph

**Description:**

This function creates an offscreen lock in an offscreen context.

This function doesn’t lock the offscreen context; to do that, call `PdLockOffscreen()`.

The `PdOSCCreateLockParams_t` structure includes the following members:

- `uint32_t flags` Flags, including:
  - `PgOSC_LOCK_SIG` — register a signal to be dropped if a request is made to remove the offscreen context while it’s locked.

- `int sig` The signal to drop on the current hard locking process if the context needs to be destroyed.
  This signal is used only if you set `PgOSC_LOCK_SIG` in the `flags` member.
You can’t lock the primary display unless the application is in direct mode.

**Returns:**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOK</td>
<td>The lock was successfully created.</td>
</tr>
<tr>
<td>PgOSC_LOCK_ALREADY_CREATED</td>
<td>The lock has already been created for this offscreen context.</td>
</tr>
<tr>
<td>PgOSC_CREATE_LOCK_FAILED</td>
<td>The lock couldn’t be created. The most likely reason is that io-graphics is running remotely.</td>
</tr>
</tbody>
</table>

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


“Offscreen locks” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PdDestroyOffscreenLock()**

*Destroy a lock for an offscreen context*

**Synopsis:**

```c
int PdDestroyOffscreenLock(
    PdOffscreenContext_t *osc);
```

**Arguments:**

- `osc` A pointer to the `PdOffscreenContext_t` structure for an offscreen context, as returned by `PdCreateOffscreenContext()`.

**Library:**

`ph`

**Description:**

This function removes an offscreen lock from the offscreen context, `osc`.

If you’ve locked the context, call `PdUnlockOffscreen()` to unlock it before destroying the lock.

**Returns:**

- `EOK` The lock was successfully destroyed.
- `Pg_OSC_LOCK_INVALID` The lock didn’t exist.

**Classification:**

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No

*continued...*
**PdDestroyOffscreenLock()**

**Safety**

| Thread | No |

**See also:**


“Offscreen locks” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
PdDirectStart()

Enter direct mode

Synopsis:

```c
PhDrawContext_t *PdDirectStart(
    PdDirectContext_t *DirectContext);
```

Arguments:

- **DirectContext**
  A pointer to the `PdDirectContext_t` structure for a direct context, as returned by `PdCreateDirectContext()`.

Library:

- ph

Description:

This function puts the application into direct mode. On successful completion of this call, the graphics driver is blocked on the application that’s awaiting rendering services. The `DirectContext` is now the default context for the application.

This call blocks until the operation is complete.

Returns:

A pointer to the previous draw context on success, or NULL on failure.

Examples:

See `PdCreateDirectContext()`.

Classification:

- Photon
PdDirectStart()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdCreateDirectContext(), PdDirectStop(), PdReleaseDirectContext()

“Direct mode” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PdDirectStop()**
*Leave direct mode*

**Synopsis:**

```c
PhDrawContext_t *PdDirectStop(
    PdDirectContext_t *DirectContext);
```

**Arguments:**

- **DirectContext**
  A pointer to the `PdDirectContext_t` structure for a direct context, as returned by `PdCreateDirectContext()`.

**Library:**

`ph`

**Description:**

This function takes the application out of direct mode, but doesn’t destroy the direct-mode context.

---

This call blocks until the operation is complete.

**Returns:**

The default draw context (draw through Photon).

**Examples:**

See `PdCreateDirectContext()`.

**Classification:**

Photon

---

**Safety**

- Interrupt handler: No

*continued...*
PdDirectStop()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdCreateDirectContext(), PdDirectStart(), PdReleaseDirectContext()

“Direct mode” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PdDupOffscreenContext()**

*Duplicate an offscreen context*

**Synopsis:**

```c
PdOffscreenContext_t *
PdDupOffscreenContext(  
PdOffscreenContext_t *context,  
unsigned long flags );
```

**Arguments:**

- `context` A pointer to the context that you want to duplicate, as returned by `PdCreateOffscreenContext()`, or NULL if you want to duplicate the screen buffer.
- `flags` Flags that indicate how you want to create the offscreen context:
  - `PgOSC_CRTC_SAFE` — tell the driver that you want to be able to point the CRT at this context at a later date (using `PgSwapDisplay()`). The driver may change the dimensions and/or image type of the context in order to comply with this flag.
  - `PgOSC_MEM_PAGE_ALIGN` — ensure that the offscreen context that’s created is aligned to `__PAGESIZE` (4K on an x86). You need to specify this flag in order to use `PdGetOffscreenContextPtr()` with this context.
  - `PgOSC_MEM_SYS_ONLY` — create an offscreen context in system RAM only. If there is not enough system RAM to create the context, the function fails.
  - `PgOSC_MEM_2D_WRITABLE` — create an offscreen context that the graphic card’s 2D engine can write to. The offscreen context is created in video RAM only to support hardware acceleration. If there is not enough video RAM to create the context, the function fails.
  - `PgOSC_MEM_2D_READABLE` — create an offscreen context that is readable by the graphic card’s 2D engine. The offscreen context is created in video RAM only to support hardware acceleration. If there is not
$PdDupOffscreenContext()$

enough video RAM to create the context, the function fails.

- **PgOSC_MEM_HINT_CPU_READ** — create an offscreen context that is optimized by the driver for fast reading by the CPU. Use this flag if you are doing unaccelerated rendering or using $PdGetOffscreenContextPtr()$. This flag has a lower priority than other flags. For example, if conditions conflict for satisfying the requirements of this flag and **PgOSC_MEM_2D_READABLE**, this flag is ignored.

- **PgOSC_MEM_HINT_CPU_WRITE** — create an offscreen context that is optimized by the driver for fast writing by the CPU. Use this flag if you are doing unaccelerated rendering or using $PdGetOffscreenContextPtr()$. This flag has a lower priority than other flags. For example, if conditions conflict for satisfying the requirements of this flag and **PgOSC_MEM_2D_WRITABLE**, this flag is ignored.

**Library:**

$ph$

**Description:**

$PdDupOffscreenContext()$ makes a copy of the given context. This not only creates a context that has the same dimensions and image type, but also copies the image data.

You must target this function at a specific card by calling $PdSetTargetDevice()$. $PdDupOffscreenContext()$ blocks until the operation is complete.
**PdDupOffscreenContext()**

**Returns:**

A pointer to a `PdOffscreenContext_t` structure, or NULL on failure.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PdCreateOffscreenContext()`, `PdGetOffscreenContextPtr()`,
`PdOffscreenContext_t`, `PdSetOffscreenTranslation()`,
`PdSetTargetDevice()`, `PgContextBlit()`, `PgSwapDisplay()`

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
int PdGetDevices( PhRid_t *rid_array,
                  uint32_t max_rids );
```

Arguments:

- `rid_array` An array that the function can fill with the region IDs of the currently available draw contexts.
- `max_rids` The number of entries in the array.

Library:

`ph`

Description:

`PdGetDevices()` fills the provided `rid_array` with up to `max_rids` region IDs of draw devices that are currently available.

Returns:

The number of draw devices currently available on the system. This value may be larger than `max_rids`, which indicates that the provided array isn’t big enough to hold the region IDs for all the draw devices on the system.

Examples:

See `PdCreateDirectContext()`.

Classification:

- Photon

Safety

- Interrupt handler  No

continued...
**PdGetDevices()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdSetTargetDevice()*

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PdGetOffscreenContextPtr()**

Create a shared memory object reference to an offscreen context

**Synopsis:**

```c
void *PdGetOffscreenContextPtr(
    PdOffscreenContext_t *osc );
```

**Arguments:**

- `osc`: A pointer to a `PdOffscreenContext_t` structure that describes an offscreen context. You must have created the context with the `Pg_OSC_MEM_PAGE_ALIGN` flag in order for this function to work.

**Library:**

`ph`

**Description:**

This function creates a shared memory reference to an offscreen context and return a pointer to the object.

---

**CAUTION:** `PdGetOffscreenContextPtr()` can fail on certain hardware. You can use this function on closed systems where you know that the graphics frame buffer is linear; don’t use it in applications that target generic hardware configurations.

If `osc` is NULL, this function returns a pointer to the currently displayed screen:

- If you’re in direct mode, the pointer to the screen is read/write.
- If you aren’t in direct mode, the pointer is read only.

If `osc` isn’t the visible screen, the pointer is read/write.

---

This call blocks until the operation is complete.
PdGetOffscreenContextPtr()

Returns:
A pointer to the shared memory object, or NULL on failure.

Examples:
Draw a white vertical line in an offscreen context using software (not PgDrawRect()):

```c
// For the purposes of this example, we'll accept only
// 565, 555, or 8888 as possible targets.
typedef union vidptr {
    uint8_t * volatile ptr8;
    uint16_t * volatile ptr16;
    uint32_t * volatile ptr32;
} VidPtr_t;

PdOffscreenContext_t *buff;
VidPtr_t main_ptr,work_ptr;
uint32_t color,bytespp;
PhRect_t rect;
int i;

// Create the offscreen context
buff=PdCreateOffscreenContext(0,100,100,
Pg_OSC_MEM_PAGE_ALIGN);
if (buff==NULL)
{
    // Error code
    return;
}

// figure out which value color should be
switch (buff->format)
{
    case Pg_IMAGE_DIRECT_565 :
        color = 0x0000FFFF;
        bytespp=2;
        break;
    case Pg_IMAGE_DIRECT_555 :
        color = 0x00007FFF;
        bytespp=2;
        break;
    case Pg_IMAGE_DIRECT_8888 :
        color = 0x00FFFFFF;
        bytespp=4;
        break;
    default:
        break;
}
```
// Error code
return;
}
rect.ul.x=rect.ul.y=0;
rect.lr.x=rect.lr.y=99;
main_ptr.ptr8=(unsigned char *)
PdGetOffscreenContextPtr(buff);
if (main_ptr.ptr8 == NULL)
{
    // Error code
}

// Clear the context to black
PhDCSetCurrent(buff);
PgSetFillColor(Pg_BLACK);
PgDrawRect(&rect,Pg_DRAW_FILL);
PgFlush();

// Ensure that all drawing operations are done before
// writing using software to this context:
PgWaitHWIdle();

// draw the line in the middle:
work_ptr.ptr8=main_ptr.ptr8 + (49 * bytespp);
for (i=0; i<100; i++, work_ptr.ptr8+=buff->pitch)
{
    switch (bytespp)
    {
    case 2 :
        *work_ptr.ptr16 = color;
        break;
    case 4 :
        *work_ptr.ptr32 = color;
        break;
    }
}
PgContextBlit(buff,&rect,NULL,&rect);
PgFlush();

// You should see a black rectangle with a vertical
// white line in the middle
**PdGetOffscreenContextPtr()**

**Classification:**

*Photon*

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdCreateOffscreenContext(), PdDupOffscreenContext(), PdOffscreenContext_t, PgContextBlit(), PgContextBlitArea(), PgSwapDisplay()*

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PdGetOffscreenSurface()**

*Get an offscreen surface handle*

**Synopsis:**

```
PgSurface_t *PdGetOffscreenSurface(  
    PdOffscreenContext_t *osc );
```

**Arguments:**

- `osc` A pointer to an offscreen context structure, as returned by `PdCreateOffscreenContext()`.

**Library:**

`ph`

**Description:**

This function gets a handle for an offscreen context surface, which can be used as a color buffer for `glSetPhotonRenderSurfaceQNX()`.

**Returns:**

A pointer to an opaque `PgSurface_t`, or NULL if an error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PdCreateOffscreenContext()`, `PdOffscreenContext_t`

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PdIsOffscreenLocked()**

* Determine whether or not an offscreen context is locked

**Synopsis:**

```c
int PdIsOffscreenLocked(
    PdOffscreenContext_t *osc);
```

**Arguments:**

- `osc`  A pointer to the offscreen context, as returned by `PdCreateOffscreenContext()`, that you want to check.

**Library:**

`ph`

**Description:**

This function tests to see if the given offscreen context is currently locked.

---

This is really only useful for debugging purposes, because the state of the lock could potentially change between the time that `PdIsOffscreenLocked()` queries the status and reports it back to the application.

**Returns:**

One of:

- `Pd_OSC_LOCKED`
- `Pd_OSC_LOCK_INVALID`
- `Pd_OSC_NOT_LOCKED`

**Classification:**

Photon
**PdIsOffscreenLocked()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PdCreateOffscreenLock(), PdDestroyOffscreenLock(),
PdGetOffscreenContextPtr(), PdLockOffscreen(),
PdUnlockOffscreen()

“Offscreen locks” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PdLockOffscreen()**

*Lock an offscreen context*

**Synopsis:**

```c
int PdLockOffscreen( PdOffscreenContext_t *osc,
                     PdOSCLockParams_t *params );
```

**Arguments:**

- `osc` A pointer to the `PdOffscreenContext_t` structure for the offscreen context. You must call `PdCreateOffscreenLock()` to create an offscreen lock for this context before you call `PdLockOffscreen()`.

- `params` A pointer to a `PdOSCLockParams_t` structure (see below) that defines parameters for the lock. You can pass NULL for this argument.

**Library:**

`ph`

**Description:**

This function locks an offscreen context.

The `PdOSCLockParams_t` structure includes these members:

- `uint32_t flags`
  Flags, including:
  - `Pg_OSC_LOCK_TIMED_OUT` — blocking occurs only until the clock has gone past the value of the `time_out` member.

- `struct timespec *time_out`
  The absolute time at which to stop blocking if you’ve set `Pg_OSC_LOCK_TIMED_OUT` in the `flags` member. This behavior is like that of `sem_timedwait()`.
PdLockOffscreen()

Returns:

EOK   The context was successfully locked.
PgOSC_LOCK_TIMED_OUT
        Time out for lock occurred; the context wasn’t locked.
PgOSC_LOCK_INVALID
        The context lock is no longer valid (or never was in the first
place i.e. it wasn’t created with PdCreateOffscreenLock()).
PgOSC_LOCK_DEADLOCK
        A deadlock condition was detected.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdCreateOffscreenLock(), PdDestroyOffscreenLock(),
PdGetOffscreenContextPtr(), PdIsOffscreenLocked(),
PdUnlockOffscreen()

“Offscreen locks” in the Raw Drawing and Animation chapter of the
Photon Programmer’s Guide
**PdOffscreenContext_t**

*Data structure that describes an offscreen context*

**Synopsis:**

See below.

**Description:**

This data structure describes an offscreen context.

Don’t change the value of any of the members of this structure.

**PdOffscreenContext_t** includes:

- **dim** A **PhDim_t** structure that defines the dimensions of the offscreen context.
- **format** The type of image; see **PhImage_t**.
- **pitch** The number of bytes per scan line.

**Classification:**

Photon

**See also:**

- *PdCreateOffscreenContext*, *PdDupOffscreenContext*, *PdGetOffscreenContextPtr*, *PdSetOffscreenTranslation*, *PgContextBlit*, *PgContextBlitArea*, *PgSwapDisplay*, **PhDim_t**
  
  “Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
void PdReleaseDirectContext(
    PdDirectContext_t *DirectContext);
```

Arguments:

- `DirectContext`: The pointer to a `PdDirectContext_t` structure returned by `PdCreateDirectContext()`. This argument must not be NULL.

Library:

`ph`

Description:

This function leaves direct mode (if the application is currently in it), destroys the direct-mode context, and restores the original default context.

You must target this function at a specific card by calling `PdSetTargetDevice()`.

Examples:

See `PdCreateDirectContext()`.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 6 • Pd—Draw Context 243
PdReleaseDirectContext()

See also:

PdCreateDirectContext(), PdDirectStart(), PdDirectStop(), PdSetTargetDevice()

“Direct mode” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PdSetOffscreenTranslation(
    PdOffscreenContext_t *osc,
    PhPoint_t *trans);
```

**Arguments:**

- `osc` A pointer to the offscreen context, as returned by `PdCreateOffscreenContext()`, whose translation you want to set.
- `trans` A pointer to a `PhPoint_t` that defines the translation.

**Library:**

`ph`

**Description:**

This function sets the translation for the offscreen context pointed to by `osc` to the value specified by the structure pointed to by `trans`. This translation is applied to the points used in graphical operations in the offscreen context.

The translation stays with the offscreen context, not the application. If another application accesses the offscreen context, the same translation applies.

**Returns:**

- 0 Success.
- -1 An error occurred.

**Classification:**

Photon
**PdSetOffscreenTranslation()**

**Safety**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PdCreateOffscreenContext(), PdDupOffscreenContext(), PdGetOffscreenContextPtr(), PdOffscreenContext_t, PdSetTargetDevice(), PgContextBlit(), PgSwapDisplay(), PhDCCreate(), PhDCRelease()

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
PdSetTargetDevice()

Set the target device

Synopsis:

```c
PhRid_t PdSetTargetDevice ( const void *dc,
    PhRid_t device_rid );
```

Arguments:

- `dc`: A void pointer to any type of draw context, or NULL for the current draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- `device_rid`: The region ID of the device that you want to be the target. To get the region IDs for the currently available draw devices, call `PdGetDevices()`.

Library:

`ph`

Description:

This function sets the device with the given region ID to be the target for device-specific queries and control operations in the given draw context. By default, the target is the screen.

You can target the following functions at specific cards:

- `PdCreateDirectContext()`
- `PdCreateOffscreenContext()`
- `PdDupOffscreenContext()`
- `PdReleaseDirectContext()`
PdSetTargetDevice()

- PgGetGraphicsHWCaps()
- PgGetPalette()
- PgGetVideoMode()
- PgGetVideoModeInfo()
- PgGetVideoModeList()
- PgReadScreen()
- PgReadScreenSize()
- PgSetVideoMode()
- PgWaitDrawComplete()
- PgWaitHWIdle()

Returns:
The region ID of the previously set device, or 0 if no device was set
Success.

-1 An error occurred.

Examples:
See PdCreateDirectContext().

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
see also:

PdGetDevices()

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PdUnlockOffscreen()**

*Unlock an offscreen context*

**Synopsis:**

```c
int PdUnlockOffscreen( PdOffscreenContext_t *osc );
```

**Arguments:**

osc A pointer to the offscreen context, as returned by `PdCreateOffscreenContext()`, that you want to unlock.

**Library:**

ph

**Description:**

This function unlocks the offscreen context, `osc`. You can lock the offscreen context by calling `PdLockOffscreen()` after calling `PdCreateOffscreenLock()` to create the lock.

**Returns:**

EOK The offscreen context was successfully unlocked.

Pg_OSC_ALREADY_UNLOCKED

The context wasn’t locked to begin with (i.e. the semaphore count wasn’t increased).

Pg_OSC_LOCK_INVALID

The lock is invalid.

Pg_OSC_NOT_LOCKED

The context wasn’t locked.

**Classification:**

Photon
PdUnlockOffscreen()

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:


“Offscreen locks” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Chapter 7

Pf—Font Server
These functions provide font services to Photon applications and drivers. Using them, you can:

- calculate text extents and metrics
- generate bitmaps of character strings.

There are two separate libraries of font functions, the Photon library and the font library `font_api`. Functions that have the suffix `Cx` or `D11` are the font library versions. The font library functions use the font server plugin `phfont.so`, and let you choose how your application attaches to the font server. See the Fonts chapter of the Photon *Programmer’s Guide* for more information.
PfAllocDetailsCx()
Retrieve render buffer details

Synopsis:
#include <font_api.h>

int PfAllocDetailsCx( struct _Pf_ctrl * context,
                      char const ** name,
                      long * size );

Arguments:

context A pointer to the font context to use, returned by
PfAttachCx() or PfAttachDllCx().

name A pointer to a location where the function stores the path
and name of the shared memory, if context is not NULL.

size A pointer to a long where the function stores the size, in
bytes, of the render buffer.

Library:

font

Description:
This function retrieves the shared memory path and size of the render
buffer associated with a font context. If the render buffer was
allocated from the heap, the function sets name to NULL.

Returns:

0 Success

-1 An error occurred (errno is set).

Errors:

EFAULT Font context is NULL.
PfAllocDetailsCx()

Examples:

See the example for PfAttachDllCx().

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
</tr>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PfAttachCx(), PfAttachDllCx()

Fonts chapter of the Photon Programmer’s Guide
**PfAllocRenderCx()**

*Allocate, or reallocate, a render buffer*

**Synopsis:**

```c
#include <font_api.h>

int PfAllocRenderCx( struct _Pf_ctrl * context,
                      long size );
```

**Arguments:**

- `context`  
  A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.

- `size`  
  The size, in bytes, of the render buffer you want to allocate. A value greater than 0 allocates a shared memory buffer; less than 0 allocates a heap buffer; -1 allocates a heap buffer using the value from the `PHFONTMEM` environment variable.

**Library:**

`font`

**Description:**

This function allocates a render buffer for the provided font context, of the requested size. If the context already has an allocated render buffer, this function releases it before it allocates a new one. You must link the font context to a `fontdll_t` context using `PfAssignDllCx()`. Otherwise, an error is returned.

**Returns:**

- 0  
  Success

- -1  
  An error occurred (`errno` is set).
Errors:

ENOMEM     Insufficient resources
EACCES     shm_open() error.
EEXIST     shm_open() error.
EINTR      shm_open() error.
ELOOP      shm_open() error.
EMFILE     shm_open() error.
ENAMETOOLONG  
            shm_open() error.
ENFILE     shm_open() error.
ENOENT     shm_open() error.
ENOSPC     shm_open() error.
ENOSYS     shm_open() error.
EBADF      ftruncate() error.
EFBIG      ftruncate() error.
EINTR      ftruncate() error.
EINVAL     ftruncate() error.
EIO        ftruncate() error.
ENOSYS     ftruncate() error.
ENOTSUP    ftruncate() error.
EROFS      ftruncate() error.
EACCES     mmap() error.
EBADF      mmap() error.
PfAllocRenderCx()

EINVAL  \textit{mmap()} error.
ENODEV  \textit{mmap()} error.
ENOMEM  \textit{mmap()} error.
ENXIO   \textit{mmap()} error.
ENOTSUP Platform does not support this operation.
EFAULT  Font context is not associated with a \texttt{fontdll_t} context.

Examples:

```c
#include <font_api.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

static int check_32 = 0;
static int bad_32 = 0;
static int check_64 = 0;
static int bad_64 = 0;

static void func(void * ctx, const pf_point_t * pos, const FontRender * render)
{
    printf("NOTE : render callback.\n");

    if(check_32)
    {
        int times = render->bpl / 4;
        int total = times * 4;
        if(total != render->bpl)
            bad_32 = 1;
    }

    if(check_64)
    {
        int times = render->bpl / 8;
        int total = times * 8;
        if(total != render->bpl)
            bad_64 = 1;
    }
}

int main(int argc, char const * argv[])
{
    fontdll_t dll;

    fprintf(stderr, "POINT : PfRenderCx(dll) and bitmap alignment.\n");

    if((dll = PfAttachLocalDll(NULL, NULL)) != NULL)
    {
        struct _Pf_ctrl * pf;

        if((pf = PfAttachDllCx(dll, 0)) != NULL)
    
```
PfAllocRenderCx()

{ FontID * id;
  int skip = 0;

  if((id = PfFindFontCx(pf, *TextFont", 0L, 9)) != NULL)
  { FontName tag;
    pf_point_t pos = { 0, 0 };

    if(PfAllocRenderCx(pf, pf, PfConvertFontIDCx(pf, id, tag), 0L, 0L, "TEST.",
      0, 0, &apos, NULL, func) == -1) && (errno == EINVAL)
      { if(setenv("PHFONTMEM", "32000", 1) == -1)
      { fprintf(stderr, "NOTE : setenv failed to write to PHFONTMEM.\n");
        fprintf(stderr, "FAIL : PfAllocRenderCx(dll) and bitmap alignment.\n");
      }
        else
        { fprintf(stderr, "NOTE : render 8-bit aligned image.\n");

          if(PfAllocRenderCx(pf, -1) == 0)
          { if(PfRenderCx(pf, pf, PfConvertFontIDCx(pf, id, tag), 0L, 0L,
              "TEST.", 0, 0, &apos, NULL, func) == 0)
              { if(PfSetOptionsDll(dll, "-Z=32", NULL) == 0)
                  { fprintf(stderr, "NOTE : render 32-bit aligned image.\n");
                    check_32 = 1;

                    if(PfRenderCx(pf, pf, PfConvertFontIDCx(pf, id, tag), 0L, 0L,
                      "TEST.", 0, 0, &apos, NULL, func) == 0)
                      { if(bad_32)
                        { fprintf(stderr, "NOTE : PfRenderCx did not render \n a 32-bit aligned image.\n");
                          fprintf(stderr, "FAIL : PfRenderCx(dll) and \n bitmap alignment.\n");
                        }
                          else
                          { check_32 = 0;
                            if(PfSetOptionsDll(dll, "-Z=64", NULL) == 0)
                            { fprintf(stderr, "NOTE : render 64-bit aligned image.\n");
                              check_64 = 1;

                              if(PfRenderCx(pf, pf, PfConvertFontIDCx(pf, id, tag), 0L, 0L,
                                "TEST.", 0, 0, &apos, NULL, func) == 0)
                                { if(bad_64)
                                  { fprintf(stderr, "NOTE : PfRenderCx did \n not render a 64-bit aligned image.\n");
                                    fprintf(stderr, "FAIL : PfRenderCx(dll) \n and bitmap alignment.\n");
                                  }
                                    else
                                    { fprintf(stderr, "PASS : PfRenderCx(dll) \n and bitmap alignment.\n");
                                  }
                                }
                                else
                                { fprintf(stderr, "NOTE : PfRenderCx failed, \n errno \d.\n", errno);
                                  fprintf(stderr, "FAIL : PfRenderCx(dll) and \n bitmap alignment.\n");
                                }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
PfAllocRenderCx()

if(PfAllocRenderCx(pf, id) == -1L)
{
    fprintf(stderr, "UNRES : PfAllocRenderCx failed, errno %d.\n", errno);
    fprintf(stderr, "NOTE : PfAllocRenderCx returned success with invalid \n render buffer.\n");
}

if(PfFreeFontCx(pf, id) == -1L)
{
    fprintf(stderr, "NOTE : PfFreeFontCx returned success with invalid \n render buffer.\n");
}

PfDetachCx(pf);
PfAllocRenderCx()

{ 
    fprintf(stderr, "UNRES : Unable to attach to fontservlet, errno \%d.\n", errno);
    fprintf(stderr, "FAIL : PfRenderCx(dll) and bitmap alignment.\n");
}

if(PfDetachLocalDll(dll) == -1)
    fprintf(stderr, "NOTE : PfDetachLocalDll failed, errno \%d.\n", errno);
else
    { 
        fprintf(stderr, "UNRES : Unable to load local dll, errno \%d.\n", errno);
        fprintf(stderr, "FAIL : PfRenderCx(dll) and bitmap alignment.\n");
    }

return(0);
}

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Cancellation point</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

PfAttachCx(), PfAttachDllCx(), PfAssignDllCx()

Fonts chapter of the Photon *Programmer’s Guide*
PfAssignDllCx()
Assign a context to a local font server

Synopsis:
#include <font_api.h>

int PfAssignDllCx( struct _Pf_ctrl * context,
              fontdll_t dll );

Arguments:

dll A font server context, returned by PfAttachLocalDll().
context A pointer to the font context to use, returned by
         PfAttachCx() or PfAttachDllCx().

Library:
font

Description:
This function assigns the dll to the font context, context.

Returns:
0 Success
-1 An error occurred (errno is set).

Errors:
EFAULT The font context is NULL
ENOTSUP The platform does not support this operation.

Examples:
/* A PtHook example. Initializes a client level font server
 * instance for any application that invokes PtInit(), and has
 * knowledge of this hook.
 */
#include <font_api.h>
#include <photon/PhProto.h>
#include <stdlib.h>
#include <unistd.h>
static fontdll_t dll;
extern struct _Ph_ctrl * _Ph_;

static void cleanup_hook(void)
{
  PfAssignDllCx(_Ph_->font, NULL);
PfDetachLocalDll(dll);
dll = NULL;
}

int PtHook(void * data)
{
  if(data != NULL)
  {
    if((dll = PfAttachLocalDll(NULL, NULL)) == NULL)
      return(0);
    else
    {
      if(PfAssignDllCx(_Ph_->font, dll) == -1)
        { PfDetachLocalDll(dll);
          return(0);
        }
      else
      {
        if(access("/dev/fontsleuthctrl", F_OK) == 0)
          if(PfAttachSleuthMonitorDll(dll, -1) == -1)
            { PfAssignDllCx(_Ph_->font, NULL);
PfDetachLocalDll(dll);
              return(0);
            }
        if(atexit(cleanup_hook) != 0)
          { PfAssignDllCx(_Ph_->font, NULL);
PfDetachLocalDll(dll);
            return(0);
          }
        }
      }
    }
  }
  else
    return(0);

  return(1);
}
**PfAssignDllCx()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

*PfAttachCx(), PfAttachDllCx(), PfAttachLocalDll()*

Fonts chapter of the Photon *Programmer’s Guide*
PfAttach(), PfAttachCx()
Attach to a font server

Synopsis:

```c
#include <photon/Pf.h>
struct _Pf_ctrl *PfAttach( const char *device,  
    long size );

#include <font_api.h>
struct _Pf_ctrl* PfAttachCx( const char *device,  
    long size );
```

Arguments:

device The prefix name of the server; if this argument is NULL, the prefix is the value in the PHFONT environment variable, or /dev/phfont if this isn’t set.

size The size of an area in shared memory to set up between the task and the server for returning text bitmaps (normally required only by graphics drivers). The value determines the type of buffer the function creates:

- 0 no buffer
- <0 allocate a heap buffer
- -1 allocate a heap buffer using the value from the environment variable PHFONTMEM
- >0 allocate a shared memory buffer

Library:

- PfAttach() ph
- PfAttachCx() font

Description:

These functions attach to the font server.

PhAttach() calls PfAttach() as part of the standard Photon initialization sequence. The font library automatically invokes
PfAttach(), PfAttachCx()

PfAttach() when the library detects that the font server has been restarted. The font control structure pointer is stored in the global Photon control structure.

If you simply want to override the default font context, you can set the two environment variables (PHFONT and PHFONTMEM) to new values before performing standard Photon initialization, instead of calling this function directly.

If you want to use multiple font contexts, open them with PfAttachCx(), and then use the Cx versions of the font functions. PfAttachCx() allocates and returns a font context, and requests a font server connection. You need the font context returned by PfAttachCx() or PfAttachDllCx() to pass as the first argument to other font library Cx functions.

To detach from the font server, call the corresponding function PfDetach() or PfDetachCx().

Returns:

A pointer to an internal control structure if successful; NULL otherwise.

Examples:

PfAttachCx(): See the examples for PfConvertFontIDCx() and PfRenderCx().

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PfAttach(), PfAttachCx()

See also:
PfAttachDllCx(), PfAttachLocalDll(), PfDetach(), PfDetachCx(), PfExtentCx(), PfExtentTextCharPositionsCx(), PfGenerateFontNameCx(), PfGetOutlineCx(), PfRenderCx()

Fonts chapter of the Photon Programmer’s Guide
**PfAttachDllCx()**

**Attach to a local font server**

**Synopsis:**
```
#include <font_api.h>

struct _Pf_ctrl* PfAttachDllCx( fontdll_t dll,
                                long size);
```

**Arguments:**

- **dll** An opaque structure that describes a local font server DLL, returned by `PfAttachLocalDll()`.
- **size** The size of render buffer allocated (the area in memory used by the font server to return rendered bitmaps). The value determines the type of buffer allocated:
  - 0 — no buffer allocated
  - >0 — allocate a shared memory buffer
  - <0 — allocate a heap buffer
  - -1 — allocate a heap buffer using the value from the environment variable `PHFONTMEM`

**Library:**

- **font**

**Description:**

This routine allocates a font context structure, and connects to the local font server referred to by `dll`.

**Returns:**

- Font context Success
- NULL Failure
Errors:

ENOMEM Insufficient resources
EACCES *shm_open()* error.
EEXIST *shm_open()* error.
EINTR *shm_open()* error.
ELOOP *shm_open()* error.
EMFILE *shm_open()* error.
ENAMETOOLONG *shm_open()* error.
ENFILE *shm_open()* error.
ENOENT *shm_open()* error.
ENOSPC *shm_open()* error.
ENOSYS *shm_open()* error.
EBADF *ftruncate()* error.
EFBIG *ftruncate()* error.
EINTR *ftruncate()* error.
EINV AL *ftruncate()* error.
EIO *ftruncate()* error.
ENOSYS *ftruncate()* error.
ENOTSUP *ftruncate()* error.
EROFS *ftruncate()* error.
EACCES *mmap()* error.
EBADF *mmap()* error.
PfAttachDllCx()

EINV AL  
mmap() error.

ENODEV  
mmap() error.

ENOMEM   
mmap() error.

ENXIO    
mmap() error.

ENOTSUP  
Platform does not support this operation.

Examples:
#include <font_api.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const * argv[])
{
    fontdll_t dll;

    if((dll = PfAttachLocalDll(NULL, NULL)) != NULL)
    {
        struct _Pf_ctrl * pf;

        if((pf = PfAttachDllCx(dll, 0)) != NULL)
        {
            if(PfAllocRenderCx(pf, 32000) == 0)
            {
                char const * name;
                long size = 0L;

                if(PfAllocDetailsCx(pf, &name, &size) == 0)
                {
                    if(name != NULL)
                        printf("shmem name %s, size %ld bytes.\n", name, size);
                    else
                        printf("heap buffer size %d bytes.\n", abs(size));
                }

                PfDetachCx(pf);
            }

            if((pf = PfAttachDllCx(dll, 0)) != NULL)
            {
                if(PfAllocRenderCx(pf, -32000) == 0)
                {
                    char const * name;
                    long size = 0L;

                    if(PfAllocDetailsCx(pf, &name, &size) == 0)
                    {
                        if(name != NULL)
                            printf("shmem name %s, size %ld bytes.\n", name, size);
                        else
                            printf("heap buffer size %d bytes.\n", abs(size));
                    }

                    PfDetachCx(pf);
                }

                PfDetachLocalDll(dll);
            }
        }
    }
}
\textbf{PfAttachDllCx()}

\begin{verbatim}
return(0);
}
\end{verbatim}

\textbf{Classification:}

\begin{center}
\textbf{Photon}
\end{center}

\begin{tabular}{|l|l|}
\hline
\textbf{Safety} &  \\
\hline
Cancellation point & No \\
Interrupt handler & No \\
Signal handler & No \\
Thread & Yes \\
\hline
\end{tabular}

\textbf{See also:}

\textit{PfAttachLocalDll()}

Fonts chapter of the Photon \textit{Programmer's Guide}
PfAttachLocalDll()
Load a local font DLL

Synopsis:

```c
#include <font_api.h>

fontdll_t PfAttachLocalDll( char const * options,
                               char const * schema );
```

Arguments:

- **options** Command-line options for the local font DLL. These commands should be comma-separated. For example: `-A,-d=/usr/photon/font_repository`. This argument may be NULL if there are no options.

- **schema** The name of a schema, a configuration file used to override the default settings for a local font server. Use the `DLL_FONT_SERVER` schema for processes that need to allocate sufficient resources to act as a default font server. This schema loads a local server instance that can be used as an external server by other applications (that is, it appears in `/def/phfont`). Use NULL to load a local server instance that cannot be used by other applications.

The string referenced by `schema` may not exceed `DLL_MAX_OPTION_NAME` bytes, including the terminating NULL.

Library:

```c
font
```

Description:

This function loads a local font DLL, which eliminates message passing to an external font server. Options are processed and applied before the server instance is activated. The `options` are the same as those supported by the font server, though some options may not be relevant to a DLL instance of the font server.
Returns:

A fontdll_t context

Success

NULL An error occurred (errno is set).

Errors:

ENOENT Unable to locate font server plugin.

ELIBBAD Font server plugin is bad.

ENOMEM Insufficient resources.

Examples:

/* An example of how to run a root level font server device. */

#include <signal.h>
#include <font_api.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <sys/siginfo.h>
#include <atomic.h>
#include <sys/procmgr.h>
#include <unistd.h>

static volatile unsigned restart = 0;

void restart_fontserver(int sig)
{
    atomic_add(&restart, 1);
}

static volatile unsigned stop = 0;

void stop_fontserver(int sig)
{
    atomic_add(&stop, 1);
}

/*@ 
 * Install useful signal handlers.
 */

static int TrapSignals(void)
{
    int sig;

    #if defined(__linux__) || defined(__CYGWIN__)
        signal(SIGTERM, SIG_DFL);
        signal(SIGQUIT, SIG_DFL);
        //signal(SIGHUP, SIG_DFL);
        signal(SIGINT, SIG_DFL);
        signal(SIGHUP, SIG_DFL);
    #else
        for(sig = _SIGMIN; sig <= _SIGMAX; ++sig)
PfAttachLocalDll()

{ if(sig == SIGTERM)
    signal(sig, stop_fontserver);
else if(sig == SIGINT || sig == SIGHUP)
    signal(sig, SIG_IGN);
else if(sig == SIGUSR2)
    signal(sig, restart_fontserver);
else
    signal(sig, SIG_DFL);
}
#endif
return(0);
}

int main(int argc, char * argv[])
{ fontdll_t dll;

    /* Make oneself a daemon. */
    procmgr_daemon(EXIT_SUCCESS, PROCMDR_DAEMON_NOCHDIR | PROCMDR_DAEMON_NODEVNULL
    | PROCMDR_DAEMON_KEEPUMASK | PROCMDR_DAEMON_NOCLOSE);

    /* Process any command line arguments here, if you so desire. */
    /* Retrieve font dll context. */
    if((dll = PfAttachLocalDll(NULL, DLL_FONT_SERVER)) == NULL)
    { perror("Unable to open font DLL");
      return(EXIT_FAILURE);
    } else
    { /* Set up signals as per documentation from PfAttachServerDll. */
      TrapSignals();

      /* Initialize root level font server device instance. */
      if(PfAttachServerDll(dll, 12, NULL) == -1)
      { perror("Unable to start server thread");
        PfDetachLocalDll(dll);
        return(EXIT_FAILURE);
      } else
      { int exit = 0;

        /* Wait for font server device to exit. */
        do
        { if(PfWaitOnServerDll(dll) == -1)
          { if((errno = EINTR) && restart)
            { if(PfRestartServerDll(dll) == -1)
              { perror("Unable to restart server thread");
                exit = 1;
              } else
                atomic_sub(&restart, 1);
            } else if((errno = EINTR) && stop)
              exit = 1;
          } else
            exit = 1;
        } while(!exit);
    }
}
PfAttachLocalDll()

/* Clean everything up tidy like. */
PfDetachLocalDll(dll);
}
return(EXIT_SUCCESS);
}

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

PfAttachServerDll()

Fonts chapter of the Photon *Programmer’s Guide*
**PfAttachServerDll()**

Load a font server

**Synopsis:**
```c
#include <font_api.h>

int PfAttachServerDll( fontdll_t dll,
                       int prio,
                       char const *device );
```

**Arguments:**
- **dll** A font server context, returned by `PfAttachLocalDll()`.
- **prio** The suggested priority at which to run the font processing thread. If this value is set to -1, the function uses the current process priority.
- **device** The `/dev` device path of the font server to attach to (e.g. `/dev/bob`).

**Library:**
`font`

**Description:**
This function loads a local font server thread for the provided `dll` context. The function attempts to attach a resource manager server thread, which attaches all relevant device names. This is useful only if no external server is running, and you want this application to function as the default font server. The application must have an effective user ID of `root` in order for this approach to succeed.

Since this behaviour incorporates handling of dynamic font loading, this routine will return an error if a sleuth thread is already in service. That is, if the application has already successfully called `PfAttachSleuthMonitorDll()`, `PfAttachServerDll()` will fail, and vice versa.
This routine overwrites the local PHFONT environment variable if the device parameter is not NULL.

If the resource manager registers the default font server device (which is the same as the global PHFONT environment variable) successfully, the application should register a SIGUSR2 signal handler. In some instances of dynamic font loading, the application is signalled by SIGUSR2 to request a restart of the server thread. The application should handle the request, then at the next available time, invoke the function PfRestartServerDll(). Otherwise, the application should block the SIGUSR2 signal if you do not wish it to honor such requests.

If the server thread attaches a device name other than the default specified by the global PHFONT environment variable, then it is not notified of dynamic font changes unless directly messaged.

Returns:

0 Success
-1 An error occurred (errno is set).

Examples:

See the example in PfAttachLocalDll().

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Cancellation point</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

continued...
PfAttachServerDll()

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PfAttachLocalDll(), PfAttachSleuthMonitorDll(), PfRestartServerDll().

Fonts chapter of the Photon Programmer’s Guide
PfAttachSleuthMonitorDll()
Start a fontsleuth monitor

Synopsis:
#include <font_api.h>

int PfAttachSleuthMonitorDll (fontdll_t dll,
    int prio);

Arguments:

dll A font server context, returned by PfAttachLocalDll().
prio The suggested priority at which to run the fontsleuth monitor thread. If this value is set to -1, the function uses the current process priority. This value cannot exceed a maximum value — if it does, it is set to an internally defined maximum.

Library:

font

Description:
This function starts a thread to monitor requests from fontsleuth to load and unload font files. If there are new fonts, or fonts are deleted, fontsleuth notifies you.

Returns:
0 Success
-1 An error occurred (errno is set).

Examples:
See the example for PfAssignDllCx().
**PfAttachSleuthMonitorDll()**

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

*PfAttachLocalDll().*

Fonts chapter of the Photon *Programmer’s Guide*
**PfConvertFontID(), PfConvertFontIDCx()**

Convert a font ID to a font name for backwards compatibility

**Synopsis:**

```c
#include <photon/Pf.h>
char *PfConvertFontID( FontID *ptsID );

#include <font_api.h>
char* PfConvertFontIDCx( struct _Pf_ctrl *context,
                         FontID *ptsID,
                         char *szTag );
```

**Arguments:**

- `context` *(PfConvertFontIDCx() only)* A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `ptsID` A pointer to a `FontID` structure, returned by `PfDecomposeStemToIDCx()` or `PfDecomposeStemToID()`.
- `szTag` *(PfConvertFontIDCx() only)* A pointer to a location where the function can store the font stem.

**Library:**

- `PfConvertFontID()`
  - `ph`
- `PfConvertFontIDCx()`
  - `font`

**Description:**

These functions convert the font ID pointed to by `ptsID` into a font name. This function can be used for backwards compatibility with the older font API.

These functions don’t check to see if `ptsID` is NULL, due to the way this routine is used.
PfConvertFontID(), PfConvertFontIDCx()

Returns:

A version of the font identifier that the older font API can understand.

The return type/value could change in future releases of Photon.

Examples:

PfConvertFontID(): See PfFindFont().

PfConvertFontIDCx():

/* A FontID example. Demonstrates aspects of how to process * FontIDs and legacy font stem names. */
#include <font_api.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main(int argc, char const * argv[])
{
    struct _Pf_ctrl * pf;
    fprintf(stderr, "POINT : FontID.");
    if((pf = PfAttachCx(NULL, 0)) != NULL)
    {
        FontID * id;
        if((id = PfFindFontCx(pf, "TextFont", 0L, 9)) != NULL)
        {
            FontName tag;
            if(PfConvertFontIDCx(pf, id, tag) != NULL)
            {
                FontName tag2;
                if(PfGenerateFontNameCx(pf, "TextFont", 0L, 9, tag2) != NULL)
                {
                    if(!strcmp(tag, tag2))
                    {
                        char const * p;
                        if((p = PfFontBaseStemCx(pf, id)) != NULL)
                        {
                            fprintf(stderr, "NOTE : base stem is %s.
" , p);
                            if((p = PfFontDescriptionCx(pf, id)) != NULL)
                            {
                                uint32_t flags, size;
                                fprintf(stderr, "NOTE : descriptive foundry is %s.
" , p);
                                flags = PfFontFlagsCx(pf, id);
                                fprintf(stderr, "NOTE : flags are %x.
" , flags);
                                size = PfFontSizeCx(pf, id);
                                fprintf(stderr, "NOTE : point size is %d.
" , size);
                            }
                        }
                    }
                }
            }
        }
    }
}
PfConvertFontID(), PfConvertFontIDCx()

else
  { fprintf(stderr, "NOTE : PfFontDescriptionCx failed, errno %d.\n", errno);
    fprintf(stderr, "FAIL : FontID.\n");
  }
else
  { fprintf(stderr, "NOTE : PfFontBaseStemCx failed, errno %d.\n", errno);
    fprintf(stderr, "FAIL : FontID.\n");
  }
else
  { fprintf(stderr, "NOTE : tags are not equivalent.\n");
    fprintf(stderr, "FAIL : FontID.\n");
  }
else
  { fprintf(stderr, "NOTE : PfGenerateFontNameCx failed, errno %d.\n", errno);
    fprintf(stderr, "FAIL : FontID.\n");
  }
else
  { fprintf(stderr, "NOTE : PfConvertFontIDCx failed, errno %d.\n", errno);
    fprintf(stderr, "FAIL : FontID.\n");
  }

  if(PfFreeFontCx(pf, id) == -1L)
    { fprintf(stderr, "NOTE : PfFreeFontCx failed, errno %d.\n", errno);
    }
  else
    { fprintf(stderr, "UNRES : Unable to attach to fontserver, errno %d.\n", errno);
      fprintf(stderr, "FAIL : FontID.\n");
    }

  PfDetachCx(pf);
else
  { fprintf(stderr, "FAIL : FontID.\n");
  }

return(0);

Classification:
Photon
PfConvertFontID(), PfConvertFontIDCx()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfAttach(), PfAttachCx(), PfAttachDllCx(), PfDecomposeStemToID(),
PfDecomposeStemToIDCx(), PfFindFont(), PfFindFontCx(),
PfFontDescription(), PfFontDescriptionCx(), PfFontFlags(),
PfFontFlagsCx(), PfFontSize(), PfFontSizeCx(), PfFreeFont(),
PfFreeFontCx(), PfGenerateFontName(), PfGenerateFontNameCx()

Fonts chapter of the Photon *Programmer’s Guide*
**PfConvertPixelsToPointSizeCx()**

*Convert pixel height to point size*

**Synopsis:**

```c
#include <font_api.h>
FontID*
PfConvertPixelsToPointSizeCx( struct _Pf_ctrl * context,
                                char const * description,
                                int flags,
                                uint32_t pixel_height );
```

**Arguments:**

- **context** A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- **description** A foundry face, such as *Swis721 BT* or *PrimaSans BT*.
- **flags** Flags that identify the requested font type. The following can be ORed together:
  - PF_STYLE_BOLD
  - PF_STYLE_ITALIC
  - PF_STYLE_ANTIALIAS
  - PF_STYLE_ULINE
  - PF_STYLE_DULINE
- **pixel_height** The height in pixels that the ascender and descender of the resultant font must fit within.

**Library:**

- font

**Description:**

This function locates a point size for the provided foundry face (*description*) that has an ascender and descender that fit within the provided pixel height. If the provided foundry face represents a bitmap font, then the ID for the closest point size available that fits within the pixel height is returned. You must release the returned FontID.
**PfConvertPixelsToPointSizeCx()**

**Returns:**

- A FontID  
  Success
- NULL  
  Failure (*errno* is set).

**Errors:**

- ESRCH  
  Unable to match a point size to the provided pixel height.

**Examples:**

```c
#include <font_api.h>
#include <stdio.h>
#define PIXEL_HEIGHT 33

int main(int argc, char *argv[]) {
    struct _Pf_ctrl * pf;

    if((pf = PfAttachCx(NULL, 0)) != NULL) {
        FontName font;

        if(PfGenerateFontNameCx(pf, "PrimaSans BT", 0, 9, font) != NULL) {
            if(PfLoadMetricsCx(pf, font) == 0) {
                FontID * id;

                if((id = PfConvertPixelsToPointSizeCx(pf, "PrimaSans BT", 0,
                                                 PIXEL_HEIGHT)) != NULL) {
                    FontName font2;

                    printf("Font stem \%s fits in a height of \%d pixels.\n",
                           PfConvertFontIDCx(pf, id, font2), PIXEL_HEIGHT);
                    PfFreeFontCx(pf, id);
                }
                PfUnloadMetricsCx(pf, font);
            }
            PfDetachCx(pf);
        }
        return(0);
    }
}
```

**Classification:**

Photon
### PfConvertPixelsToPointSizeCx()

#### Safety

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### See also:

- PfAttachCx(), PfAttachDllCx()
- Fonts chapter of the Photon Programmer’s Guide
**PfDecomposeStemToID(), PfDecomposeStemToIDCx()**

Convert a stem name into a font ID

**Synopsis:**

```c
#include <photon/Pf.h>

FontID * PfDecomposeStemToID( char const * pkszStem );
```

```c
#include <font_api.h>

FontID* PfDecomposeStemToIDCx( 
    struct _Pf_ctrl * context, 
    char const * pkszStem );
```

**Arguments:**

- **context** (PfDecomposeStemToIDCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- **pkszStem** A pointer to the stem to decompose to a FontID, for example primasansbts12.

**Library:**

- PfDecomposeStemToID()  
  - ph
- PfDecomposeStemToIDCx()  
  - font

**Description:**

These functions convert a complete font stem, such as helv12b, to a FontID representation. They parse pkszStem and query the font manager or server for the information pertinent to the font.

You’re responsible for releasing the resources associated with the returned FontID structure. To do this, call the corresponding free font function, either PfFreeFont() or PfFreeFontCx().
PfDecomposeStemToID(), 
PfDecomposeStemToIDCx()

Returns:
A pointer to a FontID, or NULL on failure.

Examples:

PfDecomposeStemToIDCx(): See the example for PfGetGlyphIndexCx().

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:
PfConvertFontID(), PfConvertFontIDCx(), PfFindFont(), PfFindFontCx(), PfFontDescription(), PfFontDescriptionCx(), PfFontFlags(), PfFontFlagsCx(), PfFontSize(), PfFontSizeCx(), PfFreeFont(), PfFreeFontCx()

Fonts chapter of the Photon Programmer’s Guide
**PfDefaultContext()**

*Return the default font context*

**Synopsis:**

```c
#include <photon/Pf.h>
_Pf_ctrl PfDefaultContext ( void );
```

**Library:**

`ph`

**Description:**

This function returns the default font context.

**Returns:**

A _Pf_ctrl structure.

**Examples:**

See the example for `PfGetGlyphIndexCx()`.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

`PfAttachDllCx(), PfAttachLocalDll(), PfDetachCx()`
PfDetach(), PfDetachCx()

Detach from font server

Synopsis:

```c
#include <photon/Pf.h>
void PfDetach( struct _Pf_ctrl *pf );
```

```c
#include <font_api.h>
void PfDetachCx( struct _Pf_ctrl *pf );
```

Arguments:

`pf`  A control structure. For `PfDetach()`, this structure was returned by a previous call to `PfAttach()`. For `PfDetachCx()`, this structure was returned by a previous call to `PfAttachCx()` or `PfAttachDllCx()`.

Library:

- `PfDetach()`  ph
- `PfDetachCx()`  font

Description:

These functions detach the task from the font server and release all the memory that the font context uses, including any shared memory and local metrics code. If your application has attached to a local font DLL (using `PfAttachLocalDll()`) , it must also call `PfDetachLocalDll()`.

Examples:

- `PfDetachCx()`: See the examples for `PfConvertFontIDCx()` and `PfRenderCx()`.

Classification:

- Photon
**PfDetach(), PfDetachCx()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

PfAttach(), PfAttachCx(), PfAttachDllCx(), PfExtentCx(), PfExtentTextCharPositionsCx(), PfGetOutlineCx(), PfRenderCx()

Fonts chapter of the Photon *Programmer’s Guide*
PfDetachLocalDll()
Unload a local server

Synopsis:

```
#include <font_api.h>
int PfDetachLocalDll( fontdll_t dll );
```

Arguments:

dll The local server context, returned by PfAttachLocalDll()

Library:

font

Description:

This function unloads a local font server dll. All resources associated with the provided fontdll_t context are deallocated. If a server thread is active, it will also terminate. If a fontsleuth thread is active, it will also terminate.

Returns:

0 Success
-1 An error occurred (errno is set).

Errors:

EFAULT Font dll context is NULL.

Examples:

See the examples for PfAttachLocalDll() and PfAssignDllCx().

Classification:

Photon
**PfDetachLocalDll()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
</tr>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

*PfAttachLocalDll().*

Fonts chapter of the Photon *Programmer’s Guide*
**PfDynamicFontIDCx()**

*Retrieve a dynamic font ID*

**Synopsis:**

```c
#include <font_api.h>
long PfDynamicFontIDCx( struct _Pf_ctrl * context,
                        char const * path );
```

**Arguments:**

- `context` A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `path` The full path and font name for the font file from which to locate the dynamic ID.

**Library:**

`font`

**Description:**

This function retrieves the assigned dynamic load ID for the given font file, if it has been loaded by `PfDynamicLoadCx()`.

**Returns:**

A dynamic font ID

- `Success`
- `-1` Failure

**Errors:**

- `EFAULT` Font context or path is NULL.
- `ESRCH` Unable to locate device.
- `EBADF` Connection has gone stale, or device error occurred.
- `ENETUNREACH` Bad message buffer.
**PfDynamicFontIDCx()**

ELIBACC  Unable to locate render plugin for specified font.
ENOTSUP  Provided font file is not supported by any render plugin.
EPERM    Permission denied to process request.

**Examples:**

See the example for *PfDynamicLoadCx().*

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

*PfAttachCx(), PfAttachDllCx(), PfDynamicLoadCx().*

Fonts chapter of the Photon *Programmer’s Guide*
**PfDynamicLoad(), PfDynamicLoadCx()**

*Dynamically load a font*

**Synopsis:**

```c
#include <photon/Pf.h>
long PfDynamicLoad( char const *pkcFontFile,
                    char *pszDescription );

#include <photon/Pf.h>
long PfDynamicLoadCx( struct _Pf_ctrl *context,
                      char const *pkcFontFile,
                      FontDescription pszDescription );
```

**Arguments:**

- `context` *(PfDynamicLoadCx() only)* A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `pkcFontFile` The full pathname of the font file that you want to load.
- `pszDescription` A pointer to an array of `MAX_DESC_LENGTH + 1` bytes, where the function can store the description of the font, or `NULL` if you don’t want the description.

**Library:**

- `PfDynamicLoad()`
  - `ph`
- `PfDynamicLoadCx()`
  - `font`

**Description:**

These functions request that the font manager dynamically install a font file. The font can be stored in any location. For example, a web server might need to dynamically load fonts embedded in HTML stored in its own temporary font directory.
PfDynamicLoad(), PfDynamicLoadCx()

If pkcFontFile is NULL, these functions set errno to EFAULT and return -1L.

If these functions successfully load the font, they fill the pszDescription array with the descriptive name of the loaded font, e.g. Comic Sans MS. If a scalable “collection” is encountered, only the first description found is returned. You can use this description to verify that the font contains what you think it does. If pszDescription is NULL, the parameter is ignored.

Returns:

A nonnegative dynamic font ID that can be used to refer to the file when calling PfDynamicUnload(), or -1L if an error occurred (errno is set).

Errors:

EBUSY There are too many dynamic fonts already loaded.
EBADF Trouble opening or closing file descriptors.
EFAULT The pkcFontFile argument is NULL.
ENOTSUP An attempt was made to load an unsupported font type.
ENOMEM Not enough memory available to proceed with the load.
EINVAL An invalid condition was encountered, possibly due to an invalid font file.
EEXIST The file that was attempted to be loaded was either already dynamically loaded, or statically installed.

Examples:

PfDynamicLoadCx():

```
#include <font_api.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <limits.h>
```
int main(int argc, char const * argv[])  
{
    fontdll_t dll;
    fprintf(stderr, "POINT : PfDynamicLoadCx.\n");
    struct _Pf_ctrl * pf;
    if((dll = PfAttachLocalDll(NULL, NULL)) != NULL)
    {
        struct _Pf_ctrl * pf;
        if((pf = PfAttachDllCx(dll, 0)) != NULL)
        {
            FontDescription desc;
            long id, id2;
            char fullpath[_POSIX_PATH_MAX];
            /* Full path to font file. */
            snprintf(fullpath, sizeof(fullpath), "%s", argv[1]);
            if((id = PfDynamicLoadCx(pf, fullpath, desc)) != -1L)
            {
                if((id2 = PfDynamicFontIDCx(pf, fullpath)) != -1L)
                {
                    fprintf(stderr, "Comparing id %ld to %ld.\n", id, id2);
                    if(id != id2)
                    {
                        fprintf(stderr,
"NOTE : Retrieved id is not the same as loaded id, \nfile : %s.\n", fullpath);
                        fprintf(stderr, "FAIL : PfDynamicLoadCx.\n");
                        exit(EXIT_FAILURE);
                    }
                }
            }
        }
        else
        {
            fprintf(stderr, "NOTE : PfDynamicLoadCx failed, errno %d.\n", errno);
            fprintf(stderr, "FAIL : PfDynamicLoadCx.\n");
            exit(EXIT_FAILURE);
        }
    }
    else
    {
        fprintf(stderr, "UNRES : Unable to attach to fontserver, errno %d.\n", errno);
        fprintf(stderr, "FAIL : PfDynamicLoadCx.\n");
        exit(EXIT_FAILURE);
    }
    PfDetachCx(pf);
    fprintf(stderr, "PASS : PfDynamicLoadCx.\n");
}
else
{
    fprintf(stderr, "UNRES : Unable to load dll font instance, errno %d.\n", errno);
    fprintf(stderr, "FAIL : PfDynamicLoadCx.\n");
}
PfDynamicLoad(), PfDynamicLoadCx()

return(0);
}

PfDynamicLoad():

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <Ph.h>
#include <Pt.h>

int fnLoad(PtWidget_t * pwgt, void * pvData,
PtCallbackInfo_t * ptsInfo);
int fnUnload(PtWidget_t * pwgt, void * pvData,
PtCallbackInfo_t * ptsInfo);
int fnChangeDisplay(void);

typedef unsigned int BOOL;
#define FALSE 0
#define TRUE !FALSE

PtWidget_t * pwndMain = NULL, * pbtnLoad = NULL,
* pbtnUnload = NULL, * ptxtDisplay = NULL;
long lID = 0L;
BOOL bLoaded = FALSE;
char * pcFace = NULL;

int main (int argc, char *argv[])
{
    PtArg_t args[4];
    PhPoint_t win_size, pntPOS, pntDIM;
    short nArgs = 0;
    char szTextFont[MAX_FONT_TAG];

    PtInit (NULL);

    // set base pwndMain parms
    win_size.x = 450;
    win_size.y = 450;

    PtSetArg(&args[0],Pt_ARG_DIM, &win_size, 0);
    // window title = name of program
    PtSetArg(&args[1],Pt_ARG_WINDOW_TITLE,
        (long)"Load On The Fly", 0);
PfDynamicLoad(), PfDynamicLoadCx()

pwndMain = PtCreateWidget (PtWindow, Pt_NO_PARENT, 2, args);

nArgs = 0;
pntDIM.x = 100;
pntDIM.y = 20;
PtSetArg(&args[0], Pt_ARG_DIM, &pntDIM, 0L);
nArgs++;
pntPOS.x = 100;
pntPOS.y = 10;
PtSetArg(&args[1], Pt_ARG_POS, &pntPOS, 0L);
nArgs++;
PtSetArg(&args[2], Pt_ARG_TEXT_STRING, (long)"LOAD", 0L);
nArgs++;
PtSetArg(&args[3], Pt_ARG_TEXT_FONT, (long)PfGenerateFontName("TextFont", 0, 9, szTextFont), 0L);
nArgs++;
pbtnLoad = PtCreateWidget(PtButton, pwndMain, nArgs, args);
PtAddCallback(pbtnLoad, Pt_CB_ACTIVATE, fnLoad, argv[1]);
PtRealizeWidget(pbtnLoad);

nArgs = 0;
pntDIM.x = 100;
pntDIM.y = 20;
PtSetArg(&args[0], Pt_ARG_DIM, &pntDIM, 0L);
nArgs++;
pntPOS.x = 100;
pntPOS.y = 50;
PtSetArg(&args[1], Pt_ARG_POS, &pntPOS, 0L);
nArgs++;
PtSetArg(&args[2], Pt_ARG_TEXT_STRING, (long)"UNLOAD", 0L);
nArgs++;
PtSetArg(&args[3], Pt_ARG_TEXT_FONT, (long)szTextFont, 0L);
nArgs++;
pbtnUnload = PtCreateWidget(PtButton, pwndMain, nArgs, args);
PtAddCallback(pbtnUnload, Pt_CB_ACTIVATE, fnUnload, NULL);
PtRealizeWidget(pbtnUnload);

nArgs = 0;
pntDIM.x = 100;
pntDIM.y = 20;
PtSetArg(&args[0], Pt_ARG_DIM, &pntDIM, 0L);
nArgs++;
pntPOS.x = 100;
pntPOS.y = 90;
PtSetArg(&args[1], Pt_ARG_POS, &pntPOS, 0L);
nArgs++;
PfDynamicLoad(), PfDynamicLoadCx()

PtSetArg(&args[2], Pt_ARG_TEXT_STRING, (long)"Hello", 0L);
nArgs++;
PtSetArg(&args[3], Pt_ARG_TEXT_FONT, (long)szTextFont, 0L);
nArgs++;
ptxtDisplay = PtCreateWidget(PtText, pwndMain, nArgs, args);

(void) PtRealizeWidget(pwndMain);

pcFace = argv[2];
PtMainLoop();

return(EXIT_SUCCESS);
}

int fnLoad(PtWidget_t * pwgt, void * pvData,
PtCallbackInfo_t * ptsInfo)
{   
if((lID = PfDynamicLoad((char const *)pvData, NULL)) == -1)
{    perror("fnLoad: ");
    return(Pt_CONTINUE);
}  
else
{    bLoaded = TRUE;
    fnChangeDisplay();
    printf("Return code is %ld\n", lID);
}

return(Pt_CONTINUE);
}

int fnUnload(PtWidget_t * pwgt, void * pvData,
PtCallbackInfo_t * ptsInfo)
{   
if(bLoaded)
    if(PfDynamicUnload(lID) == -1)
        perror("fnUnload: ");
    else
    {    bLoaded = FALSE;
        fnChangeDisplay();
    }

return(Pt_CONTINUE);
}

int fnChangeDisplay(void)
{   PtArg_t tsArg;
    char szTextFont12[MAX_FONT_TAG];
    ...
PfDynamicLoad(), PfDynamicLoadCx()

```c
pcFace = PtFontSelection(pwndMain, NULL, "Select Font",
    PfGenerateFontName("TextFont", 0, 12,
    szTextFont12),
    -1L, PHFONT_ALL_FONTS, "AaBb");

PtSetArg(&tsArg, Pt_ARG_TEXT_FONT, (long)pcFace, 0L);
PtSetResources(ptxtDisplay, 1, &tsArg);

return(Pt_CONTINUE);
}
```

**Classification:**
Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PfDynamicUnload(), PfDynamicUnloadCx()

Fonts chapter of the Photon *Programmer's Guide*
**PfDynamicUnload(), PfDynamicUnloadCx()**

*Unload a dynamically loaded font*

**Synopsis:**

```c
#include <photon/Pf.h>
long PfDynamicUnload( long lDynamicFontID );

#include <font_api.h>
long PfDynamicUnloadCx( struct _Pf_ctrl * context, 
                        long lDynamicFontID );
```

**Arguments:**

- **context**  
  (PfDynamicUnloadCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().

- **lDynamicFontID**  
  The font ID, returned by PfDynamicLoad(), of the font that you want to unload.

**Library:**

- **PfDynamicUnload()**  
  
  - **ph**

- **PfDynamicUnloadCx()**  
  
  - **font**

**Description:**

This function unloads a dynamically loaded font.

**Returns:**

0L on success, or -1L if an error occurred (errno is set).

**Errors:**

- **PfDynamicUnload():**

  ESRCH The function couldn’t locate the given dynamic font ID or font file entry.
**PfDynamicUnload(), PfDynamicUnloadCx()**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>An error occurred when attempting to close and remove the font file from the affected library.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Not enough memory was available to proceed with the unload.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>An invalid condition was encountered, possibly due to an invalid font file.</td>
</tr>
</tbody>
</table>

**PfDynamicUnloadCx():**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERANGE</td>
<td>Provided ID is less than zero.</td>
</tr>
<tr>
<td>EBADF</td>
<td>Connection has gone stale, or a device error occurred.</td>
</tr>
<tr>
<td>ENETUNREACH</td>
<td>Bad message buffer.</td>
</tr>
<tr>
<td>ELIBACC</td>
<td>Unable to locate render plugin for specified font.</td>
</tr>
<tr>
<td>ESRCH</td>
<td>Unable to locate render plugin type for specified id.</td>
</tr>
</tbody>
</table>

**Examples:**

*PfDynamicUnload():* See *PfDynamicLoad().*

*PfDynamicUnloadCx():* See the example for *PfDynamicLoadCx().*

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PfDynamicUnload(), PfDynamicUnloadCx()

See also:
PfDynamicLoad(), PfDynamicLoadCx()
Fonts chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
#include <photon/Pf.h>
int PfExtent( PhRect_t *extent,
PhPoint_t const *pos,
const char *font,
long adata,
long bdata,
const char *str,
int len,
int flags,
PhRect_t const *clip );
```

```c
#include <font_api.h>
int PfExtentCx( struct _Pf_ctrl *context,
PhRect_t *extent,
PhPoint_t const *pos,
const char *font,
long adata,
long bdata,
const char *str,
int len,
int flags,
PhRect_t const *clip );
```

**Arguments:**

- **context** *(PfExtentCx() only)* A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- **extent** A pointer to a PhRect_t structure where the function stores the calculated extent. The members are:
  
  - **ul.x** the left bearing.
  - **lr.x** the maximum x distance.
  - **ul.y** the ascender.
  - **lr.y** the descender.

The baseline of the font is at position y=0; the width of the string is:
**PfExtent(), PfExtentCx()**

\[
lr.x - \min(ul.x, 0) + 1
\]

The height of the string is:
\[
lr.y - ul.y + 1
\]

*pos*  
A pointer to a `PhPoint_t` structure that specifies an offset to apply to the extent. If `pos` is NULL, no offset is applied.

*font*  
The base font, which you should create by calling `PfGenerateFontName()`.

*adata*  
The horizontal fractional point size, if you set `PF_FRACTIONAL` in the `flags` argument.

*bdata*  
The vertical fractional point size, if you set `PF_FRACTIONAL` in the `flags` argument.

*str*  
The string whose extent you want to calculate. The string is a UTF-8 multibyte one by default.

*len*  
The length of the string, `str`, in bytes. If `len` is 0, the function uses `strlen(str)`.

*flags*  
Flags that affect the behavior of the function. You can set up to one of the following to indicate the format of the string:
- `PF_WIDE_CHARS` — the string is composed of 16-bit wide characters. If you set this flag, the function assumes that each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.
- `PF_WIDE_CHAR32` — the string is composed of 32-bit wide characters. If you set this flag, the function assumes that each character is represented by 4 bytes that conform to the ISO/IEC 10646-1 UCS-4 four-byte format.
Although this flag allows for 32-bit wide characters, the underlying font system currently supports only characters up to Unicode U+FFFE.

If you don’t set either of the above, the function assumes that the string is composed of UTF-8 multibyte characters.

You can OR in any of these flags:

- **PF_FRACTIONAL** — use a fractional 16.16 point size. If you set this flag, use the `adata` argument to specify the horizontal fractional point size, and `bdata` to specify the vertical fractional point size.

- **PF_RECT** — make the function behave like `PfExtentTextToRect()`. If you set this flag, use the `clip` argument to specify the rectangle to extent within. If successful, the function returns the number of characters that fit within the rectangle.

`clip` A pointer to a `PhRect_t` structure that’s a suggested clipping rectangle for the font manager to abide by.

**Library:**

- `PfExtent()` **ph**
- `PfExtentCx()` **font**

**Description:**

These functions calculate the extent rectangle of a text string. The base font determines the ascender and descender values of the extent. The width depends on the string; the actual font used by characters within the string may differ from this base font (as specified in the fontext and fontmap files).

The difference between `PfExtent()` and `PfExtentCx()` is that `PfExtentCx()` lets you specify the font context to use.
If metrics for the base font have been loaded locally (see \texttt{PfLoadMetrics()}) , the extent is calculated internally; otherwise, a request is sent to the font server.

The generic design of these routines allows for future expansion.

\textbf{Returns:}

If you set \texttt{PF\_RECT} in the \texttt{flags} argument, \texttt{PfExtent()} and \texttt{PfExtentCx()} return the number of characters that fit within the rectangle specified by \texttt{clip}, or -1 if an error occurred.

If you don’t set \texttt{PF\_RECT} in the \texttt{flags} argument, these functions return 0 on success, or -1 if an error occurred.

\textbf{Examples:}

\texttt{PfExtentCx():}

```c
/* This example demonstrates a straightforward method of
 * using fractional processing to fit text within a given
 * canvas size, without using floating point. Fractional
 * processing is performed using a 16.16 integer format, for
 * example, 1 point size is represented as 1 \textless{}\textless{} 16. Therefore,
 * 0.1 of a point size would be (1 \textless{}\textless{} 16) / 10. This example
 * could be further enhanced by using 0.1 of a point size to
 * further fine tune the fitting of text within the canvas size.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>
/* Define the image modules. */
#define PX\_IMAGE\_MODULES
#define PX\_BMP\_SUPPORT
#include <photon/PxImage.h>
#define NUM\_LINES 4
char const * text[] = { "Hello Bob",
   "How are you doing?",
   "I hope you have a good day.",
   "Bye for now."};
void raw\_draw(Pt\_Widget_t * widget, void * data, Ph\_Tile_t * damage);
void draw\_cb(void * ctx, const Pt\_point_t * pnt, const Font\_Render * render);
```

312 Chapter 7 • Pf—Font Server
```c
int main(int argc, char *argv[]) {
    if (PtInit(NULL) == 0) {
        struct _Pf_ctrl * pf;
        if ((pf = PfAttachCx("/dev/phfont32", 320000)) != NULL) {
            FontName font;
            if (PfGenerateFontNameCx(pf, "Swis721 BT", 0, 14, font) != NULL) {
                short nargs = 0;
                PhDim_t win_size;
                PtArg_t tsArg[7];
                PtWidget_t * wnd;
                win_size.w = 450;
                win_size.h = 450;

                nargs = 0;
                PtSetArg(&tsArg[nargs++], Pt_ARG_DIM, &win_size, 0L);
                PtSetArg(&tsArg[nargs++], Pt_ARG_WINDOW_TITLE, "fractional", 0L);
                if ((wnd = PtCreateWidget(PtWindow, NULL, nargs, tsArg)) != NULL) {
                    PhRect_t canvas;
                    PhDim_t dim;
                    long flags;
                    PtWidget_t * raw;
                    PtExtentWidget(wnd);
                    PtCalcCanvas(wnd, &canvas);

                    nargs = 0;
                    dim.w = canvas.lr.x - canvas.ul.x;
                    dim.h = canvas.lr.y - canvas.ul.y;
                    PtSetArg(&tsArg[nargs++], Pt_ARG_DIM, &dim, 0L);
                    PtSetArg(&tsArg[nargs++], Pt_ARG_RAW_DRAW_F, raw_draw, 0L);
                    PtSetArg(&tsArg[nargs++], Pt_ARG_FILL_COLOR, Pg_WHITE, 0L);
                    flags = Pt_RIGHT_ANCHORED_RIGHT | Pt_LEFT_ANCHORED_LEFT
                        | Pt_TOP_ANCHORED_TOP | Pt_BOTTOM_ANCHORED_BOTTOM;
                    PtSetArg(&tsArg[nargs++], Pt_ARG_ANCHOR_FLAGS, flags, flags);
                    PtSetArg(&tsArg[nargs++], Pt_ARG_POINTER, pf, 0L);
                    PtSetArg(&tsArg[nargs++], Pt_ARG_USER_DATA, &font, sizeof(font));
                    if ((raw = PtCreateWidget(PtRaw, wnd, nargs, tsArg)) != NULL) {
                        PtRealizeWidget(wnd);
                        PtMainLoop();
                    }
                }
            }
        }
    }
    return(0);
}

void raw_draw(PtWidget_t * widget, void * data, PhTile_t * damage) {
    { pf_point_t pos = {0, 0};
        struct _Pf_ctrl * pf;
        char const * font;
        PgColor_t old1, old2;
        PhRect_t clip;
        ...}
```
PfExtent(), PfExtentCx()

```
PhDim_t dim;
int i = 0;
PtGetResource(widget, Pt_ARG_POINTER, &pf, 0L);
PtGetResource(widget, Pt_ARG_USER_DATA, &font, 0L);
PtSuperClassDraw(PtBasic, widget, damage);
PtCalcCanvas(widget, &clip);
PtClipAdd(widget, &clip);
dim.w = clip.lr.x - clip.ul.x + 1;
dim.h = clip.lr.y - clip.ul.y + 1;
PgSetFont(font);
old1 = PgSetTextColor(Pg_BLACK);
old2 = PgSetFillColor(Pg_WHITE);
do
{
    pf_rect_t extent;
    int xsize;
    int ysize;
    int xtoggle;
    int ytoggle;
    xsize = 12 << 16;
    ysize = 12 << 16;
    pos.x = clip.ul.x;
    xtoggle = 0;
    ytoggle = 0;
    do
    {
        if(PfExtentCx(pf, &extent, NULL, font, xsize, ysize, text[i],
            strlen(text[i]), PF_FRACTIONAL, NULL) == 0)
        {
            ok_w = 0, ok_h = 0, diff;
            if((extent.lr.x - extent.ul.x + 1) <= dim.w)
            {
                diff = (dim.w - (extent.lr.x - extent.ul.x + 1)) << 16;
                if(diff > 65536)
                {
                    xsize += 65536;
                    xtoggle++;
                    if((xtoggle >= 2) && (diff < (65536 * 2)))
                        ok_w = 1;
                }
                else
                {
                    ok_w = 1;
                }
            }
            else
            {
                xsize -= 65536;
                xtoggle++;
                if((xtoggle >= 2) && (diff < (65536 * 2)))
                    ok_w = 1;
            }
        }
        else
        {
            if((extent.lr.y - extent.ul.y + 1) <= (dim.h / NUM_LINES))
            { break; }
        }
    }
    i++;
}
```

314 Chapter 7 • Pf—Font Server
PfExtent(), PfExtentCx()

```c
{ 
  diff = ((dim.h / NUM_LINES) -
              (extent.lr.y - extent.ul.y + 1)) << 16;
  if(diff > 65536)
  { 
    ysize += 65536;
    ytoggle++;
    if((ytoggle >= 2) && (diff < (65536 * 2)))
      ok_h = 1;
    else
      ok_h = 1;
  } else
  { 
    ysize -= 65536;
    ytoggle++;
    if((ytoggle >= 2) && (diff < (65536 * 2)))
      ok_h = 1;
  }
  if(ok_w && ok_h)
  { 
    if(PfRenderCx(pf, &extent, font, xsize, ysize, text[i],
                  strlen(text[i]), PF_FRACTIONAL, &pos, NULL, draw_cb) == -1)
      perror("\n");
    pos.y += extent.lr.y - extent.ul.y + 1;
    break;
  }
  while(1);
  i++;
} while(i < NUM_LINES);
PgSetTextColor(old1);
PgSetFillColor(old2);
PtClipRemove();
return;
}

void draw_cb(void * ctx, const pf_point_t * pnt, const FontRender * render)
{ 
  PhImage_t tsImage;
  PgColor_t palette[2] = { Pg_WHITE, Pg_BLACK };
  memset(&tsImage, 0x00, sizeof(PhImage_t));
  tsImage.size.w = render->size.x;
  tsImage.size.h = render->size.y;
  tsImage.bmp = render->bmptr;
  tsImage.palette = palette;
  tsImage.colors = 2;
  if(render->bpp == 1)  
```

Chapter 7 • Pf—Font Server 315
PfExtent(), PfExtentCx()

```c
{ tsImage.palette = NULL;
  tsImage.type = Pg_BITMAP_BACKFILL;
} else if(render->bpp == 4)
{ 
  palette[0] = Pg_WHITE;
  palette[1] = Pg_BLACK;
  tsImage.type = Pg_IMAGE_GRADIENT_NIBBLE;
} else if(render->bpp == 8)
{ 
  if(render->flags & FONTRENDER_RGB_PIXMAP)
  { 
    tsImage.palette = NULL;
    tsImage.type = Pg_IMAGE_DIRECT_R8G8B8;
  }
  else
  { 
    palette[0] = Pg_WHITE;
    palette[1] = Pg_BLACK;
    tsImage.type = Pg_IMAGE_GRADIENT_BYTE;
  }
}

if(PgDrawPhImage(pnt, &tsImage, 0x00) == -1)
{ 
  printf("Ouch!\n");
}

PgFFlush(1);
return;
```

**Classification:**

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No

**See also:**

PfAttach(), PfAttachCx(), PfDetach(), PfDetachCx(), PfExtentComponents(), PfExtentComponentsCx(), PfExtentFractTextCharPositions(), PfExtentText(), PfExtentTextCharPositions(), PfExtentTextCharPositionsCx(), PfExtentTextToRect(), PfExtentWideText(), PfFractionalExtentText()
PfExtent(), PfExtentCx()

PfGenerateFontName(), PfGenerateFontNameCx(), PfLoadMetrics(), PfLoadMetricsCx(), PhPoint_t, PhRect_t
**PfExtentComponents(), PfExtentComponentsCx()**

Calculate the extent of a text string and invoke a callback

**Synopsis:**
```
#include <photon/Pf.h>
PhRect_t *PfExtentComponents(
    PhRect_t *extent,
    PhPoint_t const *pos,
    const char *font,
    const char *str,
    int len,
    void (*func)(PhRect_t *,
                 const char *,
                 const char *, int) );
```
```
#include <font_api.h>
pf_rect_t* PfExtentComponentsCx(
    struct _Pf_ctrl *context,
    pf_rect_t *extent,
    pf_point_t const *pos,
    const char *font,
    const char *string,
    int len,
    void(*func)
        (pf_rect_t const *,
         const char *,
         const char *,
         int) );
```

**Arguments:**
- **context** *(PfExtentComponentsCx() only)* A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- **extent** A pointer to a PhRect_t structure where the function stores the string’s extent. For the interpretation of the members of this structure, see PfExtentText().
- **pos** NULL, or a pointer to a PhPoint_t structure that defines an offset that you want to apply to the extent.
**PfExtentComponents(), PfExtentComponentsCx()**

- **font**  
The base font to use when calculating the extent. You should create this argument by calling `PfGenerateFontName()` or `PfGenerateFontNameCx()`.

- **str**  
The UTF-8 multibyte string whose extent you want to determine.

- **len**  
The number of bytes in the string. If `len` is 0, the function assumes that it's `strlen(str)`.

- **func**  
A function that you want to call for each component of the string. The callback function is passed an extent rectangle, the filename of the font required, and the string and length of the character run.

**Library:**

- `PfExtentComponents()`  
  - `ph`

- `PfExtentComponentsCx()`  
  - `font`

**Description:**

These functions calculate the extent of a text string as per `PfExtentText()`, and also invoke a user callback function `func` for each component of the string (a run of characters sourced from a single font).

This facility is used by `phrelay` (see the QNX Neutrino *Utilities Reference*) to determine which font files have to be downloaded to the remote system in order to correctly render a string.
**PfExtentComponents(), PfExtentComponentsCx()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PfExtent(), PfExtentCx(), PfExtentText(), PfExtentTextToRect(), PfGenerateFontName(), PfGenerateFontNameCx(), PhPoint_t, PhRect_t

Fonts chapter of the Photon *Programmer’s Guide*
Synopsis:

```
#include <photon/Pf.h>

int PfExtentFractTextCharPositions(
    PhRect_t * ptsExtent,
    PhPoint_t * ptsPos,
    char * psz,
    const char * pckFont,
    int32_t * piIndices,
    int32_t * piPenPositions,
    int32_t iArrayLen,
    uint32_t ulFlags,
    int32_t iBytes,
    uint32_t uiExtentLen,
    PhRect_t const * pktsClip,
    uint32_t uiXscale,
    uint32_t uiYscale );
```

Arguments:

- `ptsExtent` A pointer to a `PhRect_t` structure that’s used to store the extent of the string.
- `ptsPos` A pointer to a `PhPoint_t` structure that’s used as an offset to apply against the extent. If NULL, no offset is added to the extent values.
- `psz` A pointer to a NUL-terminated character string.
- `pckFont` A pointer to a NUL-terminated constant character string, containing the stem name of the particular font. You should use `PfGenerateFontName()` to create this.
- `iArrayLen` The number of integer entries in the `piIndices` and `piPenPositions` arrays.
- `piIndices` A pointer to an integer array of length `iArrayLen`. An index corresponds to a location within the string pointed to by `psz`. 
PfExtentFractTextCharPositions()

For example, index 0 relates to the pen’s x position at the start of the string, index 1 corresponds to the pen’s x position after character 1, index 2 corresponds to the pen’s x position after character 2, and so on.

The indexes must be in numerical order, in order to function as expected.

piPositions A pointer to an integer array of length iArrayLen. This array contains the resulting pen x values (in pixels), for each index.

ulFlags A 32-bit value used for flags. Values that can be ORed in are:

- PF_WIDE_CHARS — the characters pointed to by psz are an array of wchar_t characters. By default, the function assumes the characters are multibyte.
- PF_CHAR_DRAWPOSITIONS — if turned on, the bearing x value of the next symbols aren’t applied to the returned pen x positions. This is useful when placing cursors:

Lucy.

If this bit isn’t set, the bearing x value of the next symbols are applied to the pen x positions. This is useful when drawing symbols individually, where you need to know where to place the x origin of each symbol:
**PfExtentFractTextCharPositions()**

*iBytes*  The number of bytes in the string. If this is 0, the function assumes that the number of bytes is:

\[ \text{strlen( psz ) / wstrlen( psz )} \]

*uiExtentLen*  The number of characters from the beginning of the string to include in the extent. If 0, the entire string is extented, as permitted by the clipping rectangle.

*pktsClip*  A clipping rectangle to be used to reduce processing, depending on the value of *pktsClip-*lr.x (in pixels). If *pktsClip* is NULL, no clipping is applied.

*uiXscale, uiYscale*  Horizontal and vertical scaling factors in 16.16 format.

**Library:**

ph

**Description:**

*PfExtentFractTextCharPositions()* lets you obtain the pen’s x position after every index specified in the function call. It’s similar to *PfExtentTextCharPositions()* , except that fractional scaling is applied.

**Returns:**

0  Success.

-1  An error occurred; *errno* is set.
**PfExtentFractTextCharPositions()**

**Errors:**

ERANGE  The font manager couldn’t fulfill the request; one of the following is true:
- The *iArrayLen* argument is larger than **strlen**(psz).
- If index 0 is requested, then *iArrayLen* is larger than **strlen**(psz) + 1.
- The *iArrayLen* argument is less than or equal to 0.
- An index in *piIndices* references a character greater than **strlen**(psz).

EFAULT  One of *ptsExtent*, *piIndices*, *piPenPositions*, *pckFont*, or *psz* is NULL.

EINVAL  The font is fixed-width, and an error occurred when trying to retrieve the common width of all characters in that particular font.

EMORE  Something unexpected occurred while processing a run of characters.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfExtent(), PfExtentCx(), PfExtentText(), PfExtentTextCharPositions(), PfExtentTextCharPositionsCx(),*
PfExtentFractTextCharPositions()

PfExtentTextToRect(), PfFractionalExtentText(),
PfGenerateFontName(), PhPoint_t, PhRect_t

Fonts chapter of the Photon Programmer’s Guide
**PfExtentText()**

*Calculate the extent rectangle of a text string*

**Synopsis:**

```c
#include <photon/Pf.h>

PhRect_t *PfExtentText( PhRect_t *extent,
    PhPoint_t const *pos,
    const char *font,
    const char *str,
    int len);
```

**Arguments:**

- `extent`: A pointer to a `PhRect_t` structure where the function stores the string’s extent. For the interpretation of the members of this structure, see below.
- `pos`: NULL, or a pointer to a `PhPoint_t` structure that defines an offset that you want to apply to the extent.
- `font`: The base font to use when calculating the extent. You should create this argument by calling `PfGenerateFontName()`.
- `str`: The UTF-8 multibyte string whose extent you want to determine.
- `len`: The number of bytes in the string. If `len` is 0, `PfExtentText()` assumes that it’s `strlen(str)`.

**Library:**

`ph`

**Description:**

This function calculates the extent rectangle of a text string. The base font determines the ascender and descender values of the extent. The width is dependent on the string — the actual font used by characters within the string may be different than this base font (as specified in the `fontext` and `fontmap` files).


PfExtentText() stores the text extent in the PhRect_t that extent points to. The members are used as follows:

- \textit{ul.x}  
  The left bearing.

- \textit{lr.x}  
  The maximum x distance.

- \textit{ul.y}  
  The ascender.

- \textit{lr.y}  
  The descender.

The baseline of the font is at position \( y = 0 \); the width of the string is \( lr.x - \text{min}(ul.x, 0) + 1 \). The height of the string is \( lr.y - ul.y + 1 \).

The resulting extent is offset by the point passed in the PhPoint_t structure pointed to by pos. If pos is NULL, no offset is applied.

If metrics for the base font have been loaded locally (see PfLoadMetrics()) then this extent may be calculated internally; otherwise a request is sent to the font server.

**Returns:**

A pointer to the extent rectangle (extent) if successful, NULL otherwise.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 7 • Pf—Font Server  327
**PfExtentText()**

See also:

* PfExtent(), PfExtentCtx(), PfExtentTextToRect(), PfExtentWideText(), PfFractionalExtentText(), PfGenerateFontName(), PfLoadMetrics(), PgExtentText(), PhPoint_t, PhRect_t*

Fonts chapter of the Photon *Programmer’s Guide*
**PfExtentTextCharPositions()**,  
**PfExtentTextCharPositionsCx()**

*Calculate individual character positions*

**Synopsis:**

```c
#include <photon/Pf.h>

int PfExtentTextCharPositions(
    PhRect_t *ptsExtent,
    PhPoint_t *ptsPos,
    char *psz,
    const char *pckFont,
    int32_t *piIndices,
    int32_t *piPenPositions,
    int32_t iArrayLen,
    uint32_t uiFlags,
    int32_t iBytes,
    uint32_t uiExtentLen,
    PhRect_t const *pktClip );
```

```c
#include <photon/Pf.h>

int PfExtentTextCharPositionsCx(
    struct _Pf_ctrl *context,
    PhRect_t *ptsExtent,
    PhPoint_t *ptsPos,
    char *psz,
    const char *pckFont,
    long adata,
    long bdata,
    int32_t *piIndices,
    int32_t *piPenPositions,
    int32_t iArrayLen,
    uint32_t uiFlags,
    int32_t iBytes,
    uint32_t uiExtentLen,
    PhRect_t const *pktClip );
```

**Arguments:**

- **context** *(PfExtentTextCharPositionsCx() only)* A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().

- **ptsExtent** A pointer to a PhRect_t structure that's used to store the extent of the string.
PfExtentTextCharPositions(), PfExtentTextCharPositionsCx()

- **ptsPos**: A pointer to a `PhPoint_t` structure that's used as an offset to apply against the extent. If NULL, no offset is added to the extent values.

- **psz**: A pointer to a NUL-terminated character string.

- **pckFont**: The font name, as created by `PfGenerateFontName()`.

- **adata**: (`PfExtentTextCharPositionsCx()` only) The horizontal fractional point size, if you set `PF_FRACTIONAL` in the flags argument.

- **bdata**: (`PfExtentTextCharPositionsCx()` only) The vertical fractional point size, if you set `PF_FRACTIONAL` in the flags argument.

- **iArrayLen**: The number of integer entries in the `piIndices` and `piPenPositions` arrays.

- **piIndices**: A pointer to an integer array of length `iArrayLen`. An index corresponds to a location within the string pointed to by `psz`. For example, index 0 relates to the pen’s x position at the start of the string, index 1 corresponds to the pen’s x position after character 1, index 2 corresponds to the pen’s x position after character 2, and so on. In order to function as expected, the indexes must be in numerical order.

- **piPositions**: A pointer to an integer array of length `iArrayLen`. This array contains the resulting pen x values (in pixels) for each index.

- **ulFlags**: A 32-bit value used for flags. Values that can be ORed in are:
  - `PF_WIDE_CHARS` — the characters pointed to by `psz` are an array of `wchar_t` characters. By
PfExtentTextCharPositions(), PfExtentTextCharPositionsCx()

default, the function assumes the characters are multibyte.

This function assumes each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.

- PF_CHAR_DRAW_POSITIONS — if turned on, the bearing x value of the next symbols aren’t applied to the returned pen x positions. This is useful when placing cursors:

  ![Lucy.

If this bit isn’t set, the bearing x value of the next symbols are applied to the pen x positions. This is useful when drawing symbols individually, when you need to know where to place the x origin of each symbol:

  ![Lucy.

iBytes  The number of bytes in the string. If this is 0, the function assumes that the number of bytes is:

  $$\text{strlen( psz ) / wstrlen( psz )}$$

uiExtentLen  The number of characters from the beginning of the string to include in the extent. If 0, the entire string is extented, as permitted by the clipping rectangle.

pktsClip  A clipping rectangle to be used to reduce processing, depending on the value of pktsClip->lr.x (in pixels). If pktsClip is NULL, no clipping is applied.
**PfExtentTextCharPositions(), PfExtentTextCharPositionsCx()**

**Library:**

\[
\begin{align*}
PfExtentTextCharPositions() & \quad ph \\
PfExtentTextCharPositionsCx() & \quad font
\end{align*}
\]

**Description:**

These functions calculate the extent up to `uiExtentLen` code points. They record horizontal pen positions for each code point referenced in `piIndices`, and continue processing until `pktsClip`, if defined, or to the end of the input string `psz`.

**Returns:**

- 0 Success.
- -1 An error occurred; `errno` is set.

**Errors:**

**PfExtentTextCharPositions():**

- **ERANGE** The font manager couldn’t fulfill the request; one of the following is true:
  - The `iArrayLen` argument is larger than `strlen(psz)`.  
  - If index 0 is requested, then `iArrayLen` is larger than `strlen(psz) + 1`.  
  - The `iArrayLen` argument is less than or equal to 0.  
  - An index in `piIndices` references a character greater than `strlen(psz)`.  

- **EFAULT** One of `ptsExtent`, `piIndices`, `piPenPositions`, `pckFont`, or `psz` is NULL.
The font is fixed-width, and an error occurred when trying to retrieve the common width of all characters in that particular font.

Something unexpected occurred while processing a run of characters.

Connection has gone stale, or device error occurred.

Bad message buffer.

Unable to locate render plugin for specified font.

Unable to locate font description.

Unable to locate suitable base font entry.

Insufficient memory to allocate scaling resources.

Fixed width font has invalid size.

Failure to load resources for specified font.

Examples:

/* Typographic positioning example. Demonstrates how to manipulate characters, and sub-strings properly at a typographic pen level. */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>
#include <font_api.h>

int draw( PtWidget_t * ptsWidget, PhTile_t * ptsDamage );

#define FALSE 0
#define __WIN_SIZE_X_ 1000

extern struct _Ph_ctrl * _Ph_;

int main (int argc, char *argv[])
{
    PtArg_t args[8];
    PhPoint_t win_size, pntPOS, pntDIM;
    int i = 0;
    PtWidget_t * win = NULL;
    fprintf(stderr, "POINT : Pen.
" );
    if(PtInit (NULL) == -1)
    { fprintf(stderr, "NOTE : PtInit failed, errno %d\n", errno);
      fprintf(stderr, "FAIL : Pen.\n" );
      exit(EXIT_FAILURE);
    }
    // set base win parms
    win_size.x = 800;
    win_size.y = 600;
    PtSetArg(args[i++], Pt_ARG_DIM, &win_size, 0);
    // window title = name of program
    PtSetArg(args[i++], Pt_ARG_WINDOW_TITLE, "Pen Test Suite", 0);
    if((win = PtCreateWidget (PtWindow, NULL, i, args)) == NULL)
    { fprintf(stderr, "NOTE : Unable to create main window, errno %d\n", errno);
      fprintf(stderr, "FAIL : Pen.\n" );
      exit(EXIT_FAILURE);
    }
    i = 0;
    pntPOS.y = 100;
    pntPOS.x = 75;
    pntDIM.x = __WIN_SIZE_X_ - 75 - 10;
    pntDIM.y = 300;
    PtSetArg(args[i++], Pt_ARG_POS, &pntPOS, 0);
    PtSetArg(args[i++], Pt_ARG_DIM, &pntDIM, 0);
    PtSetArg(args[i++], Pt_ARG_RAW_DRAW_F, draw, 0L);
    PtSetArg(args[i++], Pt_ARG_POINTER, "Hello, this is big Bobby!!", 0L);
    if(PtCreateWidget(PtRaw, win, i, args) == NULL)
    { fprintf(stderr, "NOTE : Unable to create raw canvas, errno %d\n", errno);
      fprintf(stderr, "FAIL : Pen.\n" );
      exit(EXIT_FAILURE);
    }
    PtRealizeWidget(win);
    PtMainLoop ();
    return(0);
}

int draw( PtWidget_t * ptsWidget, PhTile_t * ptsDamage )
{
    char const * text;
    if(PtGetResource(ptsWidget, Pt_ARG_POINTER, &text, 0L) == 0)
    { FontName pucFont;
      struct _PF_ctrl * ctx = _Ph_->font;
      ...
if(PfGenerateFontNameCx(ctx, "PrimaSans BT", 0L, 12L, pucFont) != NULL)
{
    int * piIndx = NULL;
    int * piPos = NULL;
    PhPoint_t pnt;
    pf_point_t tsPos = {0, 0};
    pf_rect_t tsExtent;
    short n = 0, m = 0;
    PgColor_t old;
    PhRect_t tsClip;
    int len = strlen(text);
    char const * str = text;
    int max_chars = 0;
    if((piIndx = (int *)calloc(50, sizeof(int))) == NULL)
    {
        fprintf(stderr, "NOTE : Unable to alloc indices, errno %d.\n", errno);
        fprintf(stderr, "FAIL : Pen.\n");
        exit(EXIT_FAILURE);
    }
    if((piPos = (int *)calloc(50, sizeof(int))) == NULL)
    {
        fprintf(stderr, "NOTE : Unable to alloc positions, errno %d.\n", errno);
        fprintf(stderr, "FAIL : Pen.\n");
        exit(EXIT_FAILURE);
    }
    PtCalcCanvas(ptsWidget, &tsClip);
    PtClipAdd(ptsWidget, &tsClip);
    PtSuperClassDraw(PtBasic, ptsWidget, ptsDamage);
    tsClip.ul.x += 2;
    tsClip.ul.y += 10;
    PgSetTranslation(&tsClip.ul, Pg_RELATIVE);
    while(len > 0)
    {
        wchar_t wc;
        int cl;
        if((cl = mbtowc(&wc, str, MB_CUR_MAX)) <= 0)
        {
            --len, ++str;
            wc = 0;
        }
        else
        {
            len -= cl, str += cl;
            max_chars++;
        }
    }
    len = strlen(text);
    for(n = 0; n < max_chars; n++)
    {
        piIndx[n] = n + 1;
    }
    old = PgSetStrokeColor(Pg_BLACK);
    PgSetFont(pucFont);
    PgSetTextColor(Pg_BLACK);
    if(PfExtentCx(ctx, &tsExtent, &tsPos, pucFont, 0L, 0L, text, len, 0L, NULL) == 0)
    {
        //__STRING_DRAW_
    }
}

Chapter 7 • Pf—Font Server 335
PfExtentTextCharPositions(), PfExtentTextCharPositionsCx()

fprintf(stderr, "NOTE : start Draw text string.
"pnt.x = 10 + tsClip.ul.x;
pnt.y = 10 + tsClip.ul.y;
PgDrawIRect(tsExtent.ul.x + pnt.x, tsExtent.ul.y + pnt.y,
(tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1) + pnt.x,
tsExtent.lr.y + pnt.y, Pg_DRAW_STROKE);
PgDrawText(text, len, &pnt, 0);
PgFlush();

printf("EXTENT_NORMAL: ul.x: %d ul.y: %d lr.x: %d lr.y: %d\n",
tsExtent.ul.x, tsExtent.ul.y, tsExtent.lr.x, tsExtent.lr.y);
printf("POSITIONS: ");
if(PfExtentTextCharPositionsCx(ctx, &tsExtent, &tsPos, text, pucFont,
OL, OL, piIndx, piPos, len, 0L, 0L, NULL) == 0)
{ for(n = 0; n < max_chars; n++)
    printf("%d", piPos[n]);
    printf("\n");
    printf("\nPOS: ul.x: %d ul.y: %d lr.x: %d lr.y: %d\n",
tsExtent.ul.x, tsExtent.ul.y, tsExtent.lr.x, tsExtent.lr.y);
}
else
{ fprintf(stderr, "NOTE : PfExtentTextCharPositions failed, errno %d.\n",
ererrno);
    fprintf(stderr, "FAIL : Pen.\n");
    exit(EXIT_FAILURE);
}

fprintf(stderr, "NOTE : end Draw text string.\n");
/**_SINGULAR_CHAR_DRAW_
for(n = 0; n < max_chars; n++)
    piIndx[n] = n + 1;
fprintf(stderr,
"NOTE : start Draw the string, one character at a time.\n"pnt.x = 10 + tsClip.ul.x;
pnt.y = 50 + tsClip.ul.y;
PgDrawIRect(tsExtent.ul.x + pnt.x, tsExtent.ul.y + pnt.y,
(tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1) + pnt.x,
tsExtent.lr.y + pnt.y, Pg_DRAW_STROKE);

for(n = 0; n < max_chars; n++)
{ PgDrawText(text + n, 1, &pnt, 0);
PgFlush();
pnt.x = 10 + tsClip.ul.x + piPos[n];
    fprintf(stderr, "NOTE : Single[%d]: %d\n", n, piPos[n]);
}
PfExtentTextCharPositions(), PfExtentTextCharPositionsCx()

fprintf(stderr,
    "NOTE : end Draw the string, one character at a time.\n");

//__TWO_CHUNK_DRAW_
fprintf(stderr, "NOTE : start Draw the string in two chunks.\n");
pnt.x = 10 + tsClip.ul.x;
pnt.y = 100 + tsClip.ul.y;
PgDrawText(text, 2, &pnt, 0);
PgFlush();
sleep(1);
pnt.x = 10 + tsClip.ul.x + piPos[1];
PgDrawText(text + 2, len - 2, &pnt, 0);
PgFlush();
fprintf(stderr, "NOTE : end Draw the string in two chunks.\n");

//__PRINT_POSITIONS_
fprintf(stderr, "NOTE : start print positions.\n");
for(n = max_chars - 1; n >= 0; n--)
{ piIndx[0] = n + 1;
  if(PfExtentTextCharPositionsCx(ctx, &tsExtent, &tsPos, text, pucFont,
      0L, 0L, piIndx, piPos, 1, 0L, 0, 0, NULL) == 0)
  { for(m = 0; m < 1; m++)
      fprintf(stderr, "NOTE : Position: %d\n", piPos[m]);
      fprintf(stderr, "NOTE : EXTENT_POS ul.x: %d ul.y: %d lr.x: %d lr.y: %d
", 
          tsExtent.ul.x, tsExtent.ul.y, tsExtent.lr.x, tsExtent.lr.y);
    }
  else
  { fprintf(stderr, "NOTE : PfExtentTextCharPositions failed, errno %d.\n", 
      errno);
    fprintf(stderr, "FAIL : Pen.\n");
    exit(EXIT_FAILURE);
    }
  }
fprintf(stderr, "NOTE : end print positions.\n");

//__DRAW_OVERLAY_
fprintf(stderr, "NOTE : start Draw string, then overlay individual characters on \n top from right to left.\n");
if(PfExtentCx(ctx, &tsExtent, &tsPos, pucFont, 0L, 0L, text, len, 0,
    NULL) == 0)
{ pnt.x = 10 + tsClip.ul.x;
pnt.y = 150 + tsClip.ul.y;
PgDrawRect(tsExtent.ul.x + pnt.x, tsExtent.ul.y + pnt.y,
    tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1
    + pnt.x, tsExtent.lr.y + pnt.y, Pg_DrawStroke);
for(n = max_chars - 1; n >= 0; n--)
{ switch(n)
  { case 0: pnt.x = 10 + tsClip.ul.x;
      PgDrawText(text + 0, len, &pnt, 0);
      
      } 
}
PfExtentTextCharPositions(), PfExtentTextCharPositionsCx()

PgFlush();
break;

default: piIndx[0] = n;
if(PfExtentTextCharPositionsCx(ctx, &tsExtent, &tsPos, text, pucFont, 0L, 0L, piIndx, piPos, 1L, 0L, 0L, 0L, NULL) == -1)
{
fprintf(stderr, "NOTE : PfExtentTextCharPositions failed, errno %d.\n"., errno);
fprintf(stderr, "FAIL : Pen.\n".
exit(EXIT_FAILURE);
} else
{
 fprintf(stderr, "NOTE : Position: %d, piPos[0]\n"., piPos[0]);
pnt.x = 10 + tsClip.ul.x + piPos[0];
PgDrawText(text + n, len - n, &pnt, 0);
PgFlush();
}
break;
}
}
else
{
fprintf(stderr, "NOTE : PfExtentText failed, errno %d.\n"., errno);
fprintf(stderr, "FAIL : Pen.\n".
exit(EXIT_FAILURE);
}
fprintf(stderr, "NOTE : end Draw string, then overlay individual characters on the top from right to left.\n".

//__TEST_OUTORDER_
fprintf(stderr, "NOTE : start Test indices which are non-sequential.\n".
for(n = 0; n < 4; n++)
{
 struct element
  { int first;
    int second;
  };
struct element times[4] = { { 1, 5 }, { 2, 4 }, { 3, 5 }, { 2, 5 }\npiIndx[0] = times[n].first;
piIndx[1] = times[n].second;
if(PfExtentTextCharPositionsCx(ctx, &tsExtent, &tsPos, text, pucFont, 0L, 0L, piIndx, piPos, 2L, 0L, 0L, NULL) == -1)
{
fprintf(stderr, "NOTE : PfExtentTextCharPositions failed, errno %d.\n"., errno);
fprintf(stderr, "FAIL : Pen.\n".
exit(EXIT_FAILURE);
}
else
{
 fprintf(stderr, "NOTE : Position[%d]: %d, times[n].first, piPos[0]\n"., times[n].first, piPos[0]);
 fprintf(stderr, "NOTE : Position[%d]: %d, times[n].second, piPos[1]\n".}
PfExtentTextCharPositions():

#define MAX_INDICES 5
#define FALSE 0

int iaIndex[MAX_INDICES] = {0, 1, 2, 3, 4};
int iaPosition[MAX_INDICES];
PhRect_t tsExtent;
char caBuff[MAX_FONT_TAG];
PfGenerateFontName("Helvetica", 0, 24, caBuff);

if( PfExtentTextCharPositions( &tsExtent, NULL,
   "Lucy", caBuff, iaIndex, iaPosition,
   MAX_INDICES, 0L, 0, 0, NULL ) != EOK)
  printf("Error in PfExtentTextCharPositions() \n");
else
  printf("Pixel penx positions after each character \n in string %s, are as follows: \%d \%d \%d \%d \n", "Lucy", iaPosition[0], iaPosition[1], iaPosition[2], iaPosition[3],

Chapter 7 • Pf—Font Server  339
The pixel pen x positions for each character in the string Lucy are placed in the integer array iaPos, according to the indexes specified in iaIndex. Index 0 corresponds to the position before symbol L, index 1 corresponds to the position after L, index 2 corresponds to the position after u, and so on.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PfExtent(), PfExtentCx(), PfExtentText(), PfExtentTextCharPositionsCx(), PfExtentTextToRect(), PfExtentFractTextCharPositions(), PfFractionalExtentText(), PfGenerateFontName(), PfGenerateFontNameCx(), PhPoint_t, PhRect_t

Fonts chapter of the Photon Programmer’s Guide
**PfExtentTextToRect()**

*Calculate the extent of a string, up to a given rectangle*

**Synopsis:**

```c
#include <photon/Pf.h>
int PfExtentTextToRect(PhRect_t *ptsExtent,
                      char *pkszFont,
                      PhRect_t *ptsRect,
                      char const *pkszString,
                      int iLen);
```

**Arguments:**

- `ptsExtent`: A pointer to a `PhRect_t` structure that's used to store the resultant extent.
- `pkszFont`: The font name, as created by `PfGenerateFontName()`.
- `ptsRect`: A pointer to a `PhRect_t` structure that defines the rectangle that limits the extent.
- `pkszString`: The actual string to extent. The string must be a multibyte string; `wchar_t` strings are not supported.
- `iLen`: The length, in bytes, of `pkszString`. If `iLen` is 0, `strlen(pkszString)` is assumed.

**Library:**

`ph`

**Description:**

`PfExtentTextToRect()` extents a string, `pkszString`, of length `iLen`, and font `pkszFont`, up to the bounds specified by `ptsRect`. The resultant extent, which fits within the bounds of `ptsRect`, is placed in `ptsExtent`.

**Returns:**

The number of characters that will fit within `ptsRect`, or -1 if an error occurred (`errno` is set).
Examples:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ap.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>

PtWidget_t * pwndMain = NULL, * pbtn = NULL, * pobjRaw = NULL;
char pcText[] = "pAfaBfbfffffffffCfcXfxYfyZfzf";
char pcGB[] = "\323\316\317\267";
char szFont[MAX_FONT_TAG];

int fnDrawCanvas( PtWidget_t * pwsWidget,
                 PhTile_t * ptsDamage );

#define BUFFER_SIZE 256

int main (int argc, char *argv[])
{ PtArg_t args[4];
  PhPoint_t win_size, pntPOS, pntDIM;
  short nArgs = 0;
  char * pmbGB = NULL;
  struct PxTransCtrl * ptsTrans = NULL;
  int iTemp1 = 0, iTemp2 = 0;
  if((pmbGB = calloc(BUFFER_SIZE, sizeof(char))) == NULL)
    return(EXIT_FAILURE);
  PtInit (NULL);
  if(argc > 1)
    { if(PfGenerateFontName(argv[1], 0, 9, szFont) == NULL)
       PfGenerateFontName("TextFont", 0, 9, szFont);
    } else
    PfGenerateFontName("TextFont", 0, 9, szFont);
  if((ptsTrans = PxTranslateSet(NULL, "GB2312-80")) == NULL)
    return(EXIT_FAILURE);
  if(PxTranslateToUTF(ptsTrans, pcGB, 4, &iTemp1, pmbGB, BUFFER_SIZE, &iTemp2) == -1)
    printf("Could not translate from GB to UTF.\n");
  if(argc > 2)
    strcpy (pcText, pmbGB);

  // Set base pwndMain parameters.
```
PfExtentTextToRect()

win_size.x = 450;
win_size.y = 450;

PtSetArg(args[0], Pt_ARG_DIM, &win_size, 0);

// window title = name of program
PtSetArg(args[1], Pt_ARG_WINDOW_TITLE, "PfExtentTextToRect", 0);

pwndMain = PtCreateWidget(PtWindow, Pt_NO_PARENT, 2, args);

nArgs = 0;
pntPOS.x = 100;
pntPOS.y = 10;
PtSetArg(args[nArgs], Pt_ARG_POS, &pntPOS, 0);
nArgs++;
PtSetArg(args[nArgs], Pt_ARG_TEXT_STRING, pcText, NULL);
nArgs++;
PtSetArg(args[nArgs], Pt_ARG_TEXT_FONT, szFont, NULL);
nArgs++;
pbtn = PtCreateWidget(PtButton, pwndMain, nArgs, args);
PtRealizeWidget(pbtn);

pntPOS.y = 100;
pntPOS.x = 75;
pntDIM.x = 300;
pntDIM.y = 300;
PtSetArg(args[0], Pt_ARG_POS, &pntPOS, 0);
PtSetArg(args[1], Pt_ARG_DIM, &pntDIM, 0);
PtSetArg(args[2], Pt_ARG_RAW_DRAW_F, fnDrawCanvas, 0L);
pobjRaw = PtCreateWidget(PtRaw, pwndMain, 3, args);
PtRealizeWidget(pwndMain);

PtMainLoop();

free(pmbGB);

return(0);

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage )
{
    PhRect_t tsExtentClip;
    PhRect_t rect;
    PhPoint_t pnt;
    PhRect_t tsExtent;
    PgColor_t old;
    PhPoint_t pnt2;
    PhPoint_t tsPos = {0, 0};
int iRet = 0;
int iBytes = 0;

// find our canvas
PtCalcCanvas(pobjRaw, &rect);
PtSuperClassDraw(PtBasic, ptsWidget, ptsDamage);
old = PgSetStrokeColor(Pg_BLACK);
PfExtentText(&tsExtent, &tsPos, szFont, pcText, strlen(pcText));

// draw text
pnt.x = 10 + rect.ul.x;
pnt.y = 100 + rect.ul.y;
PgSetFont(szFont);
PgSetTextColor(Pg_BLACK);
PgDrawText(pcText, strlen(pcText), &pnt, 0);
pnt.x -= 10;
pnt2.x = pnt.x + tsExtent.lr.x + 20;
pnt2.y = pnt.y;
PgSetStrokeColor(Pg_BLUE);
PgDrawLine(&pnt, &pnt2);
pnt.x = 10 + rect.ul.x;
pnt.y = 100 + rect.ul.y;
PgSetStrokeColor(Pg_RED);
PgDrawIRect(tsExtent.ul.x + pnt.x, tsExtent.ul.y + pnt.y,
            (tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1) +
pnt.x, tsExtent.lr.y + pnt.y, Pg_DRAW_STROKE);

if((iRet = PfExtentTextToRect(&tsExtentClip, szFont,
    &tsExtent, pcText, strlen(pcText))) == -1)
    printf("PfExtentTextToRect failed 1.\n");
else
    {
        printf("lrx == %d, %d characters in string.\n",
            tsExtent.lr.x, utf8strlen(pcText, &iBytes));
        printf("PfExtentTextToRect lrx == %d, %d characters will fit\nin clip of %d.\n", tsExtentClip.lr.x, iRet, tsExtent.lr.x);
    }
tsExtent.lr.x /= 2;
if((iRet = PfExtentTextToRect(&tsExtentClip, szFont, 
    &tsExtent, pcText, 
    strlen(pcText))) == -1)
    printf("PfExtentTextToRect failed 2.\n");
else
    { printf("lrx == %d, %d characters in string.\n", 
        tsExtent.lr.x, utf8strlen(pcText, &iBytes));
        printf("PfExtentTextToRect lrx == %d, %d characters will\n fit in clip of %d.\n", tsExtentClip.lr.x, iRet, tsExtent.lr.x);
    }
pnt.x = 10 + rect.ul.x;
pnt.y = 150 + rect.ul.y;
PgDrawText(pcText, iRet, &pnt, 0);
PgDrawIRect(tsExtentClip.ul.x + pnt.x, 
    tsExtentClip.ul.y + pnt.y, 
    tsExtentClip.lr.x - 
    min(tsExtentClip.ul.x, 0) + 1) + pnt.x, 
    tsExtentClip.lr.y + pnt.y, 
    Pg_DRAW_STROKE);

    tsExtent.lr.x /= 2;

if((iRet = PfExtentTextToRect(&tsExtentClip, szFont, 
    &tsExtent, pcText, 
    strlen(pcText))) == -1)
    printf("PfExtentTextToRect failed 3.\n");
else
    { printf("lrx == %d, %d characters in string.\n", 
        tsExtent.lr.x, utf8strlen(pcText, &iBytes));
        printf("PfExtentTextToRect lrx == %d, %d characters will\n fit in clip of %d.\n", tsExtentClip.lr.x, iRet, tsExtent.lr.x);
    }
pnt.x = 10 + rect.ul.x;
pnt.y = 200 + rect.ul.y;
PgDrawText(pcText, iRet, &pnt, 0);
PgDrawIRect(tsExtentClip.ul.x + pnt.x, 
    tsExtentClip.ul.y + pnt.y, 
    tsExtentClip.lr.x - 
    min(tsExtentClip.ul.x, 0) + 1) + pnt.x, 
    tsExtentClip.lr.y + pnt.y, 
    Pg_DRAW_STROKE);

    PgSetStrokeColor(old);

return( Pt_CONTINUE );
PfExtentTextToRect()

)

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfExtentComponents(), PfExtentFractTextCharPositions(), PfExtent(),
PfExtentCx(), PfExtentText(), PfExtentTextCharPositions(),
PfExtentWideText(), PfFractionalExtentText(),
PfGenerateFontName(), PhRect_t

Fonts chapter of the Photon Programmer’s Guide
PfExtentWideText()

Calculate the extent rectangle of a wide-character string

Synopsis:

```c
#include <photon/Pf.h>
PhRect_t *PfExtentWideText( PhRect_t *extent,
                           PhPoint_t const *pos,
                           const char *font,
                           const uint16_t *str,
                           int len);
```

Arguments:

- **extent** A pointer to a `PhRect_t` structure where the function can store the text extent. The members are:
  - `ul.x` — the left bearing.
  - `lr.x` — the maximum x distance.
  - `ul.y` — the ascender.
  - `lr.y` — the descender.

- **pos** A pointer to a `PhPoint_t` structure that specifies an offset that you want the function to apply to the resulting extent. If `pos` is NULL, no offset is applied.

- **font** The name of the base font. Create this name by calling `PfGenerateFontName()`.

- **str** A wide-character string.

- **len** The length of the string, in bytes. If `len` is 0, the function assumes the string is null-terminated.

Library:

`ph`

Description:

This function calculates the extent rectangle of a text string of wide characters.
PfExtentWideText()

This function assumes each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.

The font determines the ascender and descender values of the extent. The width is dependent on the string — the actual font used by characters within the string may be different than this base font (as specified in the fontext and fontmap files).

If metrics for the base font have been loaded locally (see PfLoadMetrics()) then this extent may be calculated internally; otherwise a request is sent to the font server.

Returns:

A pointer to the extent rectangle (i.e. the same pointer as extent), or NULL if an error occurred.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfExtent(), PfExtentCx(), PfExtentText(), PfExtentTextToRect(), PfFractionalExtentText(), PfGenerateFontName(), PfLoadMetrics(), PfWideTextWidthBytes(), PfWideTextWidthChars(), PgExtentText(), PhPoint_t, PhRect_t

348 Chapter 7 • Pf—Font Server
Font chapter of the Photon Programmer’s Guide

PfExtentWideText()
**PfFindFont**, **PfFindFontCx**

*Generate an ID for a font*

**Synopsis:**

```c
#include <photon/Pf.h>
FontID * PfFindFont( char const * pkcDescription,
                    uint32_t kulFlags,
                    uint32_t kulSize );

#include <font_api.h>
FontID* PfFindFontCx (  
                        struct _Pf_ctrl * context,
                        char const * pkcDescription,
                        uint32_t const kulFlags,
                        uint32_t const kulSize );
```

**Arguments:**

- **context** *(PfFindFontCx only)* A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- **pkcDescription**
  
  The foundry name of the requested font, e.g. *Helvetica*.
- **kulFlags** Flags that identify the requested font type; any combination of:
  - PF_STYLE_BOLD
  - PF_STYLE_ITALIC
  - PF_STYLE_ANTIALIAS
  - PF_STYLE_ULINE
  - PF_STYLE_DULINE
- **kulSize** The point size requested, e.g. 12.

**Library:**

- PfFindFont() *ph*
- PfFindFontCx() *font*
PfFindFont(), PfFindFontCx()

Description:

These functions compose a FontID from the provided foundry descriptive name, flags, and point size.

Returns:

A pointer to a font ID that you can use with other font functions, or NULL if an error occurred.

Errors:

ESRCH    Unable to locate font.
ENOMEM   There wasn’t enough memory to perform the desired request.

This function can also set errno to one of the values generated by PfQueryFonts().

Examples:

PfFindFontCx(): See the examples for PfConvertFontIDCx(), PfGetGlyphIndexCx(), and PfRenderCx().

PfFindFont():

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ap.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>

PtWidget_t * pwndMain = NULL, * pbtn = NULL, * pobjRaw = NULL;
char pcText[100] = "AaBbCcXxYyZz";
char ** ppcData = NULL;

int fnDrawCanvas( PtWidget_t * ptsWidget,
                 PhTile_t * ptsDamage );

#define FALSE 0

static FontID * gs_ptsID = NULL;
```c
int main(int argc, char *argv[]) {
    PtArg_t args[4];
    PhPoint_t win_size, pntPOS, pntDIM;
    short nArgs = 0;

    PtInit(NULL);

    ppcData = argv;

    if(argc < 2) {
        printf("Usage: pf2id descriptive_name, e.g. pf2id Helvetica\n");
        exit(EXIT_FAILURE);
    }

    // set base pwndMain parms
    win_size.x = 450;
    win_size.y = 450;

    // window title = name of program
    PtSetArg(&args[0], Pt_ARG_DIM, &win_size, 0);
    PtSetArg(&args[1], Pt_ARG_WINDOW_TITLE, "Pf2", 0);

    pwndMain = PtCreateWidget(PtWindow, Pt_NO_PARENT, 2, args);

    if(argc > 2) {
        strcpy(pcText, argv[2]);
        nArgs = 0;
        pntDIM.x = 80;
        pntDIM.y = 20;
        PtSetArg(&args[0], Pt_ARG_DIM, &pntDIM, 0);
        nArgs++;
        pntPOS.x = 100;
        pntPOS.y = 10;
        PtSetArg(&args[1], Pt_ARG_POS, &pntPOS, 0);
        nArgs++;
        PtSetArg(&args[2], Pt_ARG_TEXT_STRING, pcText, NULL);
        nArgs++;

        // Find the font we desire.
        gs_ptsID = PfFindFont(argv[1], 0, 12);

        PtSetArg(&args[3], Pt_ARG_TEXT_FONT, PfConvertFontID(gs_ptsID), NULL);
        nArgs++;
        pbtn = PtCreateWidget(PtButton, pwndMain, nArgs, args);
        PtRealizeWidget(pbtn);
    }
}
```
PfFindFont(), PfFindFontCx()

```c
pntPOS.y = 100;
pntPOS.x = 75;
pntDIM.x = 300;
pntDIM.y = 300;
PtSetArg(args[0], Pt_ARG_POS, &pntPOS, 0);
PtSetArg(args[1], Pt_ARG_DIM, &pntDIM, 0);
PtSetArg(args[2], Pt_ARG_RAW_DRAW_F, fnDrawCanvas, 0L);
pobjRaw = PtCreateWidget(PtRaw, pwndMain, 3, args);

(void) PtRealizeWidget(pwndMain);

printf("Size: %d\n", PfFontSize(gs_ptsID));
printf("Descriptive Name: %s\n", 
        PfFontDescription(gs_ptsID));

if(PfFontFlags(gs_ptsID) & PF_SCALABLE)
    printf("Scalable font.\n");
else if(PfFontFlags(gs_ptsID) & PF_BITMAP)
    printf("Bitmap font.\n");

PtMainLoop();
PfFreeFont(gs_ptsID);  // Free the FontID resources.
return EXIT_SUCCESS;
}

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage )
{
    PhRect_t rect;
    PhPoint_t pnt;
    PhRect_t tsExtent;
    PgColor_t old;
    PhPoint_t pnt2;
    PhPoint_t tsPos = {0, 0};

    // find our canvas
    PtCalcCanvas(pobjRaw, &rect);
    PtSuperClassDraw(PtBasic, ptsWidget, ptsDamage);

    old = PgSetStrokeColor(Pg_BLACK);

    PfExtentText(&tsExtent, &tsPos,
        PfConvertFontID(gs_ptsID), pcText,
        strlen(pcText));

    // draw text
    pnt.x = 10 + rect.ul.x;
    pnt.y = 100 + rect.ul.y;

    PgSetFont(PfConvertFontID(gs_ptsID));
```
PfFindFont(), PfFindFontCx()

PgSetTextColor(Pg_BLACK);
PgDrawText(pcText, strlen(pcText), &pnt, 0);

pnt.x -= 10;
pnt2.x = pnt.x + tsExtent.lr.x + 20;
pnt2.y = pnt.y;

PgSetStrokeColor(Pg_RED);
PgDrawLine(&pnt, &pnt2);
PgSetStrokeColor(old);

return( Pt_CONTINUE );
}

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:
PfConvertFontID(), PfConvertFontIDCx(), PfDecomposeStemToID(), PfDecomposeStemToIDCx(), PfFontDescription(), PfFontDescriptionCx(), PfFontFlags(), PfFontFlagsCx(), PfFontSize(), PfFontSizeCx(), PfFreeFont(), PfFreeFontCx(), PfGenerateFontName(), PfGenerateFontNameCx(), Fonts chapter of the Photon Programmer’s Guide
**PfFontBaseStem(), PfFontBaseStemCx()**

*Get the base stem associated with a given font ID*

**Synopsis:**

```c
#include <photon/Pf.h>
char const * PfFontBaseStem( FontID *ptsID );

#include <font_api.h>
char const* PfFontBaseStemCx( struct _Pf_ctrl * context,
                                FontID * ptsID );
```

**Arguments:**

- `context` (PfFontBaseStemCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `ptsID` A pointer to a font ID, as returned by PfFindFont() or PfFindFontCx().

**Library:**

```
PfFontBaseStem()
    ph

PfFontBaseStemCx()
    font
```

**Description:**

These functions get the base stem that’s associated with the font ID pointed to by `ptsID`. Examples of a base stem are `helv` and `swiss`. This routine is useful when differentiating between two or more installed fonts with identical descriptive names (see PfFontDescription()).

**Returns:**

A pointer to the base stem, or NULL if `ptsID` is NULL.
PfFontBaseStem(), PfFontBaseStemCx()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfFontDescription(), PfFontDescriptionCx()

Fonts chapter of the Photon Programmer’s Guide
**PfFontDescription(), PfFontDescriptionCx()**

*Get the foundry name of a font*

**Synopsis:**

```c
#include <photon/Pf.h>
char const *PfFontDescription( FontID *ptsID );

#include <font_api.h>
char const* PfFontDescriptionCx( 
    struct __Pf_ctrl * context,
    FontID * ptsID );
```

**Arguments:**

- `context` *(PfFontDescriptionCx() only)* A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `ptsID` A pointer to a font ID, as returned by PfFindFont() or PfFindFontCx().

**Library:**

- `PfFontDescription()`
  - `ph`
- `PfFontDescriptionCx()`
  - `fontlib`

**Description:**

These functions get the foundry name specific to the font ID pointed to by `ptsID`. These routines don’t check to see if `ptsID` is non-NULL.

**Returns:**

The foundry name specific to the `FontID`, e.g. **Comic Sans MS**.
**PfFontDescription(), PfFontDescriptionCx()**

**Examples:**

*PfFontDescriptionCx():* See the example for *PfConvertFontIDCx()*.  
*PfFontDescription():* See *PfFindFont()*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt handler</td>
</tr>
<tr>
<td></td>
<td>Signal handler</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PfConvertFontID(), PfConvertFontIDCx(), PfDecomposeStemToID(),*  
*PfDecomposeStemToIDCx(), PfFindFont() PfFindFontCx(),*  
*PfFontBaseStem(), PfFontBaseStemCx(), PfFontFlags(),*  
*PfFontFlagsCx(), PfFontSize(), PfFontSizeCx(), PfFreeFont(),*  
*PfFreeFontCx(), PfGenerateFontName(), PfGenerateFontNameCx()*

Fonts chapter of the Photon *Programmer’s Guide*
**PfFontFlags(), PfFontFlagsCx()**

*Get the flags for a font*

**Synopsis:**

```c
#include <photon/Pf.h>
uint32_t PfFontFlags( FontID *ptsID );

#include <font_api.h>
uint32_t PfFontFlagsCx( struct _Pf_ctrl *context,
                        FontID *ptsID );
```

**Arguments:**

- `context` (PfFontFlagsCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `ptsID` A pointer to a font ID, as returned by PfFindFont() or PfFindFontCx().

**Library:**

PfFontFlags()  ph

PfFontFlagsCx()  font

**Description:**

These functions get the flags associated with the font ID pointed to by `ptsID`. The flags can be any of the following:

- `PF_STYLE_BOLD`
- `PF_STYLE_ITALIC`
- `PF_STYLE_BI` — the same as `PF_STYLE_BOLD | PF_STYLE_ITALIC`
- `PF_STYLE_ANTIALIAS`
- `PF_SCALABLE`
- `PF_BITMAP`

Chapter 7 • Pf—Font Server 359
**PfFontFlags(), PfFontFlagsCx()**

**Returns:**

A 32-bit value containing the flags for this FontID.

**Examples:**

- PfFontFlagsCx(): See the example for PfConvertFontIDCx().
- PfFontFlags(): See PfFindFont().

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PfConvertFontID(), PfConvertFontIDCx(), PfDecomposeStemToID(), PfDecomposeStemToIDCx(), PfFindFont(), PfFindFontCx(), PfFontDescription(), PfFontDescriptionCx(), PfFontSize(), PfFontSizeCx(), PfFreeFont(), PfFreeFontCx(), PfGenerateFontName(), PfGenerateFontNameCx()

Fonts chapter of the Photon *Programmer’s Guide*
**PfFontSize(), PfFontSizeCx()**  
*Get the point size of a font*

**Synopsis:**

```c
#include <photon/Pf.h>
uint32_t PfFontSize( FontID * ptsID );

#include <font_api.h>
uint32_t PfFontSizeCx( struct _Pf_ctrl * context,
                        FontID * ptsID );
```

**Arguments:**

- `context` *(PfFontSizeCx() only)* A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `ptsID` A pointer to a font ID, as returned by `PfFindFont()` or `PfFindFontCx()`.

**Library:**

- `PfFontSize()` *ph*
- `PfFontSizeCx()` *font*

**Description:**

These functions get the point size associated with the font ID pointed to by `ptsID`. These routines don’t verify that `ptsID` is non-NULL.

**Returns:**

The point size of the font.

**Examples:**

- `PfFontSizeCx()`: See the example for `PfConvertFontIDCx()`.
- `PfFontSize()`: See `PfFindFont()`.
**PfFontSize(), PfFontSizeCx()**

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PfConvertFontID(), PfConvertFontIDCx(), PfDecomposeStemToID(), PfDecomposeStemToIDCx(), PfFindFont(), PfFindFontCx(), PfFontDescription(), PfFontDescriptionCx(), PfFontFlags(), PfFontFlagsCx(), PfFreeFont(), PfFreeFontCx(), PfGenerateFontName(), PfGenerateFontNameCx()

Fonts chapter of the Photon *Programmer’s Guide*
PfFontTypeCx

Retrieve the font type description

Synopsis:

```c
#include <font_api.h>
int PfFontTypeCx( struct _Pf_ctrl * context,
                  char const * font,
                  char * pcBuffer,
                  int iBufferLen );
```

Arguments:

- `context`: A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `font`: A string that contains the base font stem. You should create this argument by calling `PfGenerateFontNameCx()`.
- `pcBuffer`: A pointer to a buffer in which the function can store the font type.
- `iBufferLen`: Length, in bytes, of `pcBuffer`.

Library:

- `font`

Description:

This function retrieves a string from the font server describing what type of font technology is used to process the provided `font`. For example, Adobe Type 1, TrueType/T2K, and Bitstream(FFS).

Returns:

- 0: Success
- -1: An error occurred (`errno` is set).
**PfFontTypeCx()**

**Errors:**

- **ESRCH**  Unable to locate device.
- **EBADF**  Connection has gone stale, or device error occurred.
- **ENETUNREACH**  Bad message buffer.
- **ELIBACC**  Unable to locate render plugin for specified font.
- **ENOENT**  Unable to locate suitable base font entry.
- **EINVAL**  Failure to load resources for specified font.
- **ENOMEM**  Insufficient memory to allocate scaling resources.
- **EFAULT**  Provided font, or buffer is NULL, or buffer len is <= zero.
- **EFAULT**  Unable to locate satisfactory base font.
- **EINVAL**  Render plugin unable to satisfy request.

**Classification:**

- Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>
PfFontTypeCx()

See also:

PfAttachCx(), PfAttachDllCx(), PfGenerateFontNameCx().

Fonts chapter of the Photon Programmer’s Guide
**PfFractionalExtentText()**

*Calculate the extent rectangle of a text string, using fractional scaling*

**Synopsis:**

```c
#include <photon/Pf.h>
PhRect_t *PfFractionalExtentText(
    PhRect_t *extent,
    PhPoint_t const *pos,
    const char *font,
    long xsize,
    long ysize,
    const char *str,
    int len);
```

**Arguments:**

- `extent` A pointer to a `PhRect_t` structure where the function stores the string’s extent. For the interpretation of the members of this structure, see below.
- `pos` NULL, or a pointer to a `PhPoint_t` structure that defines an offset that you want to apply to the extent.
- `font` The base font to use when calculating the extent. You should create this argument by calling `PfGenerateFontName()`.
- `xsize, ysize` The size of the font in 16.16 fixed-point format.
- `str` The UTF-8 multibyte string whose extent you want to determine.
- `len` The number of bytes in the string. If `len` is 0, `PfExtentText()` assumes that it’s `strlen(str)`.

**Library:**

`ph`
**PfFractionalExtentText()**

**Description:**

This function calculates the extent rectangle of a text string. The base font determines the ascender and descender values of the extent. The *xsize* and *ysize* arguments define the size of the font in 16.16 fixed-point format.

This function is intended to be used with scalable fonts. If a bitmap font is provided, an attempt is made to map the font to a scalable equivalent, though success isn’t guaranteed.

*PfFractionalExtentText()* stores the text extent in the *PhRect_t* pointed to by *extent*. The members are used as follows:

- *ul.x*  The left bearing.
- *lr.x*  The maximum x distance.
- *ul.y*  The ascender.
- *lr.y*  The descender.

The baseline of the font is at position *y*=0; the width of the string is *lr.x - min(ul.x, 0) + 1*. The height of the string is *lr.y - ul.y + 1*.

If metrics for the base font have been loaded locally (see *PfLoadMetrics()*)) then this extent may be calculated internally; otherwise a request is sent to the font server.

**Returns:**

A pointer to the extent rectangle (*extent*) if successful, **NULL** otherwise.

**Classification:**

Photon
PfFractionalExtentText()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfExtent(), PfExtentCx(), PfExtentText(), PfExtentTextToRect(), PfGenerateFontName(), PfLoadMetrics(), PgExtentText(), PhPoint_t, PhRect_t

Fonts chapter of the Photon Programmer's Guide
PfFreeFont(), PfFreeFontCx()

Release resources associated with a font

Synopsis:

```c
#include <photon/Pf.h>
long PfFreeFont( FontID *ptsID );

#include <font_api.h>
long PfFreeFontCx( struct _Pf_ctrl * context,
                   FontID * ptsID );
```

Arguments:

- `context` (PfFreeFontCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `ptsID` A pointer to a font ID, as returned by PfFindFont() or PfFindFontCx().

Library:

- `PfFreeFont()` ph
- `PfFreeFontCx()` font

Description:

These functions release all resources bound to the font ID pointed to by `ptsID`.

Returns:

- `0L` Success.
- `-1L` An error occurred; `errno` is set.

Errors:

- `EFAULT` The `ptsID` argument is NULL.
**PfFreeFont(), PfFreeFontCx()**

**Examples:**

*PfFreeFontCx():* See the examples for *PfConvertFontIDCx(), PfGetGlyphIndexCx(),* and *PfRenderCx().*  
*PfFreeFont():* See *PfFindFont().*

**Classification:**

*Photon*

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfConvertFontID(), PfConvertFontIDCx(), PfDecomposeStemToID(),*  
*PfDecomposeStemToIDCx(), PfFindFont(), PfFindFontCx(),*  
*PfFontDescription(), PfFontDescriptionCx(), PfFontFlags(),*  
*PfFontFlagsCx(), PfFontSize(), PfFontSizeCx(),*  
*PfGenerateFontName(), PfGenerateFontNameCx(),*  

Fonts chapter of the Photon *Programmer’s Guide*
**PfGenerateFontName(), PfGenerateFontNameCx()**

Generate a font name

**Synopsis:**

```c
#include <photon/Pf.h>
char * PfGenerateFontName(
    char const * pkcDescription,
    uint32_t kuiFlags,
    uint32_t kuiSize,
    char * pcBuff );

#include <font_api.h>
char * PfGenerateFontNameCx(
    struct _Pf_ctrl * context,
    char const * pkcDescription,
    uint32_t kuiFlags,
    uint32_t kuiSize,
    char * pcBuff );
```

**Arguments:**

- **context** A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- **pkcDescription** The descriptive name of the font, usually the name provided by the font foundry, e.g. Helvetica.
- **kuiFlags** The attributes to apply to the font. The following bits can be ORed together:
  - `PF_STYLE_BOLD`
  - `PF_STYLE_ITALIC`
  - `PF_STYLE_ANTIALIAS`
  - `PF_STYLE_ULINE`
  - `PF_STYLE_DULINE`
- **kuiSize** The requested point size, e.g. 12.
- **pcBuff** A buffer in which to store the resulting font identifier. This must be of size `MAX_FONT_TAG`. 
PfGenerateFontName(), PfGenerateFontNameCx()

Library:

PfGenerateFontName()

    ph

PfGenerateFontNameCx()

    font

Description:

PfGenerateFontName() is a convenience function that generates proper font names from the given arguments.
PfGenerateFontNameCx() is similar to PfGenerateFontName(), but lets you specify the font context.

WARNING: PfGenerateFontName() uses a global context, so you must call PhAttach() or PhInit() first to set up the global context. If you don’t, your application will crash.

Returns:

NULL on failure, pcbuff on success.

Errors:

ENOMEM Not enough memory to proceed with the request.
EFAULT One of the required parameters is NULL.
EINVAL The requested size is 0.
ESRCH Unable to locate font.

Examples:

PfGenerateFontNameCx(): See the examples for PfConvertFontIDCx() and PfExtentCx().
PfGenerateFontName():
The `szHelvetica12` variable can now be used with any function that takes a “font” pointer, such as `PfExtentText()`, `PfExtentTextCx()`, `PfGlyph()`, `PfRenderCx()` or `PfRender()`.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PfAttach()`, `PfAttachCx()`, `PfDecomposeStemToID()`, `PfDecomposeStemToIDCx()`, `PfQueryFontInfo()`, `PfQueryFontInfoCx()`, `PfQueryFonts()`, `PfQueryFontsCx()`

Fonts chapter of the Photon *Programmer’s Guide*
PfGetGlyphIndexCx()

Get a glyph index

Synopsis:

```c
#include <font_api.h>
int PfGetGlyphIndexCx( struct _Pf_ctrl * context,
                      wchar_t glyph,
                      FontID * font,
                      uint32_t * pIndex );
```

Arguments:

- **context** A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- **glyph** A Unicode glyph value.
- **font** A pointer to a font ID, as returned by `PfFindFontCx()`.
- **pIndex** A pointer to a `uint32_t` where the function can store the glyph index.

Library:

```
font
```

Description:

This function retrieves the glyph index for the provided `font` and `glyph` combination.

Returns:

- 0 Success
- -1 An error occurred (`errno` is set).

Errors:

- ESRCH Unable to locate device.
- EBADF Connection has gone stale, or device error occurred.
PfGetGlyphIndexCx()

NETUNREACH
    Bad message buffer.

ELIBACC
    Unable to locate render plugin for specified font.

EINVAL
    Failure to load resources for specified font.

EFAULT
    Unable to locate suitable base font.

ERANGE
    Unable to locate suitable base font for provided code point.

EINVAL
    Render plugin is unable to satisfy request.

Examples:

```c
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <errno.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <dirent.h>
#include <photon/PhT.h>
#include <photon/PhProto.h>
#include <Pt.h>
#include <Ph.h>
#include <font_api.h>

FontName name;
uint16_t wc = 0x65;

int font_changed(PtWidget_t * widget, void * data, PtCallbackInfo_t * cbinfo)
{    snprintf(name, sizeof(name), "%s", cbinfo->cbdata);
    printf("%s\n", name);
    PtDamageWidget(data);
    return (Pt_CONTINUE);
}

void raw_draw(PtWidget_t * widget, PhTile_t * damage)
{    PhArea_t area;
    PhRect_t tsClip;
    PhPoint_t pos = { 10, 10 };
    FontQueryInfo info;
    wchar_t w;
    FontID * id;
    pf_metrics_t * metric;
    FontRender * render;
```
PfGetGlyphIndexCx()

PtSuperClassDraw(PtBasic, widget, damage);
PtCalcCanvas(widget, atsClip);
PgDrawRect(atsClip, Pg_DRAW_FILL);
PtClipAdd(widget, atsClip);
tsClip.ul.x += 2;
tsClip.ul.y += 10;
PtClipRemove();

PtSetTranslation(&atsClip.ul, Pg_RELATIVE);
PtSetTextColor(Pg_BLACK);
PtSetFont(name);

if(PfQueryFontInfoCx(PfDefaultContext(), name, &info) == -1)
{ fprintf(stderr, "NOTE : PfQueryFontInfoCx failed, errno %d.\n", errno);
  fprintf(stderr, "FAIL : Glyph capture.\n");
  exit(EXIT_FAILURE);
}
pos.y += -info.ascender;
w = wc;

if((id = PfDecomposeStemToIDCx(PfDefaultContext(), name)) != NULL)
{ FontRender r;
  int size = 32000;
  char * bitmap = calloc(size, sizeof(char));

  if(PfGlyphCx(PfDefaultContext(), name, w, &r, bitmap, size, NULL) == -1)
  { fprintf(stderr, "NOTE : PfGlyphCx failed, errno %d.\n", errno);
    fprintf(stderr, "FAIL : Glyph capture.\n");
    free(bitmap);
    exit(EXIT_FAILURE);
  }
  else
  { pos.y -= r.offset.y;
    PgDrawText(wc, sizeof(wc), &pos, Pg_TEXT_WIDECHAR);
  }

  free(bitmap);
  PfFreeFontCx(PfDefaultContext(), id);
}
else
{ fprintf(stderr, "NOTE : PfDecomposeStemToIDCx failed, errno %d.\n", errno);
  fprintf(stderr, "FAIL : Glyph capture.\n");
  exit(EXIT_FAILURE);
}

tsClip.ul.x *= -1;
tsClip.ul.y *= -1;
PtSetTranslation(&atsClip.ul, Pg_RELATIVE);
PtClipRemove();

int main(int argc, char *argv[])
{ PtArg_t tsArg[10];
  int i;
  PhDim_t dim = { 300, 300 };
  PtWidget_t * win, * raw, * fontsel;
  PhPoint_t pos = {0,0};
  PhRect_t canvas;

376 Chapter 7 • Pf—Font Server
int flags;
uint32_t index;
FontID * id;

fprintf(stderr, "POINT : Glyph capture.\n");

if(PtInit(NULL) == -1)
{
    fprintf(stderr, "NOTE : PtInit failed, no photon.\n");
    fprintf(stderr, "FAIL : Glyph Capture.\n");
    return(EXIT_FAILURE);
}

PgSetDrawBufferSize(0xFFFF);
i = 0;
PtSetArg(&tsArg[i++], Pt_ARG_DIM, &dim, 0L);
PtSetArg(&tsArg[i++], Pt_ARG_WINDOW_TITLE, "Glyph Capture", 0L);
PtSetArg(&tsArg[i++], Pt_ARG_FLAGS, Pt_GETS_FOCUS, Pt_GETS_FOCUS);
if((win = PtCreateWidget(PtWindow, NULL, i, tsArg)) == NULL)
{
    fprintf(stderr, "NOTE : Unable to create window.\n");
    fprintf(stderr, "FAIL : Glyph Capture.\n");
    return(EXIT_FAILURE);
}

PtCalcCanvas(win, &canvas);
i = 0;
dim.w = canvas.lr.x - canvas.ul.x;
dim.h = (canvas.lr.y / 2) - canvas.ul.y;
PtSetArg(&tsArg[i++], Pt_ARG_DIM, &dim, 0L);
PtSetArg(&tsArg[i++], Pt_ARG_RAW_DRAW_F, raw_draw, 0L);
pos.x = 0;
pos.y = dim.h;
PtSetArg(&tsArg[i++], Pt_ARG_POS, &pos, 0L);
PtSetArg(&tsArg[i++], Pt_ARG_FILL_COLOR, Pg_YELLOW, 0L);
flags = Pt_RIGHT_ANCHORED_RIGHT | Pt_LEFT_ANCHORED_LEFT | Pt_TOP_ANCHORED_TOP
        | Pt_BOTTOM_ANCHORED_BOTTOM;
PtSetArg(&tsArg[i++], Pt_ARG_ANCHOR_FLAGS, flags, flags);
if((raw = PtCreateWidget(PtRaw, win, i, tsArg)) == NULL)
{
    fprintf(stderr, "NOTE : Unable to create raw canvas.\n");
    fprintf(stderr, "FAIL : Glyph Capture.\n");
    return(EXIT_FAILURE);
}

i = 0;
dim.w = canvas.lr.x - canvas.ul.x;
dim.h = (canvas.lr.y / 2) - canvas.ul.y;
PtSetArg(&tsArg[i++], Pt_ARG_DIM, &dim, 0L);
pos.x = 0;
pos.y = 0;
PtSetArg(&tsArg[i++], Pt_ARG_POS, &pos, 0L);
flags = Pt_RIGHT_ANCHORED_RIGHT | Pt_LEFT_ANCHORED_LEFT | Pt_TOP_ANCHORED_TOP
        | Pt_BOTTOM_ANCHORED_BOTTOM;
PtSetArg(&tsArg[i++], Pt_ARG_ANCHOR_FLAGS, flags, flags);
PtSetArg(&tsArg[i++], Pt_ARG_FONT_FLAGS, 0L, Pt_FONTSEL_SAMPLE);
if((fontsel = PtCreateWidget(PtFontSel, win, i, tsArg)) == NULL)
{
    fprintf(stderr, "NOTE : Unable to create fontsel.\n");
}
PfGetGlyphIndexCx()

```c
    fprintf(stderr, "FAIL : Glyph Capture.\n");
    return(EXIT_FAILURE);
}
PtAddCallback(fontsel, Pt_CB_FONT_MODIFY, font_changed, raw);

id = PfFindFontCx(PfDefaultContext(), "TextFont", 0L, 9L);
PfConvertFontIDCx(PfDefaultContext(), id, name);

if(PfGetGlyphIndexCx(PfDefaultContext(), wc, id, &index) == -1)
    {
        fprintf(stderr, "NOTE : Unable to fetch index, errno %d.\n", errno);
        fprintf(stderr, "FAIL : Glyph Capture.\n");
        return(EXIT_FAILURE);
    }
else
    {
        fprintf(stderr, "NOTE : index is %d\n", index);
    }
PtRealizeWidget(win);
PtMainLoop();
return(EXIT_SUCCESS);
```

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

PfAttachCx(), PfAttachDllCx(), PfFindFontCx().

Fonts chapter of the Photon Programmer’s Guide
Get individual point information for a glyph outline

Synopsis:

```
#include <photon/Pf.h>
long PfGetOutline( char const *pkucFont,
                  unsigned long ulSymbol,
                  PHFONT_METRICS *ptsMetrics,
                  PhPoint_t **pptsPoints,
                  int **ppiLoops );
```

```
#include <font_api.h>
long PfGetOutlineCx( struct _Pf_ctrl *context,
                    char const *pkucFont,
                    unsigned long ulSymbol,
                    PHFONT_METRICS *ptsMetrics,
                    PhPoint_t **pptsPoints,
                    int **ppiLoops );
```

Arguments:

- `context` *(PfGetOutlineCx() only)* A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `pkucFont` The font name, as created by PfGenerateFontName() or PfGenerateFontNameCx().
- `ulSymbol` The Unicode value of the glyph whose outline is to be determined. Valid values are 0x0000 to 0xFFFF.
- `ptsMetrics` A pointer to a structure of type PHFONT_METRICS, which contains at least the following members:
  ```
  int32_t Advance; // 16.16 format
  int32_t BearingX; // 16.16 format
  int32_t BearingY; // 16.16 format
  int32_t MaxX; // 16.16 format
  int32_t Height; // 1.1 format (pixel)
  int32_t Width; // 1.1 format (pixel)
  ```
- `pptsPoints` A pointer to a pointer of type PhPoint_t. The function allocates memory and stores the returned outline points within this memory.
PfGetOutline(), PfGetOutlineCx()

ppiLoops A pointer to a pointer of type int. The function allocates memory and stores the number of loop iterations per contour within this memory.

Library:

PfGetOutline()

ph

PfGetOutlineCx()

font

Description:

These functions provide individual point information, in pixel coordinates, for a glyph outline. These points can be transformed in any way desired. In order to fill the resultant outlines, there are several possible routes:

- Even-odd fill algorithm (this will miss some polygons)
- Nonzero winding fill algorithm (preferred)
- Visual inspection of the outline, with manual determination of which polygons to fill, and with what colour.

Your application must free the memory pointed to by pptsPoints and ppiLoops.

PfGetOutlineCx() is similar to PfGetOutline(), but lets you specify the font context.

Returns:

The number of contours that make up the outline, or -1 if an error occurred (errno is set).
Examples:

**PfGetOutlineCx()**:

```c
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <limits.h>
#include <errno.h>
#include <Ph.h>
#include <Pt.h>

int draw_canvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage )
{
    #define DESC_FONT "PrimaSans BT"

    int main(int argc, char *argv[])
    {
        PhPoint_t win_size, pos, dim;
        PtWidget_t * wnd, * raw;
        FontName font;
        PhRect_t extent;
        long lAscender = 0L;

        PtInit (NULL);
        // set base pwndMain parms
        win_size.x = 400;
        win_size.y = 400;

        PtSetArg(&args[0], Pt_ARG_DIM, &win_size, 0);
        PtSetArg(&args[1], Pt_ARG_WINDOW_TITLE, (long)"Outline Test", 0);
        wnd = PtCreateWidget (PtWindow, Pt_NO_PARENT, 2, args);

        if(PfGenerateFontName(DESC_FONT, 0, 36, font) == NULL)
            return(Pt_CONTINUE);

        PfExtentText(&extent, NULL, font, "M", 0);
        lAscender = 200;

        PtSetArg(&args[0], Pt_ARG_POS, &pos, 0);
        PtSetArg(&args[1], Pt_ARG_DIM, &dim, 0);
        PtSetArg(&args[2], Pt_ARG_RAW_DRAW_F, draw_canvas, 0L);
        PtSetArg(&args[3], Pt_ARG_POINTER, font, 0L);

        raw = PtCreateWidget(PtRaw, wnd, 4, args);

        PtRealizeWidget(wnd);
        PtMainLoop ();
        return(EXIT_SUCCESS);
    }
}
```

Chapter 7 • Pf—Font Server 381
PfGetOutline(), PfGetOutlineCx()

```c
long s_lAdvanceY = 0L;
int * loops;

int draw_canvas( PtWidget_t * widget, PhTile_t * damage )
{
    PgColor_t old;
    pf_point_t * pnt = NULL;
    pf_metrics_t metrics;
    long contours = 0L;
    struct _Pf_ctrl * pf;
    PhRect_t rect;
    s_lAdvanceY = 0L;

    // find our canvas
    PtCalcCanvas(widget, &rect);
    PtSuperClassDraw( PtBasic, widget, damage );

    old = PgSetStrokeColor(Pg_BLACK);

    if((pf = PfAttachCx(NULL, 0)) != NULL)
    {
        char const * pfont;
        PtGetResource(widget, Pt_ARG_POINTER, &pfont, 0L);

        if((contours = PfGetOutlineCx(pf, pfont, 'i', &metrics, &pnt, &loops)) == -1L)
            return(Pt_CONTINUE);
        else
        {
            PhPoint_t pos;
            long ii;
            int offset = 0;

            pos.x = (-metrics.BearingX >> 16) + rect.ul.x;
            pos.y = (metrics.BearingY >> 16) + rect.ul.y;

            for(ii = 0L; ii < contours; ii++)
            {
                PgDrawPolygon(pnt + offset, loops[ii], &pos, Pg_DRAW_STROKE);
                offset += loops[ii];
            }

            free(pnt);
            free(loops);
        }

        PfDetachCx(pf);
    }

    PgSetStrokeColor(old);
    return( Pt_CONTINUE );
}

PfGetOutline():

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

382 Chapter 7 • Pf—Font Server
#include <sys/mman.h>
#include <unistd.h>
#include <limits.h>
#include <errno.h>
#include <Ph.h>
#include <Pt.h>

int DrawOutline( PhPoint_t * pnt, long ContourCount,
                 PhRect_t * rect, long lAscender);
int fnDrawCanvas( PtWidget_t * ptsWidget,
                 PhTile_t * ptsDamage );

PtWidget_t * pwndMain = NULL, * pobjRaw = NULL;
PhRect_t rect;
#define DESC_FONT "PrimaSans BT"
PhRect_t tsExtent;
FontName szFont;
long lAscender = 0;
int bDrawLine = 0;

int main(int argc, char *argv[])
{ PtArg_t args[4];
  PhPoint_t win_size, pntPOS, pntDIM;
  PtInit (NULL);
  if(argc > 1)
    bDrawLine = 1;

  // set base pwndMain parms
  win_size.x = 400;
  win_size.y = 400;
  PtSetArg(&args[0],Pt_ARG_DIM, &win_size, 0);
  // window title = name of program
  PtSetArg(&args[1],Pt_ARG_WINDOW_TITLE,
           (long)"Outline Test", 0);
  pwndMain = PtCreateWidget (PtWindow, Pt_NO_PARENT,
                              2, args);

  if(PfGenerateFontName(DESC_FONT, 0, 36,
                       szFont) == NULL)
    return(Pt_CONTINUE);
PfGetOutline(), PfGetOutlineCx()
\textbf{PfGetOutline(), PfGetOutlineCx()}

\begin{verbatim}
s_lAdvanceX += (-tsMetrics.BearingX + 0xFFFFL) >> 16;

DrawOutline(pnt, lNumContours, &rect, lAscender);
free(pnt);
free(loops);

s_lAdvanceX += (tsMetrics.Advance + 0xFFFFL) >> 16;

if((lNumContours = PfGetOutline(szFont, 'o', &tsMetrics, &pnt, &loops)) == -1L)
    return(Pt_CONTINUE);

DrawOutline(pnt, lNumContours, &rect, lAscender);
free(pnt);
free(loops);

s_lAdvanceX += (tsMetrics.Advance + 0xFFFFL) >> 16;

if((lNumContours = PfGetOutline(szFont, 'u', &tsMetrics, &pnt, &loops)) == -1L)
    return(Pt_CONTINUE);

DrawOutline(pnt, lNumContours, &rect, lAscender);
free(pnt);
free(loops);

s_lAdvanceX += (tsMetrics.Advance + 0xFFFFL) >> 16;

if((lNumContours = PfGetOutline(szFont, 0x5EB3, &tsMetrics, &pnt, &loops)) == -1L)
    { printf("return failed\n");
      return(Pt_CONTINUE);
    }

DrawOutline(pnt, lNumContours, &rect, lAscender);
\end{verbatim}
PfGetOutline(), PfGetOutlineCx()

```c
free(pnt);
free(loops);

s_lAdvanceX += (tsMetrics.Advance + 0xFFFFL) >> 16;

if((lNumContours = PfGetOutline(szFont, 'A', &tsMetrics, &pnt, &loops)) == -1L)
{
    printf("return failed\n");
    return(Pt_CONTINUE);
}

DrawOutline(pnt, lNumContours, &rect, lAscender);

free(pnt);
free(loops);

PgSetStrokeColor(old);

return( Pt_CONTINUE );
}

int DrawOutline( PhPoint_t * pnt, long ContourCount,
                 PhRect_t * rect, long lAscender)
{
    unsigned long ul2 = 1L, ul1 = 0L;
    long ii = 0L, jj = 0L;
    PhPoint_t pos = { s_lAdvanceX + rect->ul.x,
                     (rect->lr.y - lAscender) };
    int offset = 0;
    PgColor_t old = PgSetFillColor(Pg_BLACK);

    for(ii = 0L; ii < ContourCount; ii++)
    {
        if(!bDrawLine)
        { printf("PgDrawPolygon()\n");

            PgDrawPolygon(pnt + offset, loops[ii],
                          &pos, Pg_DRAW_STROKE);

            offset += loops[ii];
        }
        else if(bDrawLine)
        { printf("PgDrawLine()\n");

            for(jj = 0; jj < loops[ii] - 1; jj++)
            {
                PgDrawILine(pos.x + pnt[ul1].x,
                             pos.y + pnt[ul1].y,
```
**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PfAttachCx(), PfDetach(), PfDetachCx(), PfGlyph(), PfGlyphCx(), PfGenerateFontName(), PfGenerateFontNameCx(), PhPoint_t

Fonts chapter of the Photon *Programmer's Guide*
PfGlyph(), PfGlyphCx()
Obtain the metrics and/or bitmap for the specified character

Synopsis:
#include <photon/Pf.h>

int PfGlyph( const char *font,
     long symbol,
     FontRender *metrics,
     unsigned char *bitmap,
     int size,
     FontName fontused);

#include <font_api.h>
int PfGlyphCx( struct _Pf_ctrl *context,
     const char *font,
     long symbol,
     FontRender *metrics,
     unsigned char *bitmap,
     int size,
     FontName fontused);

Arguments:

context    (PfGlyphCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
font       The base font to use, which you should create by calling PfGenerateFontName() or PfGenerateFontNameCx().
symbol     The character that you’re interested in.
metrics    NULL, or a pointer to a FontRender structure, which the function fills in with the character metrics (see PfRender()).
bitmap     NULL, or a pointer to a block of memory where the function can store the character bitmap.
size       The number of bytes in the block that bitmap points to.
fontused    NULL, or a buffer where the function can store the name of the font that supplies the character.
PfGlyph(), PfGlyphCx()

Library:

PfGlyph()  ph
PfGlyphCx()  font

Description:
These routines are useful for obtaining arbitrary character glyphs, such as cursors. The functions obtain from the base font the metrics and/or bitmap for the character specified by symbol.

When bitmap is non-NULL, it must point to an area of size bytes that the character bitmap can be placed in. It may be rendered as a bitmap/image in conjunction with the metrics information. The actual font used to supply the character is placed in the string pointed to by fontused if non-NULL.

Returns:

0  Success.
-1  An error occurred (errno is set).

Examples:

PfGlyphCx(): See the example for PfGetGlyphIndexCx().

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PfGlyph(), PfGlyphCx()

See also:

PfGenerateFontName(), PfGenerateFontNameCx(), PfGetOutline(), PfGetOutlineCx(), PfRender(), PfRenderCx()

Fonts chapter of the Photon Programmer’s Guide
**PfLoadFont(), PfLoadFontCx()**

Preload a font within the font server

**Synopsis:**

```c
#include <photon/Pf.h>
int PfLoadFont( const char *font,
                unsigned flags,
                FontName fontused );

#include <font_api.h>
int PfLoadFontCx( struct _Pf_ctrl *context,
                  const char *font,
                  unsigned flags,
                  FontName fontused );
```

**Arguments:**

- `context` *(PfLoadFontCx() only)* A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `font` The name of the desired font, as created by `PfGenerateFontName()` or `PfGenerateFontNameCx()`.
- `flags` Any combination of:
  - `PHFONT_LOAD_METRICS` — load the font metrics.
  - `PHFONT_LOAD_IMAGES` — load the font bitmaps.

If you don’t specify any flags, the function checks the validity of the input font name.

- `fontused` NULL, or a buffer where the function can store the real name of the font, which may be different from the input if the font is unknown or is mapped as an alias to another font (via the `fontmap` file).

**Library:**

- `PfLoadFont()`: `ph`
- `PfLoadFontCx()`: `font`

Chapter 7 • Pf—Font Server  391
**PfLoadFont(), PfLoadFontCx()**

**Description:**
This function preloads a font (from disk into memory) within the font server to speed up subsequent use of the font. By default, a font is loaded only when required by *PgExtentText()* or *PgDrawText()*.

**Returns:**

0  Success.

-1  An error occurred (*errno* is set).

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PfGenerateFontName(), PfGenerateFontNameCx(), PgDrawText(), PgExtentText()*

Fonts chapter of the Photon *Programmer’s Guide*
**PfLoadMetrics(), PfLoadMetricsCx()**

Load metric information for the given font

**Synopsis:**

```c
#include <photon/Pf.h>
int PfLoadMetrics( const char *font );

#include <font_api.h>
int PfLoadMetricsCx( struct _Pf_ctrl *context,
                     const char *font );
```

**Arguments:**

- `context` (PfLoadMetricsCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `font` The name of the font, as created by PfGenerateFontName() PfGenerateFontNameCx() or.

**Library:**

- `PfLoadMetrics()`
  - `ph`

- `PfLoadMetricsCx()`
  - `font`

**Description:**

These functions load metric information for the given font from the font server into memory and link this font into a list of available local metrics.

Subsequent text extents of this base font, involving characters solely within this font, are performed locally by the task itself rather than by the font server. This may result in faster operation of extent-intensive tasks, such as HTML viewers, at a cost of about 1400 bytes of memory per font (for a standard font that defines characters 0x20-0xFF).
In instances where the font metrics do not contain the glyph, messaging will be used as a fallback method.

Returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred (errno is set)</td>
</tr>
</tbody>
</table>

Examples:

`PfLoadMetricsCx()`: See the example for `PfConvertPixelsToPointSizeCx()`.  

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt handler</td>
</tr>
<tr>
<td></td>
<td>Signal handler</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PfGenerateFontName()`, `PfGenerateFontNameCx()`,  
`PfUnloadMetrics()`, `PfUnloadMetricsCx()`  
Fonts chapter of the Photon *Programmer’s Guide*
**PfQueryFontInfo(), PfQueryFontInfoCx()**

Get information about a font

**Synopsis:**

```c
#include <photon/Pf.h>
int PfQueryFontInfo( const char *font,
                     FontQueryInfo *info );

#include <font_api.h>
int PfQueryFontInfoCx( struct _Pf_ctrl *context,
                       const char *font,
                       FontQueryInfo *info );
```

**Arguments:**

- `context` (PfQueryFontInfoCx() only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `font` The name of the font, as created by PfGenerateFontName().
- `info` A pointer to a FontQueryInfo structure that the function fills with information about the font; see below.

**Library:**

- `PfQueryFontInfo()`
  - `ph`
- `PfQueryFontInfoCx()`
  - `font`

**Description:**

These functions get information about the font specified by `font` after first mapping `font` to a valid font name (if appropriate).

The FontQueryInfo structure pointed to by `info` is filled in. It contains at least:

- `FontName font` Internal name of the font (e.g. TextFont09).
**PfQueryFontInfo**, **PfQueryFontInfoCx**

FontDescription *desc*

Textual name of the font family (e.g. **Helvetica**).

**short** *size*

Point size of the font, or 0 for a scalable font.

**unsigned short** *style*

Style and attributes of this font, made up of the following bits:

- **PHFONT_INFO_ALIAS** — the entry is a mapping or virtual font, like **TextFont**.
- **PHFONT_INFO_BLDITC** — bold italic style.
- **PHFONT_INFO_BOLD** — bold style.
- **PHFONT_INFO_DECORATIVE** — decorative style.
- **PHFONT_INFO_FIXED** — fixed-width font.
- **PHFONT_INFOITALIC** — italic style.
- **PHFONT_INFOPLAIN** — plain/regular style.
- **PHFONT_INFOPROP** — proportional-width font.
- **PHFONT_INFOSANSERIF** — sans-serif font.
- **PHFONT_INFOSERIF** — serif font.

**short** *ascender*

Ascender value of the font (in pixels).

**short** *descender*

Descender value of the font (in pixels).

**short** *width*

Width of widest character in this font.

**long** *lochar*

Lowest character value defined in this font.

**long** *hichar*

Highest character value defined in this font.
PfQueryFontInfo(), PfQueryFontInfoCx()

Returns:

0    Successful completion.
-1   An error occurred (errno is set).

Examples:

PfQueryFontInfoCx(): See the example for PfGetGlyphIndexCx().

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfGenerateFontName(), PfGenerateFontNameCx(), PfQueryFonts(), PfQueryFontsCx()

Fonts chapter of the Photon Programmer’s Guide
**PfQueryFonts(), PfQueryFontsCx()**

**Construct a list of installed fonts**

**Synopsis:**

```c
#include <photon/Pf.h>
int PfQueryFonts( long symbol,
                 unsigned flags,
                 FontDetails list[],
                 int n);

#include <font_api.h>
int PfQueryFontsCx( struct _Pf_ctrl *context,
                   long symbol,
                   unsigned flags,
                   FontDetails list[],
                   int n);
```

**Arguments:**

- `context` (PfQueryFontsCx only) A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().
- `symbol` The symbol that you want to be able to render, or PHFONT_ALL_SYMBOLS if you don’t want to limit the search to fonts that support a specific character.
- `flags` Flags that you can use to filter the list of fonts; any combination of the following:
  - PHFONT_SCALABLE — select scalable fonts.
  - PHFONT_BITMAP — select bitmapped fonts.
  - PHFONT_PROP — select proportional fonts.
  - PHFONT_FIXED — select fixed-width fonts.
  - PHFONT_ALL_FONTS — select all fonts.
  - PHFONT_DONT_SHOW_LEGACY — exclude legacy fonts. This flag overrides PHFONT_ALL_FONTS.
- `list` NULL, or an array of `FontDetails` structures where the function can store information about the matching fonts; see below.
- `n` 0, or the number of elements in the list.
PfQueryFonts(), PfQueryFontsCx()

Library:

PfQueryFonts()

\( ph \)

PfQueryFontsCx()

\( font \)

Description:

These functions construct a list of all fonts that may be used to render the character specified by the symbol parameter. For example, use ‘A’ to get a list of normal/Latin fonts, or 0x3A9 (omega) to get a list of Greek fonts. (See PkKeyDef.h or ISO/EIC 10646-1 for a list of symbols.)

Up to \( n \) matching font family entries are placed in the user-provided list.

If \( n \) is 0 and list is NULL, these functions return the number of matching fonts but don’t try to fill in the list. You can use this feature to determine the number of items to allocate for the list.

FontDetails structure

The entries in the list are of type FontDetails, and contain the following fields:

FontDescription desc

Textual name of the font family (e.g. Helvetica).

FontName stem

Base stem of the font family (e.g. helv).

short losize

Lowest point size available for this font. If losize and hisize are both 0, the font is scalable.

short hisize

Highest point size available for this font. If hisize and losize are both 0, the font is scalable.
PfQueryFonts(), PfQueryFontsCx()

**unsigned short flags**

Various stylistic/attribute flags for this font family:
- PHFONT_INFO_ALIAS — the entry is a mapping or virtual font, like TextFont.
- PHFONT_INFO_BLDITC — bold italic style.
- PHFONT_INFO_BOLD — bold style.
- PHFONT_INFO_DECORATIVE — decorative style.
- PHFONT_INFO_FIXED — fixed-width font.
- PHFONT_INFO_ITALIC — italic style.
- PHFONT_INFO_PLAIN — plain/regular style.
- PHFONT_INFO_PROP — proportional-width font.
- PHFONT_INFO_SAN Serif — sans-serif font.
- PHFONT_INFO_SERIF — serif font.

Use PfGenerateFontName() or PfGenerateFontNameCx() and the information in the FontDetails structure to build a font name that you can pass to functions such as PgSetFont().

**Returns:**

The number of matching fonts found, or -1 on error.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
PfQueryFonts(), PfQueryFontsCx()

See also:
PfGenerateFontName(), PfGenerateFontNameCx(),
PfQueryFontInfo(), PfQueryFontInfoCx(), PgSetFont()
PtFontSel (in the Photon Widget Reference)
Fonts chapter of the Photon Programmer’s Guide
**PfRender()**, **PfRenderCx()**

Render a string via a user callback function

**Synopsis:**

```c
#include <photon/Pf.h>
int PfRender( void *ctx,
    const char *font,
    long adata,
    long bdata,
    const char *str,
    int len,
    int flags,
    PhPoint_t const *pos,
    PhRect_t const *clip,
    void (*func)(
        void *, const PhPoint_t *pos,
        const FontRender *)));
```

```c
#include <font_api.h>
int PfRenderCx( struct _Pf_ctrl *context,
    void *ctx,
    const char *font,
    long adata,
    long bdata,
    const char *str,
    int len,
    int flags,
    PhPoint_t const *pos,
    PhRect_t const *clip,
    void (*func)(
        void *, const PhPoint_t *pos,
        const FontRender *)));
```

**Arguments:**

- **context** (PfRenderCx() only)
  A pointer to the font context to use, returned by PfAttachCx() or PfAttachDllCx().

- **ctx**
  A context/data pointer that’s passed to the user callback, *func*. This value must be nonzero; if you don’t want to pass a context, set this argument to 1.
PfRender(), PfRenderCx()

font The base font, which you should create by calling PfGenerateFontName() or PfGenerateFontNameCx().

adata The horizontal fractional point size, if you set PF_FRACTIONAL in the flags argument.

bdata The vertical fractional point size, if you set PF_FRACTIONAL in the flags argument.

str The string whose extent you want to calculate. The string is a UTF-8 multibyte one by default.

len The length of the string, str, in bytes. If len is 0, the function uses strlen(str).

flags Flags that affect the behavior of the function.

You can set up to one of the following to indicate the format of the string:

- PF_WIDE_CHARS — the string is composed of 16-bit wide characters. If you set this flag, the function assumes that each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.

- PF_WIDE_CHAR32 — the string is composed of 32-bit wide characters. If you set this flag, the function assumes that each character is represented by 4 bytes that conform to the ISO/IEC 10646-1 UCS-4 four-byte format.

Although this flag allows for 32-bit wide characters, the underlying font system currently supports only characters up to Unicode U+FFE.

If you don’t set either of the above, the function assumes that the string is composed of UTF-8 multibyte characters.

You can OR in:

- PF_FRACTIONAL — use a fractional 16.16 point size. If you set this flag, use the adata argument to specify the
PfRender(), PfRenderCx()

horizontal fractional point size, and bdata to specify the vertical fractional point size.

pos A pointer to a PhPoint_t structure that specifies the location at which to render the text.

clip A pointer to a PhRect_t that specifies the clipping rectangle for the text. If clip is NULL, it’s ignored. The font server performs coarse character clipping only.

func A user callback function that’s called to render the text. It’s called with the desired pen location and the metrics of the bitmap. If the entire bitmap doesn’t fit in the allocated memory, multiple calls to the font server and the user function can be made, advancing the pen as appropriate between calls.

Library:

PfRender() ph
PfRenderCx() font

Description:

These functions render the given string via a user callback function. The difference between PfRender() and PfRenderCx() is that PfRenderCx() lets you specify the font context to use.

When a request to construct the bitmap is sent to the font server, the string bitmap is returned, for efficiency, in the shared-memory area created through the initial call to PfAttach(), PfAttachCx() or PfAttachDllCx().

Normally, only the graphics drivers use these functions, but they may be useful for application programs that have to obtain text bitmap data directly.

The generic design of these routines allows future expansion.

The FontRender metrics structure contains at least the following members:
PfRender(), PfRenderCx()

**PhPoint_t size**

The bounding size of the bitmap, in pixels.

**PhPoint_t offset**

The offset of the bitmap (the upper-left of the extent).

**int width**

The width of the bitmap.

**short bpl**

The number of bytes per line.

**short bpp**

The number of bits per pixel (1 for normal output, 4 for anti-aliased).

**unsigned char *bmptr**

A pointer to the bitmap data (stored row-wise).

**Returns:**

0 Success.

-1 An error occurred; errno is set.

**Examples:**

*PfRenderCx():*

/* Render alignment example. Demonstrates how to achieve 8-bit, */
/* 32-bit, and 64-bit aligned render maps if supported by the */
/* particular font server. */
#include <font_api.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
static int check_32 = 0;
static int bad_32 = 0;
static int check_64 = 0;
static int bad_64 = 0;
static void func(void *ctx, const pf_point_t *pos, const FontRender *render)
{
    printf("NOTE: render callback.\n");
    if(check_32)
    { int times = render->bpl / 4;
        int total = times * 4;
        ...
PfRender(), PfrEndRenderCx()
PfDetachCx(pf);

if((pf = PfAttachCx("/dev/phfont64", -1)) != NULL)
{
  fprintf(stderr,
    "NOTE: render 64-bit aligned image.\n"");
  check_64 = 1;

  if(PfRenderCx(pf, pf, PfConvertFontIDCx(pf, id, tag), 0L, 0L, "TEST.", 0, 0,
    &pos, NULL, func) == 0)
  {
    if(bad_64)
      { fprintf(stderr,
        "NOTE: PfRenderCx did not render a 64-bit aligned image.\n"");
    fprintf(stderr,
      "FAIL: PfRenderCx and bitmap alignment.\n"");
    } else
      { fprintf(stderr,
        "PASS: PfRenderCx and bitmap alignment.\n"");
    } else
      { fprintf(stderr,
        "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
    } else
      { fprintf(stderr,
        "UNRES: Unable to attach to fontserver, \n" errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr,
        "UNRES: Unable to attach to fontserver, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      } else
      { fprintf(stderr, "NOTE: PfRenderCx failed, errno %d.\n", errno);
      fprintf(stderr,
        "FAIL: PfRenderCx and bitmap alignment.\n"");
      }
  }
}

Chapter 7 • Pf—Font Server  407
PfRender(), PfRenderCx()

else
  { fprintf(stderr, "UNRES: PfAttachCx failed, errno %d.\n", errno);
    fprintf(stderr, "FAIL: PfRenderCx and bitmap alignment.\n");
  }
}
else
  { fprintf(stderr,
    "NOTE: PfRenderCx returned success with invalid render buffer.\n");
    fprintf(stderr, "FAIL: PfRenderCx and bitmap alignment.\n");
  }

if(PfFreeFontCx(pf, id) == -1L)
  { fprintf(stderr, "NOTE: PfFreeFontCx failed, errno %d.\n", errno);
  }
else
  { fprintf(stderr, "NOTE: PfFindFontCx failed to create font id, errno %d.\n", errno);
    fprintf(stderr, "FAIL: PfRenderCx and bitmap alignment.\n");
  }

if(!skip)
  PfDetachCx(pf);
else
  { fprintf(stderr, "UNRES: Unable to attach to fontserver, errno %d.\n", errno);
    fprintf(stderr, "FAIL: PfRenderCx and bitmap alignment.\n");
  }

return(0);

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PfRender(), PfRenderCx()

See also:
PfAttach(), PfAttachCx(), PfDetach(), PfDetachCx(), PfGenerateFontName(), PfGenerateFontNameCx(), PhPoint_t, PhRect_t
PfRestartServerDll()

Restart a local server

Synopsis:

```c
#include <font_api.h>
int PfRestartServerDll( fontdll_t dll );
```

Arguments:

dll   A font server context, returned by PfAttachLocalDll().

Library:

font

Description:

This function restarts the local server associated with the provided dll context. All font DLL settings are retained during the restart.

Returns:

0   Success
-1   An error occurred (errno is set).

Examples:

See the example for PfAttachLocalDll().

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>
PfRestartServerDll()

See also:

\textit{PfAttachLocalDll}. 

Fonts chapter of the Photon \textit{Programmer's Guide}
PfSetOptionsDll()
Set active font DLL context options

Synopsis:
#include <font_api.h>
int PfSetOptionsDll( fontdll_t dll,
                    char const * options,
                    char const * schema );

Arguments:
dll A font DLL context, returned by PfAttachLocalDll().
options A pointer to a string that contains the options you want to
set for the font DLL. The options must be separated by
commas, for example,
-A,-d=/usr/photon/font_repository. This string
may be NULL.
schema The name of a schema, a configuration file used to
override the default settings for a local font server.
Use the DLL_FONT_SERVER schema for processes that
need to allocate sufficient resources to act as a default font
server. This schema loads a local server instance that can
be used as an external server by other applications (that is,
it appears in /def/phfont). Use NULL to load a local
server instance that cannot be used by other applications.
The string referenced by schema may not exceed
DLL_MAX_OPTION_NAME bytes, including the
terminating NULL.

Library:
font

Description:
This function applies options to an instantiated DLL context. Since
this font instance is active, all options changes may not be permitted.
Legal options are identical to those support by the font server. Some
options may not be relevant to a DLL instance of the font server. If
this function is invoked on a DLL context, with an active font instance, it is up to the font instance as to whether or not an option should be applied.

Returns:

0 Success

-1 An error occurred (errno is set).

Errors:

EINVAL Invalid options.

ENOMEM Insufficient resources to process options.

Examples:

See the example for PfAllocRenderCx().

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Cancellation point</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

PfAttachLocalDll()

The Fonts chapter of the Photon Programmer’s Guide
**PfSetRenderingDPICx()**

**Set rendering DPI**

**Synopsis:**

```c
int PfSetRenderingDPICx( struct _Pf_ctrl * context,
                         uint32_t res_x,
                         uint32_t res_y );
```

**Arguments:**

- `context` A pointer to the font context to use, returned by `PfAttachCx()` or `PfAttachDllCx()`.
- `res_x` The horizontal DPI value you want to set the context rendering to.
- `res_y` The vertical DPI value you want to set the context rendering to.

**Library:**

`font`

**Description:**

This function requests that the font server set the rendering and extending DPI for the provided `context`. The font server may ignore the request completely, or a particular font technology may not be able to accommodate the request.

**Returns:**

- 0 Success
- -1 An error occurred (`errno` is set).

**Errors:**

- EFAULT Font context is NULL.
Examples:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <font_api.h>

static int compare_extent_is_same(pf_rect_t const * e, pf_rect_t const * e2)
{ if(e->ul.x != e2->ul.x)
    return(0);

  if(e->ul.y != e2->ul.y)
    return(0);

  if(e->lr.x != e2->lr.x)
    return(0);

  if(e->lr.y != e2->lr.y)
    return(0);

  return(1);
}

static int compare_render_is_same(FontRender const * r, FontRender const * r2)
{ if(r->size.x != r2->size.x)
    return(0);

  if(r->size.y != r2->size.y)
    return(0);

  if(r->offset.x != r2->offset.x)
    return(0);

  if(r->offset.y != r2->offset.y)
    return(0);

  if(r->width != r2->width)
    return(0);

  if(r->bpl != r2->bpl)
    return(0);

  if(r->bpp != r2->bpp)
    return(0);

  if(r->flags != r2->flags)
    return(0);

  return(1);
}

static void func(void * ctx, const pf_point_t * pos, const FontRender * render)
{ FontRender * r = (FontRender *)ctx;
  (*r) = *render;
}

int main (int argc, char *argv[])
{ struct _Pf_ctrl * pf;
```
PfSetRenderingDPICx()

fprintf(stderr, "POINT: DPI.\n");

if((pf = PfAttachCx(NULL, 32000)) == NULL)
    { fprintf(stderr, "NOTE: PfAttachCx failed, errno %d.\n", errno);
      fprintf(stderr, "FAIL: DPI.\n");
      exit(EXIT_FAILURE);
    }
else
    { FontName tag;

      if(PfGenerateFontNameCx(pf, "PrimaSans BT", 0L, 12L, tag) != NULL)
      { char const * text = "Hello Bobby!!";
        pf_rect_t extent;

        if(PfExtentCx(pf, &extent, NULL, tag, 0L, 0, text, 0, 0, NULL) == 0)
          { if(PfSetRenderingDPICx(pf, 72, 72) == 0)
            { pf_rect_t extent2;

              if(PfExtentCx(pf, &extent2, NULL, tag, 0L, 0, text, 0, 0, NULL) == 0)
                { if(!compare_extent_is_same(&extent, &extent2))
                  { pf_point_t pos = { 0, 0 };
                    fprintf(stderr,
"NOTE: extent ulx %hd, uly %hd, lrx %hd, lry %hd\n",
extent.ul.x, extent.ul.y, extent.lr.x, extent.lr.y);
                    fprintf(stderr,
"NOTE: extent2 ulx %hd, uly %hd, lrx %hd, lry %hd\n",
extent2.ul.x, extent2.ul.y, extent2.lr.x, extent2.lr.y);

                    if(PfSetRenderingDPICx(pf, 0, 0) == 0)
                      { FontRender render;

                        if(PfRenderCx(pf, &render, tag, 0L, 0L, text, 0, 0,
apos, NULL, func) == 0)
                          { if(PfSetRenderingDPICx(pf, 72, 72) == 0)
                            { FontRender render2;

                              if(PfRenderCx(pf, &render2, tag, 0L, 0L, text,
0, 0, apos, NULL, func) == 0)
                                { if(!compare_render_is_same(&render, &render2))
                                  { fprintf(stderr,
"NOTE: render sizex %hd, sizey %hd\n",
render.size.x, render.size.y);
                                  fprintf(stderr,
"NOTE: render2 sizex %hd, sizey %hd\n",
render2.size.x, render2.size.y);
                                }
                              }
                          }
                    }
                  }
              }
            }
          }
      }

      if(PfGenerateFontNameCx(pf, "PrimaSans BT", 0L, 12L, tag) != NULL)
      { char const * text = "Hello Bobby!!";
        pf_rect_t extent;

        if(PfExtentCx(pf, &extent, NULL, tag, 0L, 0, text, 0, 0, NULL) == 0)
          { if(PfSetRenderingDPICx(pf, 72, 72) == 0)
            { pf_rect_t extent2;

              if(PfExtentCx(pf, &extent2, NULL, tag, 0L, 0, text, 0, 0, NULL) == 0)
                { if(!compare_extent_is_same(&extent, &extent2))
                  { pf_point_t pos = { 0, 0 };
                    fprintf(stderr,
"NOTE: extent ulx %hd, uly %hd, lrx %hd, lry %hd\n",
extent.ul.x, extent.ul.y, extent.lr.x, extent.lr.y);
                    fprintf(stderr,
"NOTE: extent2 ulx %hd, uly %hd, lrx %hd, lry %hd\n",
extent2.ul.x, extent2.ul.y, extent2.lr.x, extent2.lr.y);

                    if(PfSetRenderingDPICx(pf, 0, 0) == 0)
                      { FontRender render;

                        if(PfRenderCx(pf, &render, tag, 0L, 0L, text, 0, 0,
apos, NULL, func) == 0)
                          { if(PfSetRenderingDPICx(pf, 72, 72) == 0)
                            { FontRender render2;

                              if(PfRenderCx(pf, &render2, tag, 0L, 0L, text,
0, 0, apos, NULL, func) == 0)
                                { if(!compare_render_is_same(&render, &render2))
                                  { fprintf(stderr,
"NOTE: render sizex %hd, sizey %hd\n",
render.size.x, render.size.y);
                                  fprintf(stderr,
"NOTE: render2 sizex %hd, sizey %hd\n",
render2.size.x, render2.size.y);
                                }
                              }
                          }
                      }
                }
              }
            }
          }
      }

fprintf(stderr, "NOTE: PfAttachCx failed, errno %d.\n", errno);
fprintf(stderr, "FAIL: DPI.\n"); exit(EXIT_FAILURE);
}
PfSetRenderingDPICx()

exit(EXIT_FAILURE);

} else{
  fprintf(stderr,
    "NOTE: PfSetRenderingDPICx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

} else{
  fprintf(stderr,
    "NOTE: PfRenderCx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

} else{
  fprintf(stderr,
    "NOTE: PfSetRenderingDPICx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

} else{
  fprintf(stderr,
    "NOTE: PfExtentCx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

} else{
  fprintf(stderr,
    "NOTE: PfGenerateFontNameCx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

} else{
  fprintf(stderr,
    "NOTE: PfExtentCx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

} else{
  fprintf(stderr,
    "NOTE: PfGenerateFontNameCx failed, errno %d.\n",
    errno);
  fprintf(stderr, "FAIL: DPI.\n");
  exit(EXIT_FAILURE);
}

PFDetachCx(pf);

Chapter 7 • Pf—Font Server 417
PfSetRenderingDPICx()

```c
fprintf(stderr, "PASS: DPI.\n\n");  
return(0);  
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

PfAttachCx(), PfAttachDllCx().

Fonts chapter of the Photon *Programmer’s Guide*
PfTextWidthBytes()

Calculate the width of a char string of multibyte UTF-8 characters

Synopsis:

```c
#include <photon/Pf.h>
int PfTextWidthBytes( const char *font,
                      const char *str,
                      int len );
```

Arguments:

- `font` The name of the desired font. You should use PfGenerateFontName() to create this name.
- `str` A char string of multibyte UTF-8 characters.
- `len` The length of the string, in bytes. If `len` is 0, strlen(str) is assumed.

Library:

ph

Description:

PfTextWidthBytes() is a convenience function that calculates the width of the given string in the given font, using the formula:

```
extent.lr.x - min(extent.ul.x, 0) + 1
```

PfTextWidthChars() is similar, but you give it the number of characters in the string rather than the number of bytes.

Returns:

The width of the string, or 0 if an error occurred.

Examples:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ap.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>
```
PfTextWidthBytes()

```c
int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage );

int main(int argc, char *argv[])
{
    PtArg_t args[8];
    PhPoint_t win_size, pntPOS, pntDIM;
    short nArgs = 0;
    PtWidget_t * pwndMain = NULL, * pobjRaw = NULL;

    PtInit (NULL);

    // set base pwndMain parms
    win_size.x = 450;
    win_size.y = 600;

    PtSetArg(&args[0], Pt_ARG_DIM, &win_size, 0);
    // window title = name of program
    PtSetArg(&args[1], Pt_ARG_WINDOW_TITLE, "PfTextWidth", 0);

    pwndMain = PtCreateWidget (PtWindow, Pt_NO_PARENT, 2, args);

    pntPOS.y = 100;
    pntPOS.x = 75;
    pntDIM.x = 300;
    pntDIM.y = 300;

    PtSetArg(&args[0], Pt_ARG_POS, &pntPOS, 0);
    PtSetArg(&args[1], Pt_ARG_DIM, &pntDIM, 0);
    PtSetArg(&args[2], Pt_ARG_RAW_DRAW_F, fnDrawCanvas, 0L);
    pobjRaw = PtCreateWidget(PtRaw, pwndMain, 3, args);

    (void) PtRealizeWidget(pwndMain);

    PtMainLoop ();

    return(0);
}

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage )
{
    PhRect_t rect;
    PhPoint_t tsPos = {0, 0};
    PgColor_t old;
    int iLen = 0;
    char szHelvetica12[MAX_FONT_TAG];

    // find our canvas
    PtCalcCanvas(ptsWidget, &rect);

    PtSuperClassDraw(PtBasic, ptsWidget, ptsDamage);
}
```

Chapter 7 • Pf—Font Server
old = PgSetStrokeColor(Pg_BLACK);

// draw text
tsPos.x = 10 + rect.ul.x;
tsPos.y = 10 + rect.ul.y;
PgSetFont(PfGenerateFontName("Helvetica", 0, 12,
   szHelvetica12));
PgSetTextColor(Pg_BLACK);
PgDrawText("Hello", 5, &tsPos, 0);

if((iLen = PfTextWidthBytes(szHelvetica12,
   "Hello", 0)) == 0)
   return(Pt_CONTINUE);
PgDrawILine(tsPos.x, tsPos.y, tsPos.x + iLen, tsPos.y);
PgSetStrokeColor(old);
return( Pt_CONTINUE );
}

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfGenerateFontName(), PfTextWidthChars()

Fonts chapter of the Photon Programmer’s Guide
**PfTextWidthChars()**

*Calculate the width of a char string of multibyte UTF-8 characters*

**Synopsis:**

```c
#include <photon/Pf.h>
int PfTextWidthChars( const char *font,
                        const char *str,
                        int len );
```

**Arguments:**

- `font` The name of the desired font. You should use `PfGenerateFontName()` to create this name.
- `str` A `char` string of multibyte UTF-8 characters.
- `len` The number of characters in the string. If `len` is 0, `strlen(str)` is assumed.

**Library:**

`ph`

**Description:**

`PfTextWidthChars()` is a convenience function that calculates the width of the given string in the given font, using the formula:

\[ \text{extent.lr.x} - \min(\text{extent.ul.x, 0}) + 1 \]

`PfTextWidthBytes()` is similar, but you pass it the number of bytes in the string, not the number of characters.

**Returns:**

The width of the string, or 0 if an error occurred.

**Classification:**

Photon
PfTextWidthChars()

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfGenerateFontName(), PfTextWidthBytes()

Fonts chapter of the Photon Programmer’s Guide
**PfUnloadMetrics()**

Unload metric information for the given font

**Synopsis:**
```c
#include <photon/Pf.h>
int PfUnloadMetrics( const char *font );
```

**Arguments:**

- `font` The name of the font whose metrics you want to unload. You should create this name by calling `PfGenerateFontName()`.

**Library:**

- `ph`

**Description:**

This function unloads the local metrics for the given font from memory and releases any memory used for their storage. Subsequent text extents of this base font are resolved by the font server rather than being performed locally.

**Returns:**

- `0` Success.
- `-1` An error occurred (`errno` is set).

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PfUnloadMetrics()

See also:

PfLoadMetrics(), PfGenerateFontName()

Fonts chapter of the Photon Programmer’s Guide
**PfWaitOnServerDll()**

*Wait on server*

**Synopsis:**

```c
#include <font_api.h>
int PfWaitOnServerDll( fontdll_t dll );
```

**Arguments:**

`dll` A font DLL context, returned by `PfAttachLocalDll()`.

**Library:**

`font`

**Description:**

This function waits until the font server `dll` thread exits.

**Returns:**

- 0 Success
- -1 An error occurred (`errno` is set).

**Examples:**

See the example for `PfAttachLocalDll()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

426 Chapter 7 • Pf—Font Server
PfWaitOnServerDll()

See also:

PfAttachLocalDll().

Fonts chapter of the Photon Programmer’s Guide
**PfWideTextWidthBytes()**

*Calculate the width of a uint16_t string of Unicode characters*

**Synopsis:**

```c
#include <photon/Pf.h>
int PfWideTextWidthBytes( const char *font,
                           const uint16_t *str,
                           int len );
```

**Arguments:**

- `font`: The name of the desired font. Use `PfGenerateFontName()` to create the name.
- `str`: A `uint16_t` string of Unicode characters.
- `len`: The length of the string, in bytes. If `len` is 0, `strlen(str)` is assumed.

**Library:**

`ph`

**Description:**

`PfWideTextWidthBytes()` is a convenience function that calculates the width of the given string in the given font, using the formula:

```
extent.lr.x - min(extent.ul.x, 0) + 1
```

This function assumes each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.

`PfWideTextWidthChars()` is similar, but you give it the number of characters in the string instead of the number of bytes.

**Returns:**

The width of the string, or 0 if an error occurred.
**PfWideTextWidthBytes()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfGenerateFontName(), PfTextWidthBytes(), PfTextWidthChars(), PfWideTextWidthChars()*

Fonts chapter of the Photon *Programmer's Guide*
**PfWideTextWidthChars()**

*Calculate the width of a `uint16_t` string of Unicode characters*

**Synopsis:**

```c
#include <photon/Pf.h>
int PfWideTextWidthChars( const char *font,
        const uint16_t *str,
        int len );
```

**Arguments:**

- `font` The name of the desired font. Use `PfGenerateFontName()` to create the name.
- `str` A `uint16_t` string of Unicode characters.
- `len` The number of characters in the string. If `len` is 0, `strlen(str)` is assumed.

**Library:**

`ph`

**Description:**

`PfWideTextWidthChars()` is a convenience function that calculates the width of the given string in the given font, using the formula:

```
extent.lr.x - min(extent.ul.x, 0) + 1
```

This function assumes each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.

`PfWideTextWidthBytes()` is similar, but you pass it the number of bytes in the string, rather than the number of characters.

**Returns:**

The width of the string, or 0 if an error occurred.
**PfWideTextWidthChars()**

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfTextWidthBytes() PfTextWidthChars() PfWideTextWidthBytes()*

Fonts chapter of the Photon *Programmer’s Guide*
Most Photon graphics functions append draw commands to a buffer. The application sends these commands to the Photon Manager, which in turn sends them to the graphics driver. The graphics driver then renders the commands on the screen.

Photon supports a set of simple, dual-purpose draw primitives that you can stroke (that is, draw as an outline) or fill, or both. These primitives include arcs, ellipses, polygons, rectangles, and rounded rectangles.

For fast animation, you can use the $Pg...$ and $Pd...$ functions.

If you’re building a GUI, you should use widgets whenever possible instead of calling the $Pg*$ functions directly. Widgets handle interaction with the user and look after redrawing themselves when damaged.

If you need to do raw drawing in an application that uses widgets, create a $PtRaw$ widget, and call the drawing primitives in its draw function. For more information, see the Raw Drawing and Animation chapter of the Photon Programmer’s Guide.

Many of the $Pg...( )$ functions have multiple versions, distinguished by their suffix:

- $Pg*( )$ — standard function, which operates on the current draw or graphics context. It is implemented as a macro for its corresponding $Pg*Cx( )$ version, if there is one.

- $Pg*Cx( )$ — you can specify the draw or graphics context. If the first argument is a $void*$, it takes a draw context. If the first argument is a $PhGC_t*$, it takes a graphics context.

- $Pg*v( )$ and $Pg*Cxv( )$ — the data isn’t physically copied into the draw buffer. Instead, a pointer to the array is stored until the draw buffer is flushed. Make sure you call $PgFlush()$ before you modify the array contents. The $Pg*Cxv( )$ are the same as versions $Pg*Cx( )$ in that they take a draw or graphics context as the first argument.
The $P_g*mx()$ functions in the library are deprecated, and have been replaced with $P_g*vf()$ versions.
**PgAlphaOff(), PgAlphaOffCx()**

*Turn alpha blending operations off*

**Synopsis:**

```c
void PgAlphaOff( void );

void PgAlphaOffCx(PhGC_t *gc);
```

**Arguments:**

- `gc` *PgAlphaOffCx()* only. A pointer to a graphics context, as returned by *PgCreateGC()* or *PgGetGC()*.

**Library:**

`ph`

**Description:**

These functions turn alpha blending operations off. *PgAlphaOff()* works on the current graphics context, while you can specify the graphics context for *PgAlphaOffCx()*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgAlphaOn*(), *PgSetAlpha*()

“Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgAlphaOn(), PgAlphaOnCx()**

*Turn alpha blending operations on*

**Synopsis:**

```c
void PgAlphaOn( void );

void PgAlphaOnCx(PhGC_t *gc);
```

**Arguments:**

- `gc`  
  `PgAlphaOnCx()` only. A pointer to a graphics context, as returned by `PgCreateGC()` or `PgGetGC()`.

**Library:**

`ph`

**Description:**

These functions turn alpha blending operations on. `PgAlphaOn()` works on the current graphics context, while you can specify the graphics context for `PgAlphaOnCx()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgAlphaOff*()`, `PgSetAlpha*()`,

“Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PgAlphaValue( PgColor_t color );
```

**Arguments:**

color  The composite color, of type `PgColor_t`, that you want to get the alpha component of.

**Library:**

`ph`

**Description:**

This macro extracts the alpha color component from a composite color value. The result is between 0 and 255.

This macro doesn’t check the color model currently in use, and gives undefined results if you’re not using the `Pg_CM_ARGB` model.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgARGB()`, `PgBlueValue()`, `PgCMY()`, `PgColor_t`, `PgGreenValue()`, `PgHSV()`, `PgRedValue()`, `PgRGB()`, `PgSetFillColor()`, `PgSetFillDither()`
“Color” and “Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgARGB()**

*Convert alpha, red, green, and blue values to composite color format*

Synopsis:

```c
PgColor_t PgARGB( int A,
                int R,
                int G,
                int B );
```

Arguments:

- **A** The alpha value.
- **R** The red value.
- **G** The green value.
- **B** The blue value.

Library:

`ph`

Description:

This macro converts alpha, red, green, and blue values into a composite color value (of type `PgColor_t`). The values for alpha, red, green, and blue range from 0 to 255. If you set the red, green, and blue values to 0, the color is black; if you set them to 255, the color is white. The meaning of the alpha value depends on the alpha model that you’re using.

This macro doesn’t check the color model currently in use, and gives undefined results if you’re not using the `Pg_CM_ARGB` model.

Classification:

Photon
### PgARGB()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PgAlphaValue(), PgBlueValue(), PgColor_t, PgGreenValue(), PgRedValue(), PgRGB(), PgSetFillColor(), PgSetStrokeColor(), PgSetTextColor()*

“Color” and “Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
void PgBackgroundShadings( PgColor_t bg,
    PgColor_t *ts,
    PgColor_t *bs);
```

**Arguments:**

- **bg** A `PgColor_t` object that specifies the background color that you want to base the shading colors on.
- **ts** NULL, or a pointer to a `PgColor_t` object where the function stores the calculated top shading color.
- **bs** NULL, or a pointer to a `PgColor_t` object where the function stores the calculated bottom shading color.

**Library:**

`ph`

**Description:**

This function calculates the top and bottom shading colors that may be used in a border to give an object a 3D appearance. Where possible (based on the brightness of the background color), the top border color is lighter than the background and the bottom border color is darker than the background. Either of `ts` or `bs` may be NULL if that component isn’t required.

**Classification:**

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PgCalcColorContrast(), PgColor_t

“Color” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
Synopsis:

```c
int PgBevelBox( PhPoint_t *ul,
                PhPoint_t *lr,
                PgColor_t light_color,
                PgColor_t flat_color,
                PgColor_t dark_color,
                short depth,
                short width,
                PgColor_t outline_color,
                PgColor_t inline_color,
                int flags );
```

```c
int PgBevelBoxCx( void * dc,
                  PhPoint_t *ul,
                  PhPoint_t *lr,
                  PgColor_t light_color,
                  PgColor_t flat_color,
                  PgColor_t dark_color,
                  short depth,
                  short width,
                  PgColor_t outline_color,
                  PgColor_t inline_color,
                  int flags );
```

Arguments:

- **dc**: `PgBevelBoxCx()` only. A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`
PgBevelBox(), PgBevelBoxCx()

ul, lr
Pointers to PhPoint_t structures that define the upper left and lower right corners of the beveled box.

light_color
A PgColor_t that defines the lightest color in the bevel gradient.

flat_color
The middle/neutral color in the bevel gradient, and the box’s fill color.

dark_color
The darkest color in the bevel gradient.

depth
The number of isochrome lines in the light-to-flat and flat-to-dark bevel gradients. The sign of the depth controls the location of an (imaginary) light source:

- Positive depth — illuminated from the top left.
- Negative depth — illuminated from the bottom right.

width
The width of the bevel, not including outlines and inlines.

outline_color
The color of the outline rectangle.

inline_color
The color of the inline rectangle.

flags
Flags that affect the appearance of the beveled box:

- Pg_BVB_FILL — fill the beveled box with flat-color.
**PgBevelBox(), PgBevelBoxCx()**

- **Pg_BVB_FULL_GRADIENTS** — each bevel has two gradients: light-to-flat and flat-to-dark. If this isn’t set, the bevels have only one gradient; see the illustration above.
- **Pg_BVB_DRAW_LEFT** — draw the left edge of the beveled box.
- **Pg_BVB_DRAW_RIGHT** — draw the right edge of the beveled box.
- **Pg_BVB_DRAW_TOP** — draw the top edge of the beveled box.
- **Pg_BVB_DRAW_BOTTOM** — draw the bottom edge of the beveled box.
- **Pg_BVB_DRAW_ALL** — draw all edges of the beveled box.
- **Pg_BVB_DRAW_BITS** — turn on all of the above flag bits.

**Library:**

```
ph
```

**Description:**

These functions draw a beveled box with gradients. *PgBevelBox()* works on the current draw context, while you can specify the draw context for *PgBevelBoxCx()*.

The only difference between these functions and *PgDrawGradientBevelBox()* or *PgDrawGradientBevelBoxCx()* is that the latter allow separate specifications for the upper-left and the lower-right flat colors. The two flat colors are the same in these function.
*PgBevelBox(), PgBevelBoxCx()*

**Returns:**

0  Success.
-1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PgColor_t, PgContrastBevelBox*, PgDrawGradientBevelBox*, PhPoint_t

“Gradients” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgBlit(), PgBlitCx()**

**Blit an area**

**Synopsis:**

```c
int PgBlit( const PhRect_t *rect,
            const PhPoint_t *offset );

int PgBlitCx( void *dc,
              const PhRect_t *rect,
              const PhPoint_t *offset );
```

**Arguments:**

- `dc`  
  `PgBlitCx()` only. A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- `rect`  
  A pointer to a `PhRect_t` structure that defines the area the function blits.

- `offset`  
  A pointer to a `PhPoint_t` that defines an offset for the blitted area `rect`.

**Library:**

`ph`

**Description:**

These functions “blit” the area that is defined by `rect`. The area is blitted by the given `offset`. Other windows aren’t affected by the blit.

`PgBlit()` blits the region defined by the region set for the current draw context, while `PgBlitCx()` lets you define the draw context `dc`. 
**PgBlit(), PgBlitCx()**

**Returns:**

A nonnegative value

-1 The blit failed, possibly because the Photon Manager wasn’t running.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhBlit(), PhPoint_t, PhRect_t, PtClippedBlit(), PtWidgetRid()`
Synopsis:

`int PgBlueValue( PgColor_t color );`

Arguments:

color A composite color value, of type `PgColor_t`.

Library:

```
ph
```

Description:

This macro extracts the blue color component from a composite color value. The result is between 0 and 255.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

Caveats:

`PgBlueValue()` is a macro.

See also:

`PgAlphaValue(), PgRGB(), PgCMY(), PgColor_t, PgGreenValue(), PgHSV(), PgRedValue() PgRGB(), PgSetFillColor(), PgSetFillDither()`
"Color" in the Raw Drawing and Animation chapter of the Photon 
Programmer's Guide
**PgCalcColorContrast()**

*Compute light and dark colors to use for a gradient*

**Synopsis:**

```c
void PgCalcColorContrast( PgColor_t flat_color,
                          int contrast,
                          PgColor_t *light,
                          PgColor_t *dark );
```

**Arguments:**

- **flat_color**  
The base color towards which the computed colors are intended to converge. It’s the color in the middle of the gradient.

- **contrast**  
The amount of contrast that you want in the gradient, ranging from 0 (low contrast) through 255 (high contrast). The contrast is biased towards the light color.

- **light**  
A pointer to the PgColor_t in which to store the lightest color for the gradient.

- **dark**  
A pointer to the PgColor_t in which to store the darkest color for the gradient.

**Library:**

ph

**Description:**

This function computes light and dark colors that can be used to construct gradients. It’s used by Photon to compute bevel gradients.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt handler</td>
</tr>
</tbody>
</table>

*continued...*
PgCalcColorContrast()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgBackgroundShadings(), PgColor_t

“Gradients” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PgChromaOff(), PgChromaOffCx()

Turn chroma key operations off

Synopsis:

\[
\begin{align*}
\text{void PgChromaOff( void );}
\text{void PgChromaOffCx(PhGC_t *gc);} \\
\end{align*}
\]

Arguments:

\[ \text{gc} \quad \text{PgChromaOffCx()} \text{ only. A pointer to a graphics context, as} \]
\[ \text{returned by PgCreateGC()} \text{ or PgGetGC().} \]

Library:

\[ \text{ph} \]

Description:

These functions turn chroma key operations off. \textit{PgChromaOff()} works on the current graphics context, while you can specify the graphics context for \textit{PgChromaOffCx()}.

Classification:

\[ \text{Photon} \]

\begin{center}
\begin{tabular}{l|c}
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}
\end{center}

See also:

\[ \text{PgChromaOn*()}, \text{PgSetChroma*()} \]

“Chroma key support” in the Raw Drawing and Animation chapter of the Photon \textit{Programmer’s Guide}
**PgChromaOn(), PgChromaOnCx()**

*Turn chroma key operations on*

**Synopsis:**

```c
void PgChromaOn( void );

void PgChromaOnCx(PhGC_t *gc);
```

**Arguments:**

- `gc` *PgChromaOnCx() only*. A pointer to a graphics context, as returned by *PgCreateGC()* or *PgGetGC()*.

**Library:**

`ph`

**Description:**

These functions turn chroma key operations on. *PgChromaOn()* works on the current graphics context, while you can specify the graphics context for *PgChromaOnCx()*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgChromaOff*(), *PgSetChroma*()

“Chroma key support” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgClearDrawBuffer(), PgClearDrawBufferCx()**

*Reset the current draw buffer*

**Synopsis:**

```c
void PgClearDrawBuffer( void );
void PgClearDrawBufferCx( void *dc );
```

**Arguments:**

- `dc`  
  `PgClearDrawBufferCx()` only. A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

**Library:**

`ph`

**Description:**

These functions reset the current draw buffer without flushing. `PgClearDrawBuffer()` works on the current draw context, while you can specify the draw context for `PgClearDrawBufferCx()`.

**Examples:**

```c
/*
 * Draw the following group of 3 lines
 */
PgDrawILine( 100, 100, 200, 300 );
PgDrawILine( 200, 300, 700, 700 );
PgDrawILine( 700, 700, 0, 0 );
PgFlush();

/*
 * Don’t draw the following group of 3 lines
 */
PgDrawILine( 50, 100, 50, 300 );
PgDrawILine( 300, 20, 30, 700 );
PgDrawILine( 500, 700, 0, 100 );
PgClearDrawBuffer();
PgFlush();
```
**PgClearDrawBuffer(), PgClearDrawBufferCx()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgFlush*(), *PgSetDrawBufferSize*()
**Synopsis:**

```c
void PgClearTranslation( void );

void PgClearTranslationCx(PhGC_t *gc);
```

**Arguments:**

- `gc` *PgClearTranslationCx()* only. A pointer to a graphics context, as returned by *PgCreateGC()* or *PgGetGC()*.

**Library:**

ph

**Description:**

These functions restore the current translation to the default (0,0). *PgClearTranslation()* works on the current graphics context, while you can specify the graphics context for *PgClearTranslationCx()*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgSetTranslation*()
**PgCMY()**

Convert cyan, magenta, and yellow values to composite color format

**Synopsis:**

```c
PgColor_t PgCMY( int C, int M, int Y );
```

**Arguments:**

- **C** The cyan component.
- **M** The magenta component.
- **Y** The yellow component.

**Library:**

ph

**Description:**

This macro converts cyan, magenta, and yellow values into a `PgColor_t` structure. It lets you approximate print-industry colors. The values for `C`, `M`, and `Y` range from 0 to 255. If you set all three arguments to 0, the color is white; if you set all three to 255, the color is black.

**Returns:**

A composite color value.

**Examples:**

<table>
<thead>
<tr>
<th>Color</th>
<th>Composite color value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td><code>PgCMY( 255, 255, 255 );</code></td>
</tr>
<tr>
<td>White</td>
<td><code>PgCMY( 0, 0, 0 );</code></td>
</tr>
<tr>
<td>Red</td>
<td><code>PgCMY( 0, 255, 255 );</code></td>
</tr>
<tr>
<td>Green</td>
<td><code>PgCMY( 255, 0, 255 );</code></td>
</tr>
</tbody>
</table>

*continued...*
### PgCMY()

<table>
<thead>
<tr>
<th>Color</th>
<th>Composite color value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>PgCMY( 255, 255, 0 );</td>
</tr>
<tr>
<td>Orange</td>
<td>PgCMY( 0, 90, 255 );</td>
</tr>
<tr>
<td>Slate Blue</td>
<td>PgCMY( 175, 160, 121 );</td>
</tr>
</tbody>
</table>

### Classification:

Photon

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

PgColor_t, PgBlueValue(), PgGreenValue(), PgHSV(), PgRedValue(), PgRGB(), PgSetFillColor(), PgSetFillDither()

“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```
unsigned long PgColor_t;
```

**Description:**

The `PgColor_t` type definition describes a composite color value. The interpretation of the color depends on the current color model, which you can set by calling `PgSetColorModel()`.

The color models are:

- **Pg_CM_PRGB**  
  Photon-Red-Green-Blue (the default model). The most significant byte holds Photon-specific flags. The lowest 8 bits contain the blue value, the next 8 bits contain the green value, and the next 8 bits after that the red value:

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>rrrr</td>
<td>gggg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bbbb</td>
</tr>
</tbody>
</table>

- **Pg_CM_RGB**  
  The same as Pg_CM_PRGB.

- **Pg_CM_ARGB**  
  Alpha-Red-Green-Blue. The most significant byte is an Alpha value. The lowest 8 bits contain the blue value, the next 8 bits contain the green value, and the next 8 bits after that the red value, as described for Pg_CM_PRGB.

If the destination draw context (default, offscreen context, etc.) has enough room to hold the alpha channel after color conversion, the alpha channel is written in with the color information when writing the pixel into memory. For example:

```c
PgSetFillColor(0x80FFFFFF);
PgDrawIRect(0,0,99,99,Pg_DRAW_FILL);
PgSetFillColor(0x40FFFFFF);
PgDrawIRect(100,0,199,99,Pg_DRAW_FILL);
```
PgColor_t

PgSetFillColor(0xC0FFFFFF);
PgDrawIRect(200,0,299,99,Pg_DRAW_FILL);

If the current mode of the destination draw context is 32-bit ARGB, the complete values are written to the pixel RAM.

If the destination draw context is 1555 (1 bit alpha, 5 bits red, 5 bits green, 5 bits blue), the first rectangle is 0xFFFF, the second is 0x7FFF and the third is 0xFFFF. Note that the second rectangle’s alpha value is 0 (there’s only 1 bit for alpha in this mode), so 0 through 0x7F convert to 0 and 0x80 through 0xff convert to 1.

If the destination draw context is 565 or 888 mode then the alpha channel information is lost, as there is no alpha channel.

Standard colors

At least the following colors are defined in <photon/Pg.h>:

Pg_BLACK       Pg_MAGENTA
Pg_DGRAY        Pg_CYAN
Pg_MGRAY        Pg_DGREEN
Pg_GRAY         Pg_DCYAN
Pg_WHITE        Pg_DBLUE
Pg_RED          Pg_BROWN
Pg_GREEN        Pg_PURPLE
Pg_BLUE         Pg_CELIDON
Pg_YELLOW

We’ve defined the following colors for compatibility with standard VGA colors:

Pg_VGA0         Pg_VGA3
Pg_VGA1         Pg_VGA4
Pg_VGA2         Pg_VGA5
**PgColor_t**

Pg_VGA6           Pg_VGAB
Pg_VGA7           Pg_VGAC
Pg_VGA8           Pg_VGAD
Pg_VGA9           Pg_VGAE
Pg_VGAA

We've also defined the following in `<photon/Pg.h>`:

These colors work only in Pg_CM_PRGB or Pg_CM_RGB mode.

Pg_DEVICE_COLOR

Interpret up to the least significant 24 bits as the value to put into video memory.

This facility depends on the video hardware, and behaves differently depending on the graphics driver.

Pg_INDEX_COLOR

Interpret the color as an index into the current palette.

Pg_INVERT_COLOR

Use with `PgSetDrawMode(Pg_DRAWMODE_XOR)` for high-visibility XOR drawing.

Pg_TRANSPARENT

Subsequent draw events won't be rendered.

**Classification:**

Photon

**See also:**

`PgAlphaValue()`, `PgRGB()`, `PgBlueValue()`, `PgCMY()`,
`PgColorHSV_t`, `PgGetColorModel()`, `PgGreenValue()`,
`PgHSV2RGB()`, `PgRedValue()`, `PgRGB()`, `PgRGB2HSV()`,

---

464 Chapter 8 • Pg—Graphics
PgColor_t

PgSetColorModel(), PgSetFillColor(), PgSetFillDither(),
PgSetStrokeColor(), PgSetStrokeDither(), PgSetTextColor(),
PgSetTextDither()

“Color” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**Synopsis:**

```c
typedef struct {
    unsigned short hue;
    unsigned char sat, vid;
} PgColorHSV_t;
```

**Description:**

The `PgColorHSV_t` structure describes a hue-saturation-value color.
It’s used to convert a color defined as red, green, and blue into HSV. It contains at least the following members:

- **unsigned short hue**
  Color angle; see `PgHSV()`.

- **unsigned char sat**
  Color saturation.

- **unsigned char vid**
  Color value, or brightness.

**Classification:**

Photon

**See also:**

- `PgColor_t`, `PgHSV()`, `PgHSV2RGB()`, `PgRGB2HSV()`
- “Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PgColorMatch(int n,
        PgColor_t const *in,
        PgColor_t *out);
```

Arguments:

- `n` The number of colors in the array to find a match for.
- `in` An array of `PgColor_t` objects that specifies the colors that you want to match.
- `out` An array of `PgColor_t` objects that the function fills with the best color matches for the corresponding entry in the `in` array.

Library:

`ph`

Description:

This function queries the graphics driver for the best color matches for a number of color values. This is particularly useful with a palette-based graphics driver.

An array of `n` colors from the `in` array is passed to the driver, which selects the closest match for each color and returns these in the `out` array.

With a true or direct-color driver, the color is returned unchanged. With a palette-based driver, the closest color is found by computing within a RGB color cube the Cartesian distance between the color and each palette entry, and selecting the closest entry of a similar intensity.

Returns:

- 0 Successful completion.
- -1 An error occurred.
**PgColorMatch()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PgColor_t

“Color” in the Raw Drawing and Animation chapter of the Photon

*Programmer’s Guide*
Synopsis:

```c
int PgConfigScalerChannel(
    PgVideoChannel_t *channel,
    PgScalerProps_t *props);
```

Arguments:

- `channel` A pointer to a `PgVideoChannel_t` structure that specifies the channel you want to configure.
- `props` A pointer to a `PgScalerProps_t` structure that specifies how to configure the channel.

Library:

```
ph
```

Description:

This function configures the video overlay scaler channel specified by `channel`. It uses the configuration information specified by `props` to set the state of the overlay hardware.

Once the scaler is properly configured, the structure pointed to by `channel` contains pointers to offscreen context structures that describe the video data buffers. You can use `PdGetOffscreenContextPtr()` to retrieve pointers to the video data buffers.

If you specify `Pg_SCALE_PROP_DRAW_TARGETABLE` in `props->flags` and the targeted surface isn’t RGB, `PgConfigScalerChannel()` fails and returns -1.

Returns:

- 0 The state of the scaler hardware was successfully changed, and the video data frame buffers haven’t changed since the last call to `PgConfigScalerChannel()`.
1 The state of the scaler hardware was successfully changed, and the offscreen buffer contexts in the channel structure have changed. In this case, information about the video data buffers that was returned by previous calls is no longer valid.

-1 An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PdGetOffscreenContextPtr(), PgCreateVideoChannel(), PgDestroyVideoChannel(), PgGetOverlayChromaColor(), PgGetScalerCapabilities(), PgNextVideoFrame(), PgScalerCaps_t, PgScalerProps_t, PgVideoChannel_t

“Video overlay” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PgContextBlit(), PgContextBlitCx()

Copy data from a rectangle in one context to another context

Synopsis:

```c
void PgContextBlit( PdOffscreenContext_t *src,
                    PhRect_t *src_rect,
                    PdOffscreenContext_t *dst,
                    PhRect_t *dst_rect );

void PgContextBlitCx( void *dc,
                      PdOffscreenContext_t *src,
                      PhRect_t *src_rect,
                      PdOffscreenContext_t *dst,
                      PhRect_t *dst_rect );
```

Arguments:

- **dc**  
  *PgContextBlitCx() only.* A void pointer to any type of draw context. Examples of draw contexts are:
  - a PhDrawContext_t returned by PhDCCreate()
  - a PmMemoryContext_t returned by PmMemCreateMC()
  - a PpPrintContext_t returned by PpCreatePC()
  - a PdOffscreenContext_t returned by PdCreateOffscreenContext()

- **src**  
  The source context, or NULL to copy from the display.

- **src_rect**  
  A pointer to a PhRect_t structure that defines the rectangle in the source to copy. If **src** is NULL, **src_rect** is relative to the emitting region (see PgSetRegion()) but isn’t clipped by overlapping windows.

- **dst**  
  The destination context, or NULL to copy to the display.

- **dst_rect**  
  The rectangle in the destination to which to copy. The source data is scaled to fit this rectangle.
**Library:**

**Description:**

These functions copy data from a rectangle in one context to a rectangle in another context. *PgContextBlit()* works on the current draw context, while you can specify the draw context for *PgContextBlitCx()*.

These functions obey many of the parameters in the draw state: Chroma, Alpha, Raster Operation (DrawMode), Fill Pattern (FillDither), and Fill Transparency pattern (FillTransp).

Fill Patterns are used only if the Raster Operation requested includes a pattern operation (e.g. Pg_DrawModePSo). Fill Transparency patterns are obeyed whether or not the Raster operation includes a pattern. If the current Raster Operation is a Photon 1.xx raster operation (e.g. Pg_DRAWMODE_OPAQUE), then all patterns are ignored.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdCreateOffscreenContext(), PdDupOffscreenContext(), PdGetOffscreenContextPtr(), PdOffscreenContext_t, PgContextBlitArea*(), PgSetRegion*(), PgSwapDisplay*(), PhRect_t*
`PgContextBlit(), PgContextBlitCx()`

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgContextBlitArea(), PgContextBlitAreaCx()**

Copy data from an area in one context to another context

**Synopsis:**

```c
void PgContextBlitArea( PdOffscreenContext_t *src,
                        PhArea_t *src_area,
                        PdOffscreenContext_t *dst,
                        PhArea_t *dst_area );
```

```c
void PgContextBlitAreaCx( void *dc,
                          PdOffscreenContext_t *src,
                          PhArea_t *src_area,
                          PdOffscreenContext_t *dst,
                          PhArea_t *dst_area );
```

**Arguments:**

- **dc**
  
  `PgContextBlitAreaCx()` only. A void pointer to any type of draw context. Examples of draw contexts are:

  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- **src**
  
  The source context, or NULL to copy from the display.

- **src_area**
  
  A pointer to a `PhArea_t` structure that defines the area in the source to copy. If `src` is NULL, `src_rect` is relative to the emitting region (see `PgSetRegion()`) but isn’t clipped by overlapping windows.

- **dst**
  
  The destination context, or NULL to copy to the display.

- **dst_area**
  
  A pointer to a `PhArea_t` structure that defines the area in the destination to copy the data to. The source data is scaled to fit this rectangle.
Library:

Description:

These functions copy data from an area in one offscreen context to an area in another offscreen context. *PgContextBlitArea()* works on the current draw context, while you can specify the draw context for *PgContextBlitAreaCx()*.

These functions obey many of the parameters in the draw state: Chroma, Alpha, Raster Operation (DrawMode), Fill Pattern (FillDither), and Fill Transparency pattern (FillTransp).

Fill Patterns are used only if the Raster Operation requested includes a pattern operation (e.g. Pg_DrawModePSo). Fill Transparency patterns are obeyed whether or not the Raster operation includes a pattern. If the current Raster Operation is a Photon 1.xx raster operation (e.g. Pg_DRAWMODE_OPAQUE), then all patterns are ignored.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PdCreateOffscreenContext()*, *PdDupOffscreenContext()*, *PdGetOffscreenContextPtr()*, *PhOffscreenContext_t*, *PgContextBlit*(), *PgSetRegion*(), *PgSwapDisplay*(), *PhArea_t*

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgContrastBevelBox(), PgContrastBevelBoxCx()**

*Draw a beveled box with gradients and a given level of contrast*

**Synopsis:**

```c
int PgContrastBevelBox( PhPoint_t *ul,
                        PhPoint_t *lr,
                        PgColor_t flat_color,
                        int contrast,
                        short depth,
                        short width,
                        PgColor_t outline_color,
                        PgColor_t inline_color,
                        int flags );

int PgContrastBevelBoxCx( void *dc,
                           PhPoint_t *ul,
                           PhPoint_t *lr,
                           PgColor_t flat_color,
                           int contrast,
                           short depth,
                           short width,
                           PgColor_t outline_color,
                           PgColor_t inline_color,
                           int flags );
```

**Arguments:**

*dc*  
*PgContrastBevelBoxCx()* only. A void pointer to any type of draw context. Examples of draw contexts are:

- a *PhDrawContext_t* returned by *PhDCCreate()*
- a *PmMemoryContext_t* returned by *PmMemCreateMC()*
- a *PpPrintContext_t* returned by *PpCreatePC()*
- a *PdOffscreenContext_t* returned by *PdCreateOffscreenContext()*

*ul, lr*  
Pointers to *PhPoint_t* that define the upper left and lower right corners of the beveled box.
**PgContrastBevelBox(), PgContrastBevelBoxCx()**

- **flat_color**
  
  A `PgColor_t` that defines the middle/neutral color in the bevel-gradients, and the box’s fill color if you set `Pg_BVB_FILL` in the `flags`.

- **contrast**
  
  The contrast in the light-to-flat and flat-to-dark gradients, in the range 0 (low/no contrast) to 255 (high/max contrast).

- **depth**
  
  The number of isochrome lines in the light-to-flat and flat-to-dark bevel gradients. The sign of the depth controls the location of an (imaginary) light source:

  - Positive depth — illuminated from the top left.
  - Negative depth — illuminated from the bottom right.

- **width**
  
  The width of the bevel, not including outlines and inlines.

- **outline_color**
  
  The color of the outline rectangle.

- **inline_color**
  
  The color of the inline rectangle.

- **flags**
  
  Flags that affect the appearance of the beveled box:

  - `Pg_BVB_FILL` — fill the beveled box with `flat-color`.
  - `Pg_BVB_FULL_GRADIENTS` — each bevel has two gradients: light-to-flat and flat-to-dark. If
**PgContrastBevelBox(), PgContrastBevelBoxCx()**

This isn’t set, the bevels have only one gradient; see the illustration above.

- **Pg_BVB_DRAW_LEFT** — draw the left edge of the beveled box.
- **Pg_BVB_DRAW_RIGHT** — draw the right edge of the beveled box.
- **Pg_BVB_DRAW_TOP** — draw the top edge of the beveled box.
- **Pg_BVB_DRAW_BOTTOM** — draw the bottom edge of the beveled box.
- **Pg_BVB_DRAW_ALL** — draw all edges of the beveled box.
- **Pg_BVB_DRAW_BITS** — turn on all of the above flag bits.

**Library:**

ph

**Description:**

These functions draw a beveled box with gradients and a given level of contrast. **PgContrastBevelBox()** works on the current draw context, while you can specify the draw context for **PgContrastBevelBoxCx()**.

**Returns:**

- **0** Success.
- **-1** An error occurred.

**Classification:**

Photon
**PgContrastBevelBox(), PgContrastBevelBoxCx()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
</tr>
<tr>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgColor_t, PgBevelBox*, *PgDrawGradientBevelBox*,
*PhPoint_t*

“Gradients” in the Raw Drawing and Animation chapter of the Photon
*Programmer’s Guide*
**PgCreateDriverRegion()**

*Create a region that’s owned by the graphics driver*

**Synopsis:**

```c
PhRid_t PgCreateDriverRegion(
    PdOffscreenContext_t *osc,
    PhPoint_t   *origin,
    PhRect_t    *rect,
    PhRid_t     parent);
```

**Arguments:**

- `osc` A pointer to the `PdOffscreenContext_t` structure for the offscreen context. This argument must not be NULL.
- `origin` NULL, or a pointer to a `PhPoint_t` structure that specifies the origin of the region, relative to its parent. The default is (0,0) if this argument is NULL.
- `rect` NULL, or a pointer to a `PhRect_t` structure that specifies the region rectangle, relative to its origin. If this argument is NULL, the function uses a rectangle whose members are set to 0.
- `parent` The ID of the parent region, or -1 if you don’t want to specify it.

**Library:**

`ph`

**Description:**

`PgCreateDriverRegion()` creates a region that’s owned by the graphics driver and is sensitive to draw events.

The region parent defaults to the driver’s input group’s parent region. `PhRegionOpen()` defines all of the other default settings for the region. All draw events collected by the driver region are targeted at the specified offscreen context.

The driver region persists until you explicitly close it (which we don’t recommend) or until the corresponding offscreen context is destroyed.
You must target this function at a device by calling `PdSetTargetDevice()`.

A few notes:

- A single offscreen context is not usually associated with more than one driver region.
- The draw event translation isn’t cleared when the drawstream is directed to the offscreen context.
- The driver region can be offset from its offscreen context using `PdSetOffscreenTranslation()`.
- Client applications can manipulate a driver region by using its region ID. We don’t generally recommend this.
- The driver region’s handle (Ph_REGION_HANDLE) is reserved for use by the graphics driver. Don’t change it.
- Since the region is owned by the graphics driver, all events collected by the region — including input events — are delivered to the graphics driver and not to the application that called this function.

**Returns:**

A nonnegative region id, or -1 if an error occurred.

**Errors:**

- **EFAULT**
  The function couldn’t access the offscreen context.
- **EINVAL**
  The osc argument is NULL, or the call to `PhRegionOpen()` failed.
PgCreateDriverRegion()

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdOffscreenContext_t, PdSetOffscreenTranslation(), PdSetTargetDevice(), PhPoint_t, PhRect_t, PhRegionOpen()
Synopsis:

PhGC_t *PgCreateGC( int unused );

Arguments:

unused PgCreateGC() doesn’t use this argument; set it to 0.

Library:

ph

Description:

This function allocates a graphics context. A graphics context contains the entire draw state, including color, clipping, and current region.

If your application calls both Pg and Pt functions, you must provide one graphics context for the Pg functions and a separate context for the Pt functions. To do this, call PgSetGC() every time you switch from one API to the other.

Returns:

A pointer to a graphics context, or NULL if an error occurs.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
PgCreateGC()

See also:

PgDestroyGC(), PgSetDrawBufferSize(), PgSetGC()
PgCreateLayerSurface()

Create an offscreen context that a layer can display

Synopsis:

```c
PdOffscreenContext_t *PgCreateLayerSurface(
    int layer,
    int surface_index,
    int format_index,
    unsigned short width,
    unsigned short height,
    unsigned long flags);
```

Arguments:

- `layer` The layer index, which must be 0 or greater.
- `surface_index` The surface index, which must be 0 or greater.
- `format_index` The image format index, which corresponds to the index used in `PgGetLayerCaps()`.
- `width`, `height` The dimensions of the context, in pixels.
- `flags` Defined flags are:
  - `Pg_Osc_MEM_PAGE_ALIGN` — ensure that the offscreen context that’s created is aligned to __PAGESIZE (4K on an x86). You should set this flag if you’re using `PdGetOffscreenContextPtr()`.
  - `Pg_Osc_MEM_2D_READABLE` — create an offscreen context that is readable by a 2D engine.
  - `Pg_Osc_MEM_2D_WRITABLE` — create an offscreen context that is targetable by a 2D engine.
  - `Pg_Osc_MEM_HINT_CPU_READ` — create an offscreen context that is optimized by the driver for fast reading by the CPU. This flag has a lower priority than other flags.
PgCreateLayerSurface()

- PgOSC_MEM_HINT_CPU_WRITE — create an offscreen context that is optimized by the driver for fast writing by the CPU. This flag has a lower priority than other flags.

Library:

ph

Description:

PgCreateLayerSurface() creates an offscreen context that can be nearby the given layer in the given format.

For layer formats that require data from more than one surface, surface_index is used to distinguish each surface.

For these layer formats:
- Pg_LAYER_FORMAT_YVU9
- Pg_LAYER_FORMAT_YV12
- Pg_LAYER_FORMAT_YUV420

Use these surface indices:

0   Y plane
1   U plane
2   V plane

For all other layer formats, use 0 for surface_index.

You must target this function at a device by calling PdSetTargetDevice().

If a layer is reconfigured (e.g. its format is changed), previously allocated offscreen contexts might no longer be compatible with the layer.
**PgCreateLayerSurface()**

To tell a layer to read data from a surface, call *PgSetLayerSurface()*.

**WARNING:** You can’t use Photon drawing functions on a surface with a format that doesn’t match the current video mode.

**Returns:**

A pointer to a *PdOffscreenContext_t* structure, or NULL if an error occurred.

**Errors:**

- **EINVAL** The format, dimensions, or flags are incompatible with the given layer capabilities, or the layer or layer surface doesn’t exist.
- **EOPNOTSUPP** The operation isn’t supported.
- **ENOMEM** Memory allocation failed.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdGetOffscreenContextPtr(), PdOffscreenContext_t, PdSetTargetDevice(), PgGetLayerCaps(), PgSetLayerSurface()*

“Layers” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgCreateVideoChannel()**

Create a channel for video streaming

**Synopsis:**

```c
PgVideoChannel_t * PgCreateVideoChannel(
    unsigned type,
    unsigned flags);
```

**Arguments:**

- `type`  The type of channel to create. Currently, the only defined type is `Pg_VIDEO_CHANNEL_SCALER`. This specifies that the video channel is for outputting video frames using scaler hardware.

- `flags`  There are currently no flags; pass 0 for this argument.

**Library:**

`ph`

**Description:**

This function creates a channel to be used for video streaming, and reserves the video hardware for exclusive use by the application.

**Returns:**

A pointer to a `PgVideoChannel_t` structure that describes a channel for subsequent video operations, or NULL if a video channel couldn’t be created (`errno` is set).

**Errors:**

- `EBUSY`  Scaler hardware is present, but is in use by another application.

- `ENXIO`  No scaler hardware is present.
**PgCreateVideoChannel()**

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PgConfigScalerChannel()`, `PgDestroyVideoChannel()`,
- `PgGetOverlayChromaColor()`, `PgGetScalerCapabilities()`,
- `PgNextVideoFrame()`, `PgScalerCaps_t`, `PgScalerProps_t`,
- `PgVideoChannel_t`

“Video overlay” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDefaultAlpha()**
Reset the alpha attribute to its default value

**Synopsis:**

```c
void PgDefaultAlpha(PhGC_t * GC);
```

**Arguments:**

- `GC` A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

- `ph`

**Description:**

This function resets the alpha attribute portion of the provided graphics context to its system default.

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

- `PgDefaultChroma()`, `PgDefaultGC()`, `PgDefaultText()`,
- `PgDefaultModel()`, `PgDefaultStroke()`, `PgSetFillColor*()`,
- `PgSetFillDither*()`, `PgSetFillTransPat*()`, `PgSetFillXORColor*()`

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
void PgDefaultChroma( PhGC_t *GC );
```

**Arguments:**

- `GC`  
  A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

`ph`

**Description:**

This function resets the chroma attribute portion of the provided graphics context to its system default.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PgDefaultAlpha()`, `PgDefaultGC()`, `PgDefaultText()`,
- `PgDefaultMode()`, `PgDefaultStroke()`, `PgSetFillColor()`,
- `PgSetFillDither()`, `PgSetFillTransPat()`, `PgSetFillXORColor()`

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDefaultFill()**

Reset the fill attribute to its default value

**Synopsis:**

```c
void PgDefaultFill( PhGC_t *GC );
```

**Arguments:**

- `GC` A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

ph

**Description:**

This function resets the fill attribute portion of the provided graphics context to its system default.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PgDefaultGC()`, `PgDefaultText()`, `PgDefaultMode()`, `PgDefaultStroke()`, `PgSetFillColor()`, `PgSetFillDither()`, `PgSetFillTransPat()`, `PgSetFillXORColor()`

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDefaultGC()**

 Reset all graphics context attributes to their default system values

**Synopsis:**

```c
void PgDefaultGC( PhGC_t *GC );
```

**Arguments:**

- `GC` A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

`ph`

**Description:**

This function resets all attributes of the provided graphics context to their system defaults.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PgDefaultMode()`, `PgDefaultFill()`, `PgDefaultText()`, `PgDefaultStroke()`
**PgDefaultMode()**

Reset draw mode and plane mask to their default values

**Synopsis:**

```c
void PgDefaultMode( PhGC_t *GC );
```

**Arguments:**

- `GC` A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

`ph`

**Description:**

This function resets the draw mode and plane mask portions of the provided graphics context to their system defaults.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgDefaultGC()`, `PgDefaultFill()`, `PgDefaultText()`, `PgDefaultStroke()`, `PgSetDrawMode()`, `PgSetPlaneMask()`
**PgDefaultStroke()**

*Reset stroke attribute to its system default*

Synopsis:

```c
void PgDefaultStroke( PhGC_t *GC );
```

Arguments:

- **GC**  
  A pointer to a graphics context, as returned by `PgCreateGC()`.

Library:

- ph

Description:

This function resets the stroke attribute portion of the provided graphics context to its system default.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PgDefaultGC()`, `PgDefaultFill()`, `PgDefaultText()`, `PgDefaultMode()`, `PgSetStrokeCap()`, `PgSetStrokeColor()`, `PgSetStrokeDash()`, `PgSetStrokeDither()`, `PgSetStrokeFWidth()`, `PgSetStrokeJoin()`, `PgSetStrokeTransPat()`, `PgSetStrokeWidth()`, `PgSetStrokeXORColor()`

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDefaultText()**

*Reset text attribute to its system default*

**Synopsis:**

```c
void PgDefaultText( PhGC_t *GC );
```

**Arguments:**

- **GC** A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

`ph`

**Description:**

This function resets the text attribute portion of the provided graphics context to system defaults.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PgDefaultGC()`, `PgDefaultFill()`, `PgDefaultMode()`, `PgDefaultStroke()`, `PgSetTextColor()`, `PgSetTextDither()`, `PgSetTextTransPat()`, `PgSetTextXORColor()`

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDestroyGC()**

*Release the resources of a graphics context*

**Synopsis:**

```c
void PgDestroyGC( PhGC_t *GC );
```

**Arguments:**

- *GC* A pointer to a graphics context, as returned by *PgCreateGC()*.

**Library:**

*ph*

**Description:**

This function releases any resources consumed by the specified graphics context.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgCreateGC(), PgGetGC(), PgSetGC()*
**PgDestroyVideoChannel()**  
*Destroy resources associated with a video channel*

**Synopsis:**

```c
void PgDestroyVideoChannel(
    PgVideoChannel_t *channel );
```

**Arguments:**

- `channel` A pointer to a `PgVideoChannel_t` structure for the video channel that you want to destroy.

**Library:**

`ph`

**Description:**

`PgDestroyVideoChannel()` releases any resources associated with the video channel that was created by `PgCreateVideoChannel()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PgConfigScalerChannel(), PgCreateVideoChannel(), PgGetOverlayChromaColor(), PgScalerCapabilities(), PgNextVideoFrame(), PgScalerCaps_t, PgScalerProps_t, PgVideoChannel_t`

“Video overlay” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawArc(), PgDrawArcCx()**

*Draw an arc, pie, or chord*

**Synopsis:**

```c
int PgDrawArc( PhPoint_t const *center,
                 PhPoint_t const *radii,
                 unsigned int start,
                 unsigned int end,
                 int flags );

int PgDrawArcCx( void *dc,
                 PhPoint_t const *center,
                 PhPoint_t const *radii,
                 unsigned int start,
                 unsigned int end,
                 int flags );
```

**Arguments:**

- **dc** *PgDrawArcCx() only.* A void pointer to any type of draw context. Examples of draw contexts are:
  - a PhDrawContext_t returned by PhDCCreate()
  - a PmMemoryContext_t returned by PmMemCreateMC()
  - a PpPrintContext_t returned by PpCreatePC()
  - a PdOffscreenContext_t returned by PdCreateOffscreenContext()

- **center** A pointer to a PhPoint_t structure that specifies the center of the arc.

- **radii** A pointer to a PhPoint_t structure that specifies the x and y radii of the arc.

- **start, end** The start and end angles, in bi-grads (see below).

- **flags** Flags that control what type of arc is drawn; see below.
**PgDrawArc(), PgDrawArcCx()**

**Library:**

`ph`

**Description:**

These functions build a command in the draw buffer to draw an arc. `PgDrawArc()` works on the current draw context, while you can specify the draw context for `PgDrawArcCx()`.

The arc is drawn counter-clockwise, from the start angle to the end angle. To draw a complete circle, make the two angles equal to each other. An angle of 0 bi-grads is on the horizon to the right of the center.

A circle is divided into 65536 gradations called *binary gradations* or *bi-grads*. Thus, `0x2000` is 45 degrees, `0x4000` is 90 degrees, `0x8000` is 180 degrees, and `0xC000` is 270 degrees.

The *flags* argument controls what type of arc is drawn:

- **Pg_ARC_CHORD**: A curve with the end points connected by a straight line.
- **Pg_ARC PIE**: A curve with the end points connected to the arc’s center.
- **Pg_ARC**: The curve alone.

You can OR one of the following into any *flags* value:

- **Pg_DRAW_STROKE**: draw as a line.
- **Pg_DRAW_FILL**: fill the arc.
- **Pg_DRAW_FILL_STROKE**: fill the arc, then stroke it.
**PgDrawArc(), PgDrawArcCx()**

Returns:

0  Success.

-1  The draw buffer is too small to hold the current draw state and the draw command.

Examples:

The following example:

```c
DrawFillArc() {
    PhPoint_t  c = { 80, 60 };  
    PhPoint_t  r = { 72, 52 };  

    PgSetFillColor( Pg_RED );
    PgDrawArc( &c, &r, 0x0000, 0x4000, Pg_DRAW_FILL | Pg_ARC_CHORD);

    PgSetFillColor( Pg_YELLOW );
    PgDrawArc( &c, &r, 0x5555, 0x9555, Pg_DRAW_FILL | Pg_ARC_PIE);

    PgSetStrokeColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawArc( &c, &r, 0xAAAA, 0xEAAA, Pg_DRAW_FILL_STROKE | Pg_ARC_PIE);
}
```

will draw:

![Graphical representation of the drawn arcs]

The following example:

```c
DrawStrokeArc() {
    PhPoint_t  c = { 80, 60 };  
    PhPoint_t  r = { 72, 52 };  

    PgSetStrokeColor( Pg_WHITE );
    PgDrawArc( &c, &r, 0x0000, 0x4000, Pg_DRAW_STROKE | Pg_ARC_CHORD );
```
**PgDrawArc(), PgDrawArcCx()**

```
PgSetStrokeColor( Pg_YELLOW );
PgDrawArc( &c, &r, 0x5555, 0x9555, Pg_DRAW_STROKE | Pg_ARC_PIE );
PgSetStrokeColor( Pg_YELLOW );
PgDrawArc( &c, &r, 0xAAAA, 0xEAAA, Pg_DRAW_STROKE | Pg_ARC );
```

will draw:

![Arcs](image)

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

**PhPoint_t**

To draw stroked arcs, see also:

`PgSetStrokeCap*(), PgSetStrokeColor*(), PgSetStrokeDash*(), PgSetStrokeDither*(), PgSetStrokeJoin*(), PgSetStrokeWidth*()`

To draw filled arcs, see also:

`PgSetFillColor*(), PgSetFillDither*(), PgSetFillTransPat*()`
"Arrows, ellipses, polygons, and rectangles" in the Raw Drawing and Animation chapter of the Photon *Programmer's Guide*
### PgDrawArrow(), PgDrawArrowCx()

**Draw an arrow that fits inside a given rectangle**

#### Synopsis:

```c
void PgDrawArrow( PhRect_t const *rect,
                  short margin,
                  PgColor_t color,
                  int direction );

void PgDrawArrowCx( void *dc,
                     PhRect_t const *rect,
                     short margin,
                     PgColor_t color,
                     int direction );
```

#### Arguments:

- **dc**  
  `PgDrawArrowCx()` only. A void pointer to any type of draw context. Examples of draw contexts are:

  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- **rect**  
  A pointer to a `PhRect_t` structure that defines the area in which the arrow must fit. If this area is too small to accommodate a tail, the function draws only a triangular arrowhead, it doesn’t draw the rectangle itself.

- **margin**  
  The size of the margin, in pixels, to leave inside the rectangle.

- **color**  
  The fill and stroke color to use for the arrow, expressed as a `PgColor_t`.

- **direction**  
  The direction you want the arrow to point. One of:

  - `Pg_BOTTOM`
  - `Pg_LEFT`
**PgDrawArrow(), PgDrawArrowCx()**

- Pg_RIGHT
- Pg_TOP

**Library:**

ph

**Description:**

These functions call `PgDrawPolygon()` to draw an arrow that fits inside a given rectangle. `PgDrawArrow()` works on the current draw context, while you can specify the draw context for `PgDrawArrowCx()`.

These functions use the current dithering and transparency settings for the fill and stroke, and a stroke width of 1. After drawing the arrow, they reset the fill and stroke colors and the stroke width to the values they had when you called the function.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgColor_t`, `PgDrawBeveled*()`, `PgDrawPolygon*()`, `PhRect_t`

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawBevelBox()**, **PgDrawIBevelBox()**,  
**PgDrawBevelBoxCx()**, **PgDrawIBevelBoxCx()**

**Draw a beveled box**

**Synopsis:**

```c
int PgDrawBevelBox( PhRect_t const *rect,
                    PgColor_t secondary,
                    short width,
                    int flags );

int PgDrawIBevelBox( int x1, int y1,
                      int x2, int y2,
                      PgColor_t secondary,
                      short width,
                      int flags );

int PgDrawBevelBoxCx( void *dc,
                       PhRect_t const *rect,
                       PgColor_t secondary,
                       short width,
                       int flags );

int PgDrawIBevelBoxCx( void *dc,
                        int x1, int y1,
                        int x2, int y2,
                        PgColor_t secondary,
                        short width,
                        int flags );
```

**Arguments:**

- **dc**  
  *PgDrawBevelBoxCx()* and *PgDrawIBevelBoxCx()* only. A void pointer to any type of draw context.  
  Examples of draw contexts are:
  - a *PhDrawContext_t* returned by *PhDCCreate()*
  - a *PmMemoryContext_t* returned by *PmMemCreateMC()*
  - a *PpPrintContext_t* returned by *PpCreatePC()*
  - a *PdOffscreenContext_t* returned by *PdCreateOffscreenContext()*

- **rect** (*PgDrawBevelBox()* and *PgDrawBevelBoxCx()* only)  
  A rectangle that describes the size of the bevel box.
**PgDrawBevelBox(), PgDrawIBevelBox(), PgDrawBevelBoxCx(), PgDrawIBevelBoxCx()**

- $x1, x2, y1, y2$ (PgDrawIBevelBox() and PgDrawIBevelBoxCx() only)
  The coordinates of the top left ($x1, y1$) and bottom right ($x2, y2$) of the bevel box.

- **secondary**
  The color of the lower and right edges of the bevel box. The top and left edges are drawn with the current stroke color.

- **width**
  The thickness of the lines.

- **flags**
  One of the following:
  - Pg_DRAW_STROKE — draw as a line.
  - Pg_DRAW_FILL — fill the box.
  - Pg_DRAW_FILL_STROKE — fill the box, then stroke it.

The **flags** argument must be one of the following:

**Library:**

- **ph**

**Description:**

These functions build a command in the draw buffer to draw a beveled box. This box is used for the outlines of buttons and panes. **PgDrawBevelBox()** and **PgDrawIBevelBox()** work on the current draw context, while you can specify the draw context for **PgDrawBevelBoxCx()** and **PgDrawIBevelBoxCx()**.

For **PgDrawBevelBox()** and **PgDrawBevelBoxCx()**, the **width** parameter must be $\leq 15$.

When you increase the thickness, the lines grow toward the middle of the beveled box. The maximum **width** is given by **Pg_BEVEL_MAX**.
**PgDrawBevelBox()**, **PgDrawIBevelBox()**, **PgDrawBevelBoxCx()**, **PgDrawIBevelBoxCx()**

*PgDrawBevelBox()* and *PgDrawBevelBoxCx()* require a pointer to a *PhRect_t* structure, whereas *PgDrawBevelBox()* and *PgDrawIBevelBoxCx()* take individual arguments.

**Returns:**

0  Success.

-1  The draw buffer is too small to hold the current draw state and the draw command.

**Examples:**

The following example:

```c
DrawBevelBox() {
    PhRect_t r = { 8, 8, 152, 112 };
    PgSetFillColor( Pg_GREY );
    PgSetStrokeColor( Pg_WHITE );
    PgDrawBevelBox( &r, Pg_DGREY, 4,
                    Pg_DRAW_FILL_STROKE );
}
```

will draw:

![Bevel Box Example](image)

**Classification:**

Photon
**PgDrawBevelBox(), PgDrawIBevelBox(), PgDrawBevelBoxCx(), PgDrawIBevelBoxCx()**

### Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

- PgColor_t, PgDrawBeveled*, PgDrawRect*, PgSetFillColor*, PgSetFillDither*, PgSetFillTransPat*, PgSetStrokeColor*, PgSetStrokeDither*, PgSetStrokeTransPat*, PhRect_t

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*


**PgDrawBeveled(), PgDrawBeveledCx()**

*Draw a beveled rectangle or arrow*

**Synopsis:**

```c
int PgDrawBeveled( PhRect_t const *rect,
                   PhPoint_t const *radii,
                   PgColor_t secondary,
                   short width,
                   int flags );

int PgDrawBeveledCx( void *dc;
                      PhRect_t const *rect,
                      PhPoint_t const *radii,
                      PgColor_t secondary,
                      short width,
                      int flags );
```

**Arguments:**

- **dc**: *PgDrawBeveledCx() and PgDrawIBevelBoxCx() only.* A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`
- **rect**: A pointer to a `PhRect_t` that defines the area that the object fills.
- **radii**: A pointer to a `PhPoint_t` structure. The interpretation of the contents of this structure depends on the flags.
- **secondary**: A `PgColor_t` object that specifies the colors that you want to use for the lower and right edges of the object.
- **width**: The thickness of the lines. When you increase the thickness, the lines grow toward the middle of the beveled object.
- **flags**: Flags that control how the object is drawn; see below.
**Library:**

ph

**Description:**

These functions build a command in the draw buffer to draw a beveled rectangle or arrow. The *flags* parameter defines how the object is drawn, and includes options for defining the types of corners that it has. *PgDrawBeveled()* works on the current draw context, while you can specify the draw context for *PgDrawBeveledCx()*.

The upper and left edges of the object are drawn in the current stroke color; the lower and right edges are drawn in the *secondary* color.

The *flags* argument must be one of the following:

- **Pg_BEVEL_CLIP**
  - Draw a box with clipped corners. The *radii* argument determines how much each corner is clipped.

- **Pg_BEVEL_ROUND**
  - Draw a box with rounded corners. The *radii* argument determines how much each corner is rounded.

- **Pg_BEVEL_SQUARE**
  - Draw a box with square corners (default). The *radii* argument isn’t used with this option. The corners will look like those created by *PgDrawBevelBox()*.

- **Pg_BEVEL_AUP**
  - Draw an arrow pointing up. The *radii* argument isn’t used with this option.

- **Pg_BEVEL_ADOWN**
  - Draw an arrow pointing down. The *radii* argument isn’t used with this option.
**PgDrawBeveled(), PgDrawBeveledCx()**

**Pg**.

**BEVEL**

- **ALEFT**
  
  Draw an arrow pointing left. The \textit{radii} argument isn’t used with this option.

- **ARIGHT**
  
  Draw an arrow pointing right. The \textit{radii} argument isn’t used with this option.

You can OR the \textit{flags} argument with one of the following:

- **Pg**.
  
  - **DRAW**
    
    - **STROKE** — draw as an outline.
  - **Pg**.
    
    - **DRAW**
      
      - **FILL** — fill the box.
    - **Pg**.
      
      - **DRAW**
        
        - **FILL_STROKE** — fill the box, then stroke it.

You can also OR the \textit{flags} argument with the following:

- **Pg**.
  
  - **BEVEL**
    
    - **SET** — swap the upper-left and lower-right colors; this makes the beveled object appear to be set.

**Returns:**

- **0**  
  
  Success.

- **-1**  
  
  The draw buffer is too small to hold the current draw state and the draw command.

**Examples:**

The following example:

```c
DrawBeveled() {
    PhRect_t rc = { 8, 8, 152, 56 };
    PhRect_t rr = { 8, 64, 152, 112 };
    PhPoint_t pc = { 8, 8 };
    PhPoint_t pr = { 12, 12 };

    PgSetFillColor( Pg_GREY );
    PgSetStrokeColor( Pg_WHITE );
    PgDrawBeveled( &rc, &pc, Pg_DGREY, 2,
                   Pg_DRAW_FILL_STROKE | Pg_BEVEL_CLIP );
```

512  

Chapter 8  

Pg—Graphics
**PgDrawBeveled(), PgDrawBeveledCx()**

```
PgDrawBeveled( &rr, &pr, Pg_DGREY, 2,
              Pg_DRAW_FILL_STROKE | Pg_BEVEL_ROUND );
```

will draw:

The following example:

```
DrawBevelArrow() {
    PhRect_t rup = { 20, 4, 44, 16 };
    PhRect_t rdown = { 20, 48, 44, 60 };
    PhRect_t rleft = { 4, 20, 16, 44 };
    PhRect_t rright = { 48, 20, 60, 44 };

    PgSetFillColor( Pg_GREY );
    PgSetStrokeColor( Pg_WHITE );
    PgDrawBeveled( &rup, NULL, Pg_DGREY, 1,
                   Pg_DRAW_FILL_STROKE | Pg_BEVEL_AUP );
    PgDrawBeveled( &rdown, NULL, Pg_DGREY, 1,
                   Pg_DRAW_FILL_STROKE | Pg_BEVEL_ADOWN );
    PgDrawBeveled( &rleft, NULL, Pg_DGREY, 1,
                   Pg_DRAW_FILL_STROKE | Pg_BEVEL_ALEFT );
    PgDrawBeveled( &rright, NULL, Pg_DGREY, 1,
                   Pg_DRAW_FILL_STROKE | Pg_BEVEL_ARIGHT );
}
```

will draw:
**PgDrawBeveled(), PgDrawBeveledCx()**

**Classification:**

- Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No

**See also:**

- `PgColor_t`, `PgDrawArrow*()`, `PgDrawBevelBox*()`, `PgDrawRect*()`, `PgDrawIRect*()`, `PgDrawRoundRect*()`, `PgSetFillColor*()`, `PgSetFillDither*()`, `PgSetFillTransPat*()`, `PgSetStrokeColor*()`, `PgSetStrokeDither*()`, `PgSetStrokeTransPat*()`, `PhPoint_t`, `PhRect_t`

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
int PgDrawBezier( PhPoint_t const *ptr,
                  int num,
                  PhPoint_t const *pos,
                  int flags );

int PgDrawBezierv( PhPoint_t const *ptr,
                   int num,
                  PhPoint_t const *pos,
                  int flags );

int PgDrawBezierCx( void *dc,
                     PhPoint_t const *ptr,
                     int num,
                  PhPoint_t const *pos,
                  int flags );

int PgDrawBezierCxv( void *dc,
                     PhPoint_t const *ptr,
                     int num,
                  PhPoint_t const *pos,
                  int flags );
```

Arguments:

- **dc**  
  *PgDrawBezierCx() and PgDrawBezierv() only.* A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- **ptr**  
  An array of four `PhPoint_t` structures that define the Bézier curve.

- **num**  
  The number of points in the array (i.e. 4).
**PgDrawBezier(), PgDrawBezierv(), PgDrawBezierCx(), PgDrawBezierCxv()**

- **pos**
  A pointer to a `PhPoint_t` structure that specifies offsets to be added to all of the points or the first point, depending on the flags.

- **flags**
  Flags that control how the curve is drawn; see below.

**Library:**

```ph```

**Description:**

These functions build a command in the draw buffer to draw a multisegment Bézier curve from an array of points. `PgDrawBezier()` and `PgDrawBezierv()` work on the current draw context, while you can specify the draw context for `PgDrawBezierCx()` and `PgDrawBezierCxv()`.

Each Bézier curve is defined by 4 points. The last point of a curve becomes the first point of the next curve. The first and fourth points are anchor points; the line passes through these. The second and third points are control points; the curve is “pulled” toward these points.

The **flags** argument must be one of the following:

- **Pg_DRAW_STROKE** — draw a stroked curve.
- **Pg_DRAW_FILL** — draw a filled curve.
- **Pg_DRAW_FILL_STROKE** — draw a filled curve, then stroke it.

You can OR **flags** with any combination of the following:

- **Pg_CLOSED** — connect the last point to the first.
- **Pg_RELATIVE** — use relative coordinates to draw the curve. Each point is relative to the previous point.

For absolute coordinates, **pos** is added to each point pointed to by **ptr**. For relative coordinates, the first coordinate is the sum of **pos** and the first point of the array; any subsequent coordinate is the sum of the previous point and the next point of the array.
If you call the “v” or “Cxv” form of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the array is stored until the draw buffer is flushed. Make sure you call \texttt{PgFlush()} before you modify the point array.

**Returns:**

0 Success.

-1 The draw buffer is too small to hold the current draw state, the draw command, and the data. Increase the size of the draw buffer or decrease the number of points.

**Examples:**

The following example:

```c
void DrawFillStrokeBezier() {
    PhPoint_t o = { 0, 0 };    PhPoint_t p[] = {43, 71, -92, -18, 344, -6, 20, 99};

    PgSetStrokeDash( "\l", 1, 0x10000 );    PgSetStrokeColor( Pg_GRAY );
    PgDrawPolygon( &p, 4, &o,
                    Pg_DRAW_STROKE | Pg_CLOSED );
    PgSetStrokeDash( NULL, 0, 0 );    PgSetStrokeColor( Pg_YELLOW );
    PgSetFillColor( Pg_PURPLE );    PgDrawBeziers( &p, 4, &o,
                        Pg_DRAW_FILL_STROKE | Pg_CLOSED );
}
```

will draw:

![Image](image-url)
The dotted lines show where the control points are relative to the anchor points.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDrawPolygon*(), *PgFlush*(), *PhPoint_t*

To draw stroked Bézier curves, see also:

*PgSetStrokeColor*(), *PgSetStrokeCap*(), *PgSetStrokeDash*(), *PgSetStrokeDither*(), *PgSetStrokeJoin*(), *PgSetStrokeWidth*()

To draw filled Bézier curves, see also:

*PgSetFillColor*(), *PgSetFillDither*(), *PgSetFillTransPat*()

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PgDrawBitmap( void const *ptr,
                 int flags,
                 PhPoint_t const *pos,
                 PhPoint_t const *size,
                 int bpl,
                 long tag );

int PgDrawBitmapv( void const *ptr,
                   int flags,
                   PhPoint_t const *pos,
                   PhPoint_t const *size,
                   int bpl,
                   long tag );

int PgDrawBitmapCx( void *dc,
                    void const *ptr,
                    int flags,
                    PhPoint_t const *pos,
                    PhPoint_t const *size,
                    int bpl,
                    long tag );

int PgDrawBitmapCxv( void *dc,
                     void const *ptr,
                     int flags,
                     PhPoint_t const *pos,
                     PhPoint_t const *size,
                     int bpl,
                     long tag );
```

**Arguments:**

- `dc`  
  \textit{PgDrawBitmapCx()} and \textit{PgDrawBitmapCxv()} only. A void pointer to any type of draw context. Examples of draw contexts are:
  
  - a PhDrawContext\_t returned by \textit{PhDCCreate()}
  
  - a PmMemoryContext\_t returned by \textit{PmMemCreateMC()}
**PgDrawBitmap()**, **PgDrawBitmapv()**,  
**PgDrawBitmapCx()**, **PgDrawBitmapCxv()**

- a **PpPrintContext_t** returned by **PpCreatePC()**
- a **PdOffscreenContext_t** returned by **PdCreateOffscreenContext()**

**ptr** A pointer to the bitmap data.

**flags** Drawing flags. Can be 0 or **Pg_BACK_FILL** (see below).

**pos** The starting position for the bitmap.

**size** The size of the bitmap.

**bpl** The number of bytes per line of image data (that is, the offset from one line of data to the next).

**tag** Used for data caching by programs such as **phrelay** (see the QNX Neutrino **Utilities Reference**). This argument is ignored if you set it to 0. To calculate a tag value, use **PtCRC()**.

**Library:**

**ph**

**Description:**

These functions build a command in the draw buffer to draw a bitmap. The function starts the bitmap at **pos** and extends it down and to the right according to **size**.

**PgDrawBitmap()** and **PgDrawBitmapv()** work on the current draw context, while you can specify the draw context for **PgDrawBitmapCx()** and **PgDrawBitmapCxv()**.

To calculate the size of the data transferred to the graphics driver, multiply **bpl** by **size.y**. You can determine the **size** and **bpl** arguments with the value returned by a **PxLoadImage()** call.

The data pointed to by **ptr** is one bit per pixel. If the pixel value is 1, the pixel is drawn with the color set by **PgSetTextColor()** or **PgSetTextDither()**. If the pixel value is 0, the pixel is drawn as
transparent unless you’ve set flags to Pg_BACK_FILL. With
Pg_BACK_FILL, the pixel is drawn with the color set by
PgSetFillColor() or PgSetFillDither(). The pixels are drawn most
significant bit first.

If you call the “v” or “Cxv” form of this function, the data isn’t
physically copied into the draw buffer. Instead, a pointer to the array
is stored until the draw buffer is flushed. Make sure you call
PgFlush() before you modify the bitmap.

If the data is in shared memory, the mx form of this function will
automatically pass a shared memory reference instead of the bitmap.

Returns:

0   Successful completion

-1  The draw buffer is too small to hold the current draw state, the
draw command, and one pixel line of the image. Increase the
size of the draw buffer or decrease the width of the image.

Examples:

The following example:

PhPoint_t TestBitmapSize = { 64, 64 };
int TestBitmapBPL = 8;
char TestBitmap[64*8] = { "512 bytes of bitmap data" };

DrawSimpleBitmap() {
    PhPoint_t p = { 8, 8 };

    PgSetTextColor( Pg_WHITE );
    PgDrawBitmap( TestBitmap, 0, &p,
                 &TestBitmapSize, TestBitmapBPL, 0 );
}

will draw:
The following example:

```c
DrawBackFillBitmap() {
    PhPoint_t p = { 8, 8 };
    PgSetTextColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawBitmap( TestBitmap, Pg_BACK_FILL, &p,
                  &TestBitmapSize, TestBitmapBPL, 0 );
}
```

will draw:

![Ph photon](QNX_Photon.png)

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDrawRepBitmap*(), *PgFlush*(), *PgSetFillColor*(),
*PgSetFillDither*(), *PgSetTextColor*(), *PgSetTextDither*(),
*PgShmemCreate*(), *PhPoint_t*, *PtCRC*(), *PxLoadImage*()

“Drawing attributes” and “Bitmaps” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawEllipse()**, **PgDrawEllipseCx()**

*Draw an ellipse*

**Synopsis:**

```c
int PgDrawEllipse( PhPoint_t const *center,
                   PhPoint_t const *radii,
                   unsigned int flags );
```

```c
int PgDrawEllipseCx( void *dc,
                     PhPoint_t const *center,
                     PhPoint_t const *radii,
                     unsigned int flags );
```

**Arguments:**

- **dc** *(PgDrawEllipseCx() only)*: A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- **center**: Defines the ellipse’s center.

- **radii**: Defines the ellipse’s x and y radii.

- **flags**: Must be one of the following:
  - `Pg_DRAW_STROKE` — draw as a line.
  - `Pg_DRAW_FILL` — fill the ellipse.
  - `Pg_DRAW.FILL_STROKE` — fill the ellipse, then stroke it.

To have the function interpret the `center` and `radii` arguments as the upper-left and lower-right coordinates respectively, OR `flags` with `Pg_EXTENT_BASED`. 
**PgDrawEllipse(), PgDrawEllipseCx()**

**Library:**

ph

**Description:**

These functions build a command in the draw buffer to draw an ellipse. *PgDrawEllipse()* works on the current draw context, while you can specify the draw context for *PgDrawEllipseCx()*.

**Returns:**

0    Success.

-1    The draw buffer is too small to hold the current draw state and the draw command.

**Examples:**

The following example:

```c
DrawStrokeElli() {
    PhPoint_t c = { 80, 60 };
    PhPoint_t r = { 72, 52 };
    PgSetStrokeColor( Pg_WHITE );
    PgDrawEllipse( &c, &r, Pg_DRAW_STROKE );
}
```

will draw:

![Ellipse](image)

The following example:

```c
DrawFillElli() {
    PhPoint_t c = { 80, 60 };
    PhPoint_t r = { 72, 52 };
    PgSetFillColor( Pg_WHITE );
    PgFillEllipse( &c, &r, Pg_DRAW_STROKE );
}
```
**PgDrawEllipse(), PgDrawEllipseCx()**

```c
PhPoint_t r = { 72, 52 };  
PgSetFillColor( Pg_PURPLE );  
PgDrawEllipse( &c, &r, Pg_DRAW_FILL );
}
```

will draw:

![Ellipse](image)

The following example:

```c
DrawFillStrokeEllipsis() {
    PhPoint_t c = { 80, 60 };  
    PhPoint_t r = { 72, 52 };  
    PgSetFillColor( Pg_PURPLE );  
    PgSetStrokeColor( Pg_WHITE );  
    PgDrawEllipse( &c, &r, Pg_DRAW_FILL_STROKE );
}
```

will draw:

![Ellipse](image)

**Classification:**

Photon
**PgDrawEllipse(), PgDrawEllipseCx()**

**Safety**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDrawRoundRect*(), *PgSetFillColor*(), *PgSetFillDither*(), *PgSetFillTransPat*(), *PgSetStrokeColor*(), *PgSetStrokeDither*(), *PgSetStrokeTransPat*(), **PhPoint_t**

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawGradient(), PgDrawGradientCx()**

Ask the graphics driver to render a gradient

**Synopsis:**

```c
int PgDrawGradient( PhPoint_t *ul,
                     PhPoint_t *lr,
                     unsigned long gradient_type,
                     unsigned long transition_type,
                     unsigned long num_color_pts,
                     PgColor_t color1,
                     PgColor_t color2,
                     PgColor_t color3,
                     PgColor_t color4,
                     unsigned long table_size,
                     unsigned char *transition_table );
```

```c
int PgDrawGradientCx( void *dc,
                       PhPoint_t *ul,
                       PhPoint_t *lr,
                       unsigned long gradient_type,
                       unsigned long transition_type,
                       unsigned long num_color_pts,
                       PgColor_t color1,
                       PgColor_t color2,
                       PgColor_t color3,
                       PgColor_t color4,
                       unsigned long table_size,
                       unsigned char *transition_table );
```

**Arguments:**

- `dc` *PgDrawGradientCx() only.* A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`
**PgDrawGradient(), PgDrawGradientCx()**

*ul, lr*  
Pointers to `PhPoint_t` structures that define the upper left and lower right corners of the rendering rectangle.

*gradient_type*  
The type of gradient:

- `Pg_GRAD_HORIZONTAL`
- `Pg_GRAD_VERTICAL`
- `Pg_GRAD_DIAGF` — forward diagonal.
- `Pg_GRAD_DIAGB` — backward diagonal.
- `Pg_GRAD_4POINT` — four-point gradient (`Pg_GRAD_DIAGF` and `Pg_GRAD_DIAGB` mixed together).
- `Pg_GRAD_BOX_DIAGF` — boxy forward diagonal.
- `Pg_GRAD_BOX_DIAGB` — boxy backward diagonal.
- `Pg_GRAD_BOX_4POINT` — boxy four-point.

Boxy Gradients are similar to the nonboxy gradients, but are rendered using rectangles instead of horizontal lines while scaling the gradient information to the rendered rectangle. As a result, it takes less time to render boxy gradients. They don’t look as precise as the nonboxy versions, but look cool anyway. ;-)

*transition_type*  
One of the following:

- `Pg_GRAD_LINEAR` — the color of the isochrome lines changes linearly from the starting color to the end color.
- `Pg_GRAD_HILL` — the color of the isochrome lines changes from the starting color to the end color and back to the starting color again. The end color is reached in the middle isochrome line.
**PgDrawGradient(), PgDrawGradientCx()**

- **Pg_GRAD_HILL2** — similar to **Pg_GRAD_HILL**, except there are two transitions from the starting color to the end color.
- **Pg_GRAD_EXP** — the color of the isochrome lines changes exponentially from the starting color to the end color.
- **Pg_GRAD_TABLE** — the transition from the starting color to the end color is controlled by the `transition_table` and `table_size` arguments.

`num_color_pts` The resolution of the gradient. Basically it’s the number of colors you want to have the driver calculate between the endpoint colors.

`color1, color2, color3, color4` PgColor_t values that define the color endpoints. The `color3` and `color4` arguments are used only in four-point gradients.

`table_size` The size of the user-defined transition table. This needs to be set only if the transition type is **Pg_GRAD_TABLE**.

`transition_table` A pointer to the user-defined transition table. This needs to be set only if the transition type is **Pg_GRAD_TABLE**.

**Library:**

**ph**

**Description:**

These functions request the graphics driver to render a gradient. **PgDrawGradient()** works on the current draw context, while you can specify the draw context for **PgDrawGradientCx()**.
**PgDrawGradient**, **PgDrawGradientCx**

**Returns:**

0  Success.
-1  An error occurred.

**Examples:**

```c
// Draw a basic horizontal gradient from blue to red,
// with 20 colors in a rectangle of size 100 x 200

PhRect_t GradRect={{0,0},{100,200}};

PgDrawGradient(&GradRect.ul,&GradRect.lr,Pg_GRAD_HORIZONTAL,
    Pg_GRAD_LINEAR,20,Pg_RED,Pg_BLUE,0,0,0,NULL);
```

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgCalcColorContrast*, **PgColor_t**, **PhPoint_t**

“Gradients” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PgDrawGradientBevelBox( PhPoint_t *ul,
    PhPoint_t *lr,
    PgColor_t light_color,
    PgColor_t ul_flat_color,
    PgColor_t lr_flat_color,
    PgColor_t dark_color,
    short depth,
    short width,
    PgColor_t outline_color,
    PgColor_t inline_color,
    int flags );
```

```c
int PgDrawGradientBevelBoxCx( void *dc,
    PhPoint_t *ul,
    PhPoint_t *lr,
    PgColor_t light_color,
    PgColor_t ul_flat_color,
    PgColor_t lr_flat_color,
    PgColor_t dark_color,
    short depth,
    short width,
    PgColor_t outline_color,
    PgColor_t inline_color,
    int flags );
```

**Arguments:**

- `dc`: *PgDrawGradientCx() only*. A void pointer to any type of draw context. Examples of draw contexts are:

  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`
**PgDrawGradientBevelBox(), PgDrawGradientBevelBoxCx()**

- **ul, lr**
  Pointers to `PhPoint_t` structures that define the upper left and lower right corners of the beveled box.

- **light_color**
  A `PgColor_t` that defines the lightest color in the bevel gradient.

- **ul_flat_color**
  The middle/neutral color in the bevel gradient of the top and left edges. This is also the fill color of the box if `Pg_BVB_FILL` is set in the `flags`.

- **lr_flat_color**
  The middle/neutral color in the bevel gradient of the bottom and right edges.

- **dark_color**
  The darkest color in the bevel gradient.

- **depth**
  The number of isochrome lines in the light-to-flat and flat-to-dark bevel gradients. The sign of the depth controls the location of an (imaginary) light source:

  - Positive depth — illuminated from the top left.
  - Negative depth — illuminated from the bottom right.

- **width**
  The total width of the beveled border, including outlines and inlines.

- **outline_color**
  The color of the outline rectangle.

- **inline_color**
  The color of the inline rectangle.
These functions draw a beveled box with gradients.  
*PgDrawGradientBevelBox()* works on the current draw context, while you can specify the draw context for *PgDrawGradientBevelBoxCx()*.

The only difference between these functions and *PgBevelBox()* and *PgBevelBoxCx()* is that in the latter, the upper-left and the lower-right flat colors are identical.

The bits in the *flags* argument affect the appearance of the beveled box:

```
Pg_BVB_FILL Fill the beveled box with *ul_flat_color*.
Pg_BVB_FULL_GRADIENTS
  Each bevel has two gradients: light-to-flat and flat-to-dark. If this isn’t set, the bevels have only one gradient; see the illustration above.
Pg_BVB_DRAW_ALL_INLINE
Pg_BVB_DRAW_BOTTOM_INLINE
Pg_BVB_DRAW_LEFT_INLINE
Pg_BVB_DRAW_RIGHT_INLINE
Pg_BVB_DRAW_TOP_INLINE
  Draw the indicated side or sides of the inline rectangle.
Pg_BVB_DRAW_ALL_OUTLINE
Pg_BVB_DRAW_BOTTOM_OUTLINE
Pg_BVB_DRAW_LEFT_OUTLINE
Pg_BVB_DRAW_RIGHT_OUTLINE
Pg_BVB_DRAW_TOP_OUTLINE
  Draw the indicated side or sides of the outline rectangle.
```
**PgDrawGradientBevelBox(), PgDrawGradientBevelBoxCx()**

Pg_BVB_DRAW_ALL_BV
Pg_BVB_DRAW_BOTTOM Pg_BVB_DRAW_LEFT
Pg_BVB_DRAW_RIGHT
Pg_BVB_DRAW_TOP

Draw the indicated edge or edges of the beveled box.

Pg_BVB_DRAW_ALL

Draw all bevels, inlines, and outlines of the beveled box.

Pg_BVB_DRAW_BITS

Turn on all of the above flag bits.

**Returns:**

0 Success.

-1 An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgBevelBox*(), *PgColor_t*, *PgContrastBevelBox*(), *PhPoint_t*

“Gradients” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawGrid(), PgDrawGridCx()**

*Draw a grid*

Synopsis:

```c
int PgDrawGrid( PhRect_t const *r,
                PhPoint_t const *g );

int PgDrawGridCx( void *dc,
                   PhRect_t const *r,
                   PhPoint_t const *g );
```

Arguments:

- **dc** *PgDrawGridCx() only.* A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- **r** A pointer to a `PhRect_t` structure that defines the upper left and lower right corners of the grid box.

- **g** A pointer to a `PhPoint_t` structure that defines the number of divisions in the grid. The number of lines drawn equals the number of divisions plus 1.

Library:

- `ph`

Description:

These functions draw a rectangular grid. *PgDrawGrid()* works on the current draw context, while you can specify the draw context for *PgDrawGridCx()*.

These functions build a draw command to draw the grid. The size of the grid is defined by the `r` argument with `g.x+1` vertical lines and `g.y+1` horizontal lines. If `g.x` is 0, no vertical lines are drawn; if `g.y` is 0, no horizontal lines are drawn.
**PgDrawGrid(), PgDrawGridCx()**

**Returns:**

0  Success.

-1  The draw buffer is too small to hold the current draw state and
the draw command.

**Examples:**

The following example uses `PgDrawGrid()` to make a grid of 8
squares by 8 squares; each square is 8 by 8 pixels:

```c
void GridStandard() {
    PhRect_t r = { 8, 8, 72, 72 };  
    PhPoint_t g = { 8, 8 };  

    PgSetStrokeColor( Pg_WHITE );  
    PgDrawGrid( &r, &g );  
}
```

This code draws:

![Grid Standard](image)

The following example uses `PgDrawGrid()` to generate 20 ticks.
Every 5th tick is made larger by calling `PgDrawGrid()` again with
different parameters:

```c
void GridTicks() {
    PhRect_t r = { 8, 24, 108, 28 };  
    PhPoint_t g = { 20, 0 };  

    PgSetStrokeColor( Pg_WHITE );  
    PgDrawGrid( &r, &g );  
    r.ul.y-=1;  
    r.lr.y+=1;  
    g.x=4;  
    PgSetStrokeWidth( 3 );  
    PgSetStrokeCap( Pg_POINT_CAP );  
    PgDrawGrid( &r, &g );  
    PgSetStrokeWidth( 0 );
}
```
**PgDrawGrid(), PgDrawGridCx()**

This code draws:

```
---
---
---
---
---
---
---
---
---
---
---
---
```

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgSetStrokeCap*(), PgSetStrokeColor*(), PgSetStrokeDash*(), PgSetStrokeDither*(), PgSetStrokeJoin*(), PgSetStrokeTransPat*(), PgSetStrokeWidth*(), PgSetStrokeXORColor*(), PhPoint_t, PhRect_t`

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawImage(), PgDrawImagev(), PgDrawImageCx(), PgDrawImageCxv()**

**Draw an image**

**Synopsis:**

```c
int PgDrawImage( void const *ptr,
        int type,
        PhPoint_t const *pos,
        PhDim_t const *size,
        int bpl,
        long tag );

int PgDrawImagev( void const *ptr,
        int type,
        PhPoint_t const *pos,
        PhDim_t const *size,
        int bpl,
        long tag );

int PgDrawImageCx( void *dc,
        void const *ptr,
        int type,
        PhPoint_t const *pos,
        PhDim_t const *size,
        int bpl,
        long tag );

int PgDrawImageCxv( void *dc,
        void const *ptr,
        int type,
        PhPoint_t const *pos,
        PhDim_t const *size,
        int bpl,
        long tag );
```

**Library:**

```
ph
```

**Description:**

These functions build a command in the draw buffer to draw an image. The functions start the image at `pos` and extend it down and to the right according to the dimensions specified by the `PhDim_t` structure pointed to by `size`. 

538   Chapter 8 • Pg—Graphics
**PgDrawImage(), PgDrawImagev(), PgDrawImageCx(), PgDrawImageCxv()**

*PgDrawImage()* and *PgDrawImageCx()* don’t draw an image if there isn’t enough room in the draw buffer to store at least one scan line. *PgDrawImagev()* and *PgDrawImageCxv()* act the same way if the image isn’t in shared memory.

*PgDrawImage()* and *PgDrawImagev()* work on the current draw context, while you can specify the draw context *dc* for *PgDrawImageCx()* and *PgDrawImageCxv()*.

Instead of using these functions, we recommend using a *PhImage_t* structure and calling one of the *PgDrawPhImage*() functions. These functions automatically handle palettes, transparency, and so on.

The *bpl* argument indicates the number of bytes per line of image data (that is, the offset from one line of data to the next).

To calculate the size of the data transferred to the graphics driver, multiply *bpl* by *size.y*. You can determine the *size* and *bpl* arguments with the value returned by *PxLoadImage()*.

The *tag* argument is used for data caching by programs such as *phrelay* (see the QNX Neutrino Utilities Reference). To calculate the tag, use *PtCRC()*). This argument is ignored if you set it to 0.

The *type* argument controls how the image data pointed to by *ptr* is interpreted by the graphics driver. For information on the possible types, see *PhImage_t*.

If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the array is stored until the draw buffer is flushed. Make sure you call *PgFlush()* before you modify the image.

If the data is in shared memory, the mx form of this function will automatically pass a shared memory reference instead of the image.
**PgDrawImage(), PgDrawImagev(), PgDrawImageCx(), PgDrawImageCxv()**

**Returns:**

0  Success.

-1  The draw buffer is too small to hold the current draw state, the draw command, and one pixel line of the image. Increase the size of the draw buffer or decrease the width of the image.

**Examples:**

The following example:

```c
PhImage_t  *pimage;

InitPalImage() {
  if (pimage != NULL) return;
  if ((pimage = PxLoadImage( "mackface.bmp", NULL )) == NULL) {
    perror( "Unable to load image" );
    return;
  }
}

DrawPalImage() {
  PhPoint_t  p = { 0, 0 };

  InitPalImage();
  if (pimage == NULL) return;
  if ((pimage->palette != NULL) && (pimage->colors > 0))
    PgSetPalette( pimage->palette, 0, 0, pimage->colors,
                     Pg_PALSET_SOFT, 0 );
  PgDrawImage( pimage->image, pimage->type, &p,
               &pimage->size, pimage->bpl, 0 );
}
```

will draw:

![Image Example](image.png)
**Classification:**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PgDrawImage*()`, `PgDrawImagev()`, `PgDrawImageCx()`, `PgDrawImageCxv()`,
- `PgFlush*()`, `PgSetFillColor*()`, `PgSetPalette*()`, `PgShmemCreate()`,
- `PhDim_t`, `PhImage_t`, `PhMakeTransBitmap()`,
- `PhMakeTransparent()`, `PhPoint_t`, `PtCRC()`, `PxLoadImage()`

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawLine(), PgDrawILine(), PgDrawLineCx(), PgDrawILineCx()**

**Draw a single line**

**Synopsis:**

```c
int PgDrawLine( PhPoint_t const *p1,
                PhPoint_t const *p2 );

int PgDrawILine( int x1, int y1,
                 int x2, int y2 );

int PgDrawLineCx( void *dc,
                  PhPoint_t const *p1,
                  PhPoint_t const *p2 );

int PgDrawILineCx( void *dc,
                  int x1, int y1,
                  int x2, int y2 );
```

**Library:**

ph

**Description:**

These functions build a command in the draw buffer to draw a line. Note that `PgDrawLine()` and `PgDrawLineCx()` require two pointers to `PhPoint_t` structures, whereas `PgDrawILine()` and `PgDrawILineCx()` take individual arguments.

`PgDrawLine()` and `PgDrawLineCx()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawLineCx()` and `PgDrawILineCx()`.

**Returns:**

- **0**  Success.
- **-1**  The draw buffer is too small to hold the current draw state and the draw command.
**PgDrawLine(), PgDrawILine(), PgDrawLineCx(), PgDrawILineCx()**

**Examples:**

The following example:

```c
DrawLines() {
    PgSetStrokeColor( Pg_RED );
    PgDrawILine( 8, 8, 152, 8 );
    PgSetStrokeColor( Pg_GREEN );
    PgDrawILine( 8, 8, 152, 60 );
    PgSetStrokeColor( Pg_YELLOW );
    PgDrawILine( 8, 8, 152, 112 );
}
```

will draw:

![Line drawing example](image)

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgSetStrokeCap*(), *PgSetStrokeColor*(), *PgSetStrokeDash*(),
*PgSetStrokeDither*(), *PgSetStrokeJoin*(), *PgSetStrokeWidth*(),
*PhPoint_t*

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw
Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()**

**Synopsis:**

```c
int PgDrawMultiTextArea( char *text,
                        int len,
                        PhRect_t *canvas,
                        int text_flags,
                        int canvas_flags,
                        int linespacing );

int PgDrawMultiTextAreaCx( void *dc,
                            char *text,
                            int len,
                            PhRect_t *canvas,
                            int text_flags,
                            int canvas_flags,
                            int linespacing );
```

**Arguments:**

- **dc** 
  PgDrawMultiTextAreaCx() only. A void pointer to any type of draw context. Examples of draw contexts are:
  - a PhDrawContext_t returned by PhDCCreate()
  - a PmMemoryContext_t returned by PmMemCreateMC()
  - a PpPrintContext_t returned by PpCreatePC()
  - a PdOffscreenContext_t returned by PdCreateOffscreenContext()

- **text** 
  The multiline text string to be drawn.

- **len** 
  The number of characters to draw. If this argument is 0, all the characters are drawn.

- **canvas** 
  A pointer to a PhRect_t structure that defines the area into which the text is to be drawn.
**PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()**

*text_flags* Flags that affect how the text is drawn within the text-extent rectangle:

- Pg_TEXT_LEFT — left align the text.
- Pg_TEXT_RIGHT — right align the text.
- Pg_TEXT_CENTER — horizontally center the text.
- Pg_BACK_FILL — backfill the text extent rect with the currently set fill color.

*canvas_flags* Flags that affect how the text-extent rectangle is aligned within the canvas:

- Pg_TEXT_LEFT
- Pg_TEXT_RIGHT
- Pg_TEXT_CENTER — horizontally center the text-extent rectangle within the canvas.
- Pg_TEXT_TOP
- Pg_TEXT_BOTTOM
- Pg_TEXT_MIDDLE — vertically center the text-extent rectangle within the canvas.

*lineseSpacing* The leading (spacing) between lines, in pixels. A positive linespacing has the obvious effect: increased spacing between lines, and a taller extent. A negative linespacing causes the function to compute an extent for overlapping lines. Larger negative line spacings make the extent decrease in height. The minimum height of the extent is the height of the current font.

**Library:**

ph
**PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()**

**Description:**

These functions draw multiline text within an area called a canvas, using the font specified by a previous call to `PgSetFont()`.

These functions call `PgExtentMultiText()` to compute the extent of the text. Text can be aligned within the text-extent rectangle, and the text-extent rectangle itself can be aligned within the canvas.

**Returns:**

0  Success.

-1  An error occurred.

**Examples:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <Pt.h>

void MultiTextDraw( PtWidget_t *widget,
                    PhTile_t *damage )
{
    PhRect_t canvas;
    int r;
    char Helvetica14b[MAX_FONT_TAG];

    char s[100] = " clever \n is \n not he who wins \n \
 but he who \n wins \n easily ";

    // Find the size of the canvas on which the text
    // will be drawn
    PtCalcCanvas(widget, &canvas);

    // Paint the canvas red
    PgSetFillColor( Pg_RED );
    PgDrawRect( &canvas, Pg_DRAW_FILL );

    // Set the fill color, text color, and font.
    PgSetFillColor( Pg_BLUE );
    PgSetTextColor( Pg_WHITE );

    if(PfGenerateFontName("Helvetica", PF_STYLE_BOLD, 14,
                          Helvetica14b) == NULL) {
```
PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()

```c
perror("Unable to find font");
} else {
    PgSetFont( Helvetica14b );
}

// Draw multiline text. Note the text-extent and
// canvas flags, and the linespacing.

r = PgDrawMultiTextArea( s, 0, &canvas,
    Pg_TEXT_RIGHT|Pg_BACK_FILL,
    Pg_TEXT_CENTER|Pg_TEXT_MIDDLE, 10 );

if ( r == -1 )
    fprintf( stderr, "\n Error."
);

int main(int argc, char **argv)
{
    PtWidget_t *base;
    PtArg_t args[2];
    PhDim_t dim;

    // Initialize Photon and create a base window

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    dim.w = 250; dim.h = 250;
    PtSetArg( &args[0], Pt_ARG_DIM, &dim, 0 );

    if ((base = PtCreateWidget(PtWindow, Pt_NO_PARENT,
        1, args)) == NULL)
        PteXit(EXIT_FAILURE);

    // Create a raw widget parented to the base window

    PtSetArg( &args[1], Pt_ARG_RAW_DRAW_F,
        (long) MultiTextDraw, 0 );
    PtCreateWidget( PtRaw, base, 2, args );

    // Realize the base window. This will realize the
    // raw widget, which in turn will draw itself with
    // the MultiTextDraw() function above.

    PtRealizeWidget(base);

    PtMainLoop();
    return EXIT_SUCCESS;
}
```
**PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()**

This code produces the following output:

![Output Image](image)

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler No</td>
<td></td>
</tr>
<tr>
<td>Signal handler No</td>
<td></td>
</tr>
<tr>
<td>Thread No</td>
<td></td>
</tr>
</tbody>
</table>

**See also:**

-PgDrawString*(), PgDrawText*(), PgDrawTextArea*(),
-PgExtentMultiText*(), PgFillColor*(), PgFont*(),
-PgTextColor*(), PgTextDither*(), PgTextTransPat*(),
-PgTextXORColor*(), PgUnderline*(), PhRect_t
PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()

“Text” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**PgDrawPhImage(), PgDrawPhImagev(), PgDrawPhImageCx(), PgDrawPhImageCxv()**

*Draw an image that's contained in a PhImage_t structure*

**Synopsis:**

```c
int PgDrawPhImage( PhPoint_t const *pos,
                   PhImage_t const *image,
                   int flags );
```

```c
int PgDrawPhImagev( PhPoint_t const *pos,
                    PhImage_t const *image,
                    int flags );
```

```c
int PgDrawPhImageCx( void *dc,
                      PhPoint_t const *pos,
                      PhImage_t const *image,
                      int flags );
```

```c
int PgDrawPhImageCxv( void *dc,
                       PhPoint_t const *pos,
                       PhImage_t const *image,
                       int flags );
```

**Library:**

ph

**Description:**

These functions draw the provided image at the position specified in the PhPoint_t structure pointed to by *pos*. The image parameter must be a pointer to a PhImage_t structure that defines the image to be rendered.

If the image has a transparency mask, it's used. These functions set the palettes for palette-based images, and apply alpha blending if the alpha member of the PhImage_t structure isn't NULL.

You can pass the following bit in the flags argument:

Pg_GHOST or Pt_GHOST

Render the image using the ghost bitmap as a transparency mask.
If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the image is stored until the draw buffer is flushed. Make sure you call `PgFlush()` before you modify the image.

If the data is in shared memory, the “v” forms of this function automatically pass a shared memory reference instead of the image.

`PgDrawPhImage()` and `PgDrawPhImagev()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawPhImageCx()` and `PgDrawPhImageCxv()`.

**Returns:**

0 Success.

-1 The draw buffer couldn’t be resized enough to fit a single scan line of the image (insufficient memory).

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`ApGetImageRes()`, `PgDrawPhImageRect*v()`,
`PgDrawRepPhImage*()`, `PgFlush*()`, `PhCreateImage()`, `PhImage_t`,
`PhMakeGhostBitmap()`, `PhMakeTransBitmap()`,
\textbf{PgDrawPhImage()}, \textbf{PgDrawPhImagev()}, \textbf{PgDrawPhImageCx()}, \textbf{PgDrawPhImageCxv()}

\textit{PhMakeTransparent(), PhPoint_t, PhReleaseImage(), PmMemCreateMC(), PmMemFlush(), PxLoadImage()}

“Images” in the Raw Drawing and Animation chapter of the Photon \textit{Programmer’s Guide}
**PgDrawPhImageRectv(), PgDrawPhImageRectCxv()**

*Draw part of an image that's contained in a PhImage_t structure*

**Synopsis:**

```c
int PgDrawPhImageRectv( PhPoint_t const *pos,
                        PhImage_t const *image,
                        PhRect_t const *rect,
                        int flags );

int PgDrawPhImageRectCxv( void *dc,
                          PhPoint_t const *pos,
                          PhImage_t const *image,
                          PhRect_t const *rect,
                          int flags );
```

**Library:**

`ph`

**Description:**

These functions draw a rectangular piece (specified by the PhRect_t structure pointed to by `rect`) of the provided image at position `pos`. The `image` parameter must be a pointer to a PhImage_t structure that defines the image to be rendered.

`PgDrawPhImageRectv()` works on the current draw context, while you can specify the draw context `dc` for `PgDrawPhImageRectCxv()`.

If the image has a transparency mask, it’s used. These functions set the palettes for palette-based images, and apply alpha blending if the `alpha` member of the PhImage_t structure isn’t NULL.

The currently defined bits for the `flags` parameter are:

- **Pg_GHOST** or **Pt_GHOST**
  
  Render the image using the ghost bitmap as a transparency mask.

The rectangle is clipped to the image boundaries (i.e. the area bounded by `(0,0)` to `(image->size.w - 1, image->size.h - 1)`). If `rect` is NULL, the entire image is drawn.
The drawing of the piece of the image begins at \( pos \) (i.e. the \( rect \) argument doesn’t introduce an additional offset.

For image formats where the number of pixels per byte is greater than 1 (e.g. \( \text{Pg}_\text{IMAGE}_\text{GRADIENT}_\text{NIBBLE} \), \( \text{Pg}_\text{IMAGE}_\text{PALETTE}_\text{NIBBLE} \), \( \text{Pg}_\text{BITMAP}_\text{BACKFILL} \), and \( \text{Pg}_\text{BITMAP}_\text{TRANSPARENT} \)), the portion specified by \( rect \) might be grown horizontally so that it falls on even byte boundaries.

**Returns:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>-1</td>
<td>The draw buffer couldn’t be resized enough to fit a single scan line of the image (insufficient memory).</td>
</tr>
</tbody>
</table>

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- \( \text{ApGetImageRes()} \), \( \text{PgDrawPhImage*()} \), \( \text{PgDrawRepPhImage*()} \), \( \text{PhCreateImage()} \), \( \text{PhImage_t} \), \( \text{PhMakeGhostBitmap()} \), \( \text{PhMakeTransBitmap()} \), \( \text{PhMakeTransparent()} \), \( \text{PhPoint_t} \), \( \text{PhRect_t} \), \( \text{PhReleaseImage()} \), \( \text{PmMemCreateMC()} \), \( \text{PmMemFlush()} \), \( \text{PxLoadImage()} \)

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawPixel(), PgDrawIPixel(), PgDrawPixelCx(), PgDrawIPixelCx()**

**Draw a point**

**Synopsis:**

```c
int PgDrawPixel( PhPoint_t const *pt );
int PgDrawIPixel( int x, int y );
int PgDrawPixelCx( void *dc, PhPoint_t const *pt );
int PgDrawIPixelCx( void *dc, int x, int y );
```

**Library:**

`ph`

**Description:**

These functions build a command in the draw buffer to draw a pixel. For `PgDrawPixel*()`, the `pt` argument points to a `PhPoint_t` structure that defines the pixel location; for `PgDrawIPixel*()`, `x` and `y` specify the location.

`PgDrawPixel()` and `PgDrawIPixel()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawPixelCx()` and `PgDrawIPixelCx()`.

**Returns:**

0  Successful completion

-1  The draw buffer is too small to hold the current draw state, the draw command, and the data. Increase the size of the draw buffer.

**Classification:**

Photon
**PgDrawPixel(), PgDrawIPixel(), PgDrawPixelCx(), PgDrawIPixelCx()**

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDrawPixelArray*(), *PgFlush*(), *PgSetStrokeColor*(), *PgSetStrokeDither*(), *PhPoint_t*

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PgDrawPixelArray( PhPoint_t const *ptr,
                       int num,
                       PhPoint_t const *pos );

int PgDrawPixelArrayv( PhPoint_t const *ptr,
                        int num,
                        PhPoint_t const *pos );

int PgDrawPixelArrayCx( void *dc,
                         PhPoint_t const *ptr,
                         int num,
                         PhPoint_t const *pos );

int PgDrawPixelArrayCxv( void *dc,
                          PhPoint_t const *ptr,
                          int num,
                          PhPoint_t const *pos );
```

**Library:**

ph

**Description:**

These functions build a command in the draw buffer to draw an array of pixels. The `ptr` argument points to an array of pixel locations; `num` indicates how many points to draw; and the value of `pos` is added to every pixel location.

`PgDrawPixelArray()` and `PgDrawPixelArrayv()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawPixelArrayCx()` and `PgDrawPixelArrayCxv()`. 
If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the array is stored until the draw buffer is flushed. Make sure you call `PgFlush()` or `PgFlushCx()` before you modify the pixel array.

**Returns:**

0 Success.

-1 The draw buffer is too small to hold the current draw state, the draw command, and the data. Increase the size of the draw buffer or decrease the number of points.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgDrawPixel()*(), *PgDrawIPixel()*(), *PgFlush()*(), *PgSetStrokeColor()*(), *PgSetStrokeDither()*(), *PhPoint_t*

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgDrawPolygon(), PgDrawPolygonv(), PgDrawPolygonCx(), PgDrawPolygonCxv()**

*Draw a stroked and/or filled polygon*

**Synopsis:**

```c
int PgDrawPolygon( PhPoint_t const *ptr,
                    int num,
                    PhPoint_t const *pos,
                    int flags );

int PgDrawPolygonv( PhPoint_t const *ptr,
                    int num,
                    PhPoint_t const *pos,
                    int flags );

int PgDrawPolygonCx( void *dc,
                      PhPoint_t const *ptr,
                      int num,
                      PhPoint_t const *pos,
                      int flags );

int PgDrawPolygonCxv( void *dc,
                       PhPoint_t const *ptr,
                       int num,
                       PhPoint_t const *pos,
                       int flags );
```

**Library:**

`ph`

**Description:**

These functions build a command in the draw buffer to draw a polygon from an array of points, pointed to by `ptr`, with `num` entries. The array of points must fit in the draw buffer for all these functions.

*PgDrawPolygon()* and *PgDrawPolygonv()* work on the current draw context, while you can specify the draw context `dc` for *PgDrawPolygonCx()* and *PgDrawPolygonCxv()*.

The `flags` argument must be one of the following:
**PgDrawPolygon()**, **PgDrawPolygonv()**, **PgDrawPolygonCx()**, **PgDrawPolygonCxv()**

- **Pg_DRAW_STROKE**
  - Draw a stroked polygon.

- **Pg_DRAW_FILL**
  - Draw a filled polygon.

- **Pg_DRAW_FILL_STROKE**
  - Draw a filled polygon, then stroke it.

You can OR flags with any combination of the following:

- **PgCLOSED**
  - Connect the last point to the first.

- **PgRELATIVE**
  - Use relative coordinates to draw the polygon.
  - Each point is relative to the previous point.

For absolute coordinates, *pos* is added to each point pointed to by *ptr*. For relative coordinates, the first coordinate is the sum of *pos* and the first point of the array; any subsequent coordinate is the sum of the previous point and the next point of the array.

If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the array is stored until the draw buffer is flushed. Make sure you call *PgFlush()* or *PgFlushCx()* before you modify the point array.

**Returns:**

- **0** Successful completion
- **-1** The draw buffer is too small to hold the current draw state, the draw command, and the data. Increase the size of the draw buffer or decrease the number of points.
Examples:

The following example:

```c
DrawFillStrokePoly() {
    PhPoint_t o = { 0, 0 };
    PhPoint_t p[] = { 80, 8, 120, 120, 16, 32, 
                     136, 32, 40, 120 };

    PgSetStrokeColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawPolygon( &p, 5, &o, Pg_DRAW_FILL_STROKE |
                   Pg_CLOSED );
}
```

will draw:

![Star](image)

The following example:

```c
DrawRelPoly() {
    PhPoint_t o = { 0, 0 };
    PhPoint_t p[] = { 80, 8, 40, 112, -104, -88, 
                     120, 0, -96, 88 };

    PgSetStrokeColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawPolygon( &p, 5, &o, Pg_DRAW_FILL_STROKE |
                   Pg_CLOSED |
                   Pg_RELATIVE );
}
```
The following example:

```c
DrawUnclosedPoly() {
    PhPoint_t o = { 0, 0 };
    PhPoint_t p[] = { 80, 8, 120, 120, 16, 32,
                     136, 32, 40, 120 };

    PgSetStrokeColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawPolygon( &p, 5, &o, Pg_DRAW_FILL_STROKE );
}
```

will draw:

The following example:

```c
DrawUnclosedPoly() {
    PhPoint_t o = { 0, 0 };
    PhPoint_t p[] = { 80, 8, 120, 120, 16, 32,
                     136, 32, 40, 120 };

    PgSetStrokeColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawPolygon( &p, 5, &o, Pg_DRAW_FILL_STROKE );
}
```

will draw:

**Classification:**

Photon

**Safety**

Interrupt handler  No

continued...
PgDrawPolygon(), PgDrawPolygonv(), PgDrawPolygonCx(), PgDrawPolygonCxv()

Safety

<table>
<thead>
<tr>
<th>Signal handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgFlush(), PhPoint_t

To draw stroked polygons, see also:
PgSetStrokeColor*(), PgSetStrokeCap*(), PgSetStrokeDash*(), PgSetStrokeDither*(), PgSetStrokeJoin*(), PgSetStrokeWidth*()

To draw filled polygons, see also:
PgSetFillColor*(), PgSetFillDither*(), PgSetFillTransPat*()

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawRect(), PgDrawIRect(), PgDrawRectCx(), PgDrawIRectCx()**

*Draw a rectangle*

**Synopsis:**

```c
int PgDrawRect( PhRect_t const *rect,
                 unsigned int flags );

int PgDrawIRect( int ulx, int uly,
                 int lrx, int lry,
                 unsigned int flags );

int PgDrawRectCx( void *dc,
                  PhRect_t const *rect,
                  unsigned int flags );

int PgDrawIRectCx( void *dc,
                  int ulx, int uly,
                  int lrx, int lry,
                  unsigned int flags );
```

**Library:**

`ph`

**Description:**

These functions build a command in the draw buffer to draw a rectangle. Note that `PgDrawRect*()` requires a pointer to a `PhRect_t` structure, whereas `PgDrawIRect*()` takes individual arguments.

`PgDrawRect()` and `PgDrawIRect()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawRectCx()` and `PgDrawIRectCx()`.

The `flags` argument must be one of the following:

- **Pg_DRAW_STROKE**
  
  Draw as a line.

- **Pg_DRAW_FILL**
  
  Fill the rectangle.

- **Pg_DRAW_FILL_STROKE**
  
  Fill the rectangle, then stroke it.
For `PgDrawIRect*()` , `ulx` and `uly` are the coordinates of the upper-left corner, and `lrx` and `lry` are those for the lower-right.

**Returns:**

0  Success.

-1  The draw buffer is too small to hold the current draw state and the draw command.

**Examples:**

The following example:

```plaintext
DrawFillRect() {  
    PgSetFillColor( Pg_PURPLE );  
    PgDrawIRect( 8, 8, 152, 112, Pg_DRAW_FILL );  
}
```

will draw:

![Example draw fill rectangle]

The following example:

```plaintext
DrawStrokeRect() {  
    PgSetStrokeColor( Pg_WHITE );  
    PgDrawIRect( 8, 8, 152, 112, Pg_DRAW_STROKE );  
}
```
\textit{PgDrawRect(), PgDrawIRect(), PgDrawRectCx(), PgDrawIRectCx()} will draw:

\texttt{
DrawFillStrokeRect() {
    PgSetFillColor( Pg\_PURPLE );
    PgSetStrokeColor( Pg\_WHITE );
    PgDrawIRect( 8, 8, 152, 112, Pg\_DRAW\_FILL\_STROKE );
}
}

will draw:

\textbf{Classification:}

Photon

\begin{tabular}{|l|c|}
\hline
\textbf{Safety} &  \\
\hline
Interrupt handler & No  \\
Signal handler & No  \\
Thread & No  \\
\hline
\end{tabular}
See also:

PgDrawBeveled*(), PgDrawRoundRect*(), PgSetFillColor*(), PgSetFillDither*(), PgSetFillTransPat*(), PhRect_t

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PgDrawRepBitmap( void const *ptr,
    int flags,
    PhPoint_t const *pos,
    PhPoint_t const *size,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );
```

```c
int PgDrawRepBitmapv( void const *ptr,
    int flags,
    PhPoint_t const *pos,
    PhPoint_t const *size,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );
```

```c
int PgDrawRepBitmapCx( void *dc,
    void const *ptr,
    int flags,
    PhPoint_t const *pos,
    PhPoint_t const *size,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );
```

```c
int PgDrawRepBitmapCxv( void *dc,
    void const *ptr,
    int flags,
    PhPoint_t const *pos,
    PhPoint_t const *size,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );
```
These functions build a command in the draw buffer to repeatedly draw the bitmap pointed to by `ptr`. For an explanation of bitmaps, see `PgDrawBitmap()`.

`PgDrawRepBitmap()` and `PgDrawRepBitmapv()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawRepBitmapCx()` and `PgDrawRepBitmapCxv()`.

These functions:

- Draw the first bitmap at `pos` and the next bitmap `space.x` pixels to the right.
- Draw the bitmap `rep.x` times in the x direction and `rep.y` times in the y direction.
- Draw each row of bitmaps `space.y` pixels below the previous row.

If you OR `flags` with `Pg_REPB ALT ER NATE`, all the bitmaps are drawn twice; the second time the position is offset by half of `space.x` and `space.y`.

The `bpl` argument indicates the number of bytes per line of image data (that is, the offset from one line of data to the next). To calculate the size of the data transferred to the graphics driver, multiply `bpl` by `size.y`. You can determine the `size` and `bpl` arguments with the value returned by `PxLoadImage()`.

The data pointed to by `ptr` is one bit per pixel. If the pixel value is 1, the pixel is drawn with the color set by `PgSetTextColor*()` or `PgSetTextDither*()`. If the pixel value is 0, the pixel is drawn as transparent unless you’ve set `flags` to `Pg_BACK_FILL`. With `Pg_BACK_FILL`, the pixel is drawn with the color set by `PgSetFillColor*()` or `PgSetFillDither*()`. The pixels are drawn most significant bit first.
The tag argument is used for data caching by programs such as phrelay (see the QNX Neutrino Utilities Reference). To calculate the tag, use PtCRC(). This argument is ignored if you set it to 0.

If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the bitmap is stored until the draw buffer is flushed. Make sure you call PgFlush() before you modify the bitmap.

If the data is in shared memory, the mx form of this function automatically passes a shared memory reference instead of the bitmap.

Returns:

0   Success.
-1   The draw buffer is too small to hold the current draw state, the draw command, and one pixel line of the image. Increase the size of the draw buffer, or decrease the width of the image.

Examples:

The following example:

```c
DrawRepBitmap() {
    PhPoint_t p = { -32, -32 };
    PhPoint_t rep = { 5, 3 };

    PgSetTextColor( Pg_WHITE );
    PgDrawRepBitmap( TestBitmap, 0, &p,
                     &TestBitmapSize, TestBitmapBPL,
                     &rep, &TestBitmapSize, 0 );
}
```

draws:
The following example:

```c
DrawAltRepBitmap() {
    PhPoint_t p = { 0, 0 };  
    PhPoint_t rep = { 3, 2 };  
    PhPoint_t space;

    space.x = TestBitmapSize.x * 2;  
    space.y = TestBitmapSize.y * 2;

    PgSetTextColor( Pg_WHITE );  
    PgDrawRepBitmap( TestBitmap, Pg_REPBMAP_ALTERNATE, &p,  
                     &TestBitmapSize, TestBitmapBPL,  
                     &rep, &space, 0 );
}
```

draws:

![Example Image](image)

**Classification:**

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No
See also:

`PgDrawBitmap*()`, `PgDrawRepImage*()`, `PgFlush*()`, `PgSetFillColor*()`, `PgSetFillDither*()`, `PgSetTextColor*()`, `PgSetTextDither*()`, `PhPoint_t`, `PtCRC()`, `PxLoadImage()

“Drawing attributes” and “Bitmaps” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PgDrawRepImage( void const *ptr,
    int flag,
    PhPoint_t const *pos,
    PhPoint_t const *area,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );

int PgDrawRepImagev( void const *ptr,
    int flag,
    PhPoint_t const *pos,
    PhPoint_t const *area,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );

int PgDrawRepImageCx( void *dc,
    void const *ptr,
    int flag,
    PhPoint_t const *pos,
    PhPoint_t const *area,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );

int PgDrawRepImageCxv( void *dc,
    void const *ptr,
    int flag,
    PhPoint_t const *pos,
    PhPoint_t const *area,
    int bpl,
    PhPoint_t const *rep,
    PhPoint_t const *space,
    long tag );
```

`PgDrawRepImage()`, `PgDrawRepImagev()`, `PgDrawRepImageCx()`, `PgDrawRepImageCxv()`

Draw an image several times
**PgDrawRepImage(), PgDrawRepImagev(), PgDrawRepImageCx(), PgDrawRepImageCxv()**

**Library:**

`ph`

**Description:**

These functions build a command in the draw buffer to repeatedly draw the image pointed to by `ptr`. These functions:

- Draw the first image at `pos` and the next image `space.x` pixels to the right.
- Draw the image `rep.x` times in the x direction and `rep.y` times in the y direction.
- Draw each row of images `space.y` pixels below the previous row.

`PgDrawRepImage()` and `PgDrawRepImagev()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawRepImageCx()` and `PgDrawRepImageCxv()`.

If you OR `flags` with `Pg_REPBMM_ALTERNATE`, all the images are drawn twice; the second time the position is offset by half of `space.x` and `space.y`.

The image formats are as described for `PhImage_t`. Specify the image format by ORing it into the `flags` argument.

The `bpl` argument indicates the number of bytes per line of image data (that is, the offset from one line of data to the next). To calculate the size of the data transferred to the graphics driver, multiply `bpl` by `size.y`. You can determine the `size` and `bpl` arguments with the value returned by `PxLoadImage()`.

The `tag` argument is used for data caching by programs such as `phrelay` (see the QNX Neutrino Utilities Reference). To calculate the tag, use `PtCRC()`. This argument is ignored if you set it to 0.
If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the image is stored until the draw buffer is flushed. Make sure you call `PgFlush()` or `PgFlushCx()` before you modify the image.

If the data is in shared memory, the “v” forms of this function automatically passes a shared memory reference instead of the image.

**Returns:**

0  Success.

-1  The draw buffer is too small to hold the current draw state, the draw command, and one pixel line of the image. Increase the size of the draw buffer, or decrease the width of the image.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PgDrawImage*()`, `PgDrawRepBitmap*()`, `PgDrawRepPhImage*()`, `PgFlush*()`, `PhImage_t`, `PhPoint_t`, `PtCRC()`, `PxLoadImage()`

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawRepPhImage**(*)

Repeatedly draw an image stored in a PhImage_t structure

**Synopsis:**

```c
int PgDrawRepPhImage( PhImage_t const *image,  
                      int flags,  
                      PhPoint_t const *pos,  
                      PhPoint_t const *rep,  
                      PhPoint_t const *space );

int PgDrawRepPhImagev( PhImage_t const *image,  
                       int flags,  
                       PhPoint_t const *pos,  
                       PhPoint_t const *rep,  
                       PhPoint_t const *space );

int PgDrawRepPhImageCx( void *dc,  
                        PhImage_t const *image,  
                        int flags,  
                        PhPoint_t const *pos,  
                        PhPoint_t const *rep,  
                        PhPoint_t const *space );

int PgDrawRepPhImageCxv( void *dc,  
                         PhImage_t const *image,  
                         int flags,  
                         PhPoint_t const *pos,  
                         PhPoint_t const *rep,  
                         PhPoint_t const *space );
```

**Library:**

ph

**Description:**

These functions unite the convenience of *PgDrawPhImage()* and the ability to tile images of *PgDrawRepImage()* or *PgDrawRepBitmap()*.

These functions draw:

- the first image at pos and the next image space.x pixels to the right
**PgDrawRepPhImage***

- the image `rep.x` times in the x direction and `rep.y` times in the y direction
- each row of images `space.y` pixels below the previous row.

The `pos`, `rep`, and `space` arguments all point to structures of type `PhPoint_t`.

You can pass any combination of the following bits in the `flags` argument:

- `Pg_GHOST` or `Pt_GHOST`
  - Render the image using the ghost bitmap as a transparency mask.

- `Pg_REPBm_ALTERNATE`
  - Draw all the images twice; the second time the position is offset by half of `space.x` and `space.y`.

If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the image is stored until the draw buffer is flushed. Make sure you call `PgFlush()` or `PgFlushCx()` before you modify the image.

If the data is in shared memory, the “v” form of this function automatically passes a shared memory reference instead of the image.

*PgDrawRepPhImage*() and *PgDrawRepPhImagev*() work on the current draw context, while you can specify the draw context `dc` for *PgDrawRepPhImageCx*() and *PgDrawRepPhImageCxv*().

**Returns:**

- **0** Success.
- **-1** The draw buffer is too small to hold the current draw state, the draw command, and one pixel line of the image. Increase the size of the draw buffer, or decrease the width of the image.
PgDrawRepPhImage*()

Classification:
Photon

Safety
Interrupt handler  No
Signal handler  No
Thread  No

See also:
ApGetImageRes(), PgDrawPhImage*(), PgDrawPhImageRect*(), PgDrawRepImage*(), PgDrawRepBitmap*(), PgFlush*(), PhCreateImage(), PhImage_t, PhMakeGhostBitmap(), PhMakeTransBitmap(), PhMakeTransparent(), PhPoint_t, PhReleaseImage(), PmMemCreateMC(), PmMemFlush(), PxLoadImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawRoundRect(), PgDrawRoundRectCx()**

Draw a rounded rectangle

**Synopsis:**

```c
int PgDrawRoundRect( PhRect_t const *rect,
                      PhPoint_t const *radii,
                      unsigned flags );
```

```c
int PgDrawRoundRectCx( void *dc,
                        PhRect_t const *rect,
                        PhPoint_t const *radii,
                        unsigned flags );
```

**Library:**

ph

**Description:**

These functions build a command in the draw buffer to draw a rounded rectangle. The `rect` argument is a pointer to a `PhRect_t` structure that defines the extent of the rectangle. The `radii` is a pointer to a `PhPoint_t` structure that defines the roundness of the corners, in pixels.

The `flags` argument must be one of the following:

- Pg_DRAW_STROKE — draw as an outline.
- Pg_DRAW_FILL — fill the rectangle.
- Pg_DRAW_FILL_STROKE — fill the rectangle, then stroke it.

Since the value of `radii` is truncated to the size of the rectangle, you should find this function useful for drawing ellipses within a rectangular area (see example below).

`PgDrawRoundRect()` works on the current draw context, while you can specify the draw context `dc` for `PgDrawRoundRectCx()`.

Chapter 8 • Pg—Graphics 579
Returns:

0 Success.

-1 The draw buffer is too small to hold the current draw state and the draw command.

Examples:

The following example:

```c
DrawStrokeRRect() {
    PhRect_t rect = { 8, 8, 152, 112 };  
    PhPoint_t radii = { 32, 32 }; 

    PgSetStrokeColor( Pg_WHITE );
    PgDrawRoundRect( &rect, &radii, Pg_DRAW_STROKE );
}
```

will draw:

![Circle](image)

The following example:

```c
DrawFillRRect() {
    PhRect_t rect = { 8, 8, 152, 112 }; 
    PhPoint_t radii = { 32, 32 }; 

    PgSetFillColor( Pg_PURPLE );
    PgDrawRoundRect( &rect, &radii, Pg_DRAW_FILL );
}
```
**PgDrawRoundRect(), PgDrawRoundRectCx()**

will draw:

![Image](image)

The following example:

```c
DrawFillStrokeRRect() {
    PhRect_t rect = { 8, 8, 152, 112 };
    PhPoint_t radii = { 1000, 1000 };

    PgSetFillColor( Pg_PURPLE );
    PgSetStrokeColor( Pg_WHITE );
    PgDrawRoundRect( &rect, &radii, Pg_DRAW_FILL_STROKE );
}  
```

will draw:

![Image](image)

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
</tbody>
</table>

*continued...*
PgDrawRoundRect(), PgDrawRoundRectCx()

Safety

| Thread | No |

See also:

PgDrawBeveled*, PgDrawIRect*, PgDrawRect*, PgSetFillColor*, PgSetFillDither*, PgSetFillTransPat*, PhPoint_t, PhRect_t

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PgDrawSpan( PgSpan_t const *ptr,
                  int num,
                  PhPoint_t const *pos,
                  int flags );

int PgDrawSpanv( PgSpan_t const *ptr,
                  int num,
                  PhPoint_t const *pos,
                  int flags );

int PgDrawSpanCx( void *dc,
                  PgSpan_t const *ptr,
                  int num,
                  PhPoint_t const *pos,
                  int flags );

int PgDrawSpanCxv( void *dc,
                   PgSpan_t const *ptr,
                   int num,
                   PhPoint_t const *pos,
                   int flags );
```

Library:

`ph`

Description:

These low-level draw primitives let you render complex shapes not supported by the Photon graphics drivers.

The functions draw a list of spans. The spans are defined as a list of `PgSpan_t` records. Here are the members of `PgSpan_t`:

- `short x1`   starting x position
- `short x2`   last x position
- `short y`    y position
The number of spans is defined by the `num` parameter. The location of the spans is offset by the `pos` parameter.

You can set `flags` to one of the following:

- `Pg_DRAW_FILL` — draw with fill parameters.
- `Pg_DRAW_STROKE` — draw with stroke parameters.
- `Pg_DRAW_TEXT` — draw with text parameters.

If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the list of spans is stored until the draw buffer is flushed. Make sure you call `PgFlush()` or `PgFlushCx()` before you modify the list.

`PgDrawSpan()` and `PgDrawSpanv()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawSpanCx()` and `PgDrawSpanCxv()`.

Returns:

- **0** Success.
- **-1** The draw buffer is too small to hold the current draw state, the draw command and the data. Increase the size of the draw buffer, or decrease the number of points.

Examples:

The following example:

```c
void DrawSpan() {
    PgSpan_t spans[152];
    PgSpan_t *sp = spans;
    PhPoint_t p = { 12, 10 };
    int i, v, n=0;

    for (i=0; i<=100; i++) {
        sp->x1 = (i*i)>>6;
        v = 100 - i;
        sp->x2 = 160 - ((v*v)>>6);
    }
}
```
\begin{verbatim}
sp->y = i;
sp++; n++;
}
for (i=0; i<=50; i++) {
    sp->x1 = 100 - ((i*i)>>6);
    v = 50 - i;
    sp->x2 = 60 + ((v*v)>>6);
    sp->y = i+25;
    sp++; n++;
}
PgSetFillColor( Pg_WHITE );
PgDrawSpan( spans, n, &p, Pg_DRAW_FILL );
\end{verbatim}

will draw:

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

\texttt{PgDrawPolygon*()}, \texttt{PgFillColor*()}, \texttt{PgStrokeColor*()},
\texttt{PgTextColor*()}, \texttt{PgFlush*()}, \texttt{PhPoint_t}
PgDrawSpan(), PgDrawSpanv(), PgDrawSpanCx(), PgDrawSpanCxv()

“Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawString(), PgDrawStringv(), PgDrawStringCx(), PgDrawStringCxv()**

*Draw a string of characters*

**Synopsis:**

```c
int PgDrawString( char const *ptr,
                  PhPoint_t const *pos );

int PgDrawStringv( char const *ptr,
                    PhPoint_t const *pos );

int PgDrawStringCx( void *dc,
                     char const *ptr,
                     PhPoint_t const *pos );

int PgDrawStringCxv( void *dc,
                      char const *ptr,
                      PhPoint_t const *pos );
```

**Library:**

```
ph
```

**Description:**

These convenience functions for `PgDrawText*()` calculate the length of the string internally using `strlen()`. They then pass the string, along with its length, its position, and a flags setting of 0, to the corresponding text function.

If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the string is stored until the draw buffer is flushed. Make sure you call `PgFlush()` before you modify the text.

`PgDrawString()` and `PgDrawStringv()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawStringCx()` and `PgDrawStringCxv()`.
Returns:

0  Success.
-1  The draw buffer is too small to hold the current draw state, the
    draw command, and the data. Increase the size of the draw
    buffer, or decrease the size of the string.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

\texttt{PgDrawMultiTextArea*(),PgDrawText*(),PgDrawTextArea*(),}
\texttt{PgFlush*(),PgSetFillColor*(),PgSetFillDither*(),}
\texttt{PgSetFillTransPat*(),PgSetFont*(),PgSetTextColor*(),}
\texttt{PgSetTextDither*(),PgSetTextTransPat*(),PgSetTextXORColor*(),}
\texttt{PgSetUnderline*(),PhPoint_t}

“Text” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide

\texttt{strlen()} in the QNX Neutrino Library Reference

588  Chapter 8  •  Pg—Graphics
Synopsis:

```
int PgDrawText( char const *ptr,
                int len,
                PhPoint_t const *pos,
                int flags);

int PgDrawTextv( char const *ptr,
                int len,
                PhPoint_t const *pos,
                int flags);

int PgDrawTextChars( char const *ptr,
                    int len,
                    PhPoint_t const *pos,
                    int flags);

int PgDrawTextCx( void *dc,
                char const *ptr,
                int len,
                PhPoint_t const *pos,
                int flags);

int PgDrawTextvCx( void *dc,
                char const *ptr,
                int len,
                PhPoint_t const *pos,
                int flags);

int PgDrawTextCharsCx( void *dc,
                    char const *ptr,
                    int len,
                    PhPoint_t const *pos,
                    int flags);
```

Library:

ph
Description:

Each of these functions builds a command in the draw buffer to draw the text indicated by \textit{ptr} at location \textit{pos}, using the font specified in a previous call to \textit{PgSetFont()}. The \textit{len} parameter specifies the number of bytes required to store the string. For pure ASCII strings (characters 0 to 127), this is the number of characters. For multibyte strings, \textit{len} may be larger than the number of characters. For double-byte strings, \textit{len} is twice the number of characters.

By default, the function assumes that all strings consist of multibyte characters that conform to the ISO/IEC 10646-1 UTF-1 multibyte format. However, if \textit{Pg\_TEXT\_WIDECHAR} is set, the function assumes each character is represented by 2 bytes that conform to the ISO/IEC 10646-1 UCS-2 double-byte format.

\textit{PgDrawTextChars()} assumes that \textit{len} is the number of characters to draw. Using this number, \textit{PgDrawTextChars()} determines the number of bytes required to store the string.

<table>
<thead>
<tr>
<th>In order to:</th>
<th>You can:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define the color of the text</td>
<td>Use \textit{PgSetTextColor()}, \textit{PgSetTextDither()}, or \textit{PgSetTextXORColor()}</td>
</tr>
<tr>
<td>Mask the text</td>
<td>Use \textit{PgSetTextTransPat()}</td>
</tr>
<tr>
<td>Fill the extent of the text</td>
<td>Set the background color with \textit{PgSetFillColor()} or \textit{PgSetFillDither()} and specify the \textit{Pg_BACK_FILL} flag to \textit{PgDrawText()} or \textit{PgDrawTextv()}</td>
</tr>
<tr>
<td>Underline the text</td>
<td>Use \textit{PgSetUnderline()}</td>
</tr>
</tbody>
</table>

By default, the text is left-aligned (\textit{Pg\_TEXT\_LEFT}), and the text is drawn with \textit{pos->y} as its baseline. You can set \textit{flags} to a combination of:
**`PgDrawText*(), PgDrawTextChars*()`**

- **Pg_BACK_FILL**
  
  Fill the text extent with fill-color parameters.

- **Pg_TEXT_WIDECCHAR**
  
  The text is specified as wide characters. Each character is represented by 16 bits.

- **Pg_TEXT_LEFT**
  
  Left align text to `pos` (text is drawn to the right).

- **Pg_TEXT_RIGHT**
  
  Right align text to `pos` (text is drawn to the left).

- **Pg_TEXT_CENTER**
  
  Center text horizontally on `pos`.

- **Pg_TEXT_TOP**
  
  Top align text to `pos` (text is drawn below).

- **Pg_TEXT_BOTTOM**
  
  Bottom align text to `pos` (text is drawn above).

- **Pg_TEXT_MIDDLE**
  
  Center text vertically on `pos`.

---

*Text justification relative to the indicated positions.*
**PgDrawText*(), PgDrawTextChars*()**

If you call the “v” forms of these functions, the data isn’t physically copied into the draw buffer. Instead, a pointer to the string is stored until the draw buffer is flushed. Make sure you call `PgFlush()` or `PgFlushCx()` before you modify the text.

`PgDrawText()`, `PgDrawTextv()`, and `PgDrawTextChars()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawTextCx()`, `PgDrawTextCxv()`, and `PgDrawTextCharsCx()`.

**Returns:**

0 Success.

-1 The draw buffer is too small to hold the current draw state, the draw command and the data. Increase the size of the draw buffer, or decrease the size of the string.

**Examples:**

```c
DrawSimpleText() {
    char *s = "Hello World!";
    PhPoint_t p = { 8, 30 };
    char Helvetica18[MAX_FONT_TAG];

    if(PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
        perror("Unable to find font");
    } else {
        PgSetFont( Helvetica18 );
    }
    PgSetTextColor( Pg_WHITE );
    PgDrawText( s, strlen( s ), &p, 0 );
}
```

The above code draws:

![Hello World!](image)

```c
DrawBackFillText() {
    char *s = "Hello World!";
    PhPoint_t p = { 8, 30 };
```
**PgDrawText*, *PgDrawTextChars***

```c
char Helvetica18[MAX_FONT_TAG];

if(PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
    perror("Unable to find font");
} else {
    PgSetFont( Helvetica18 );
    PgSetTextColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgDrawText( s, strlen( s ), &p, Pg_BACK_FILL );
}
```

The above code draws:

```
Hello World!
```

```c
DrawUnderlineText() {
    char *s = "Hello World!";
    PhPoint_t p = { 8, 30 };
    char Helvetica18[MAX_FONT_TAG];

    if(PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
        perror("Unable to find font");
    } else {
        PgSetFont( Helvetica18 );
        PgSetTextColor( Pg_WHITE );
        PgSetUnderline( Pg_RED, Pg_TRANSPARENT, 0 );
        PgDrawText( s, strlen( s ), &p, 0 );
        PgSetUnderline( Pg_TRANSPARENT, Pg_TRANSPARENT, 0 );
    }
}
```

The above code draws:

```
Hello World!
```

```c
DrawBackFillUnderlineText() {
    char *s = "Hello World!";
    PhPoint_t p = { 8, 30 };
    char Helvetica18[MAX_FONT_TAG];
```
**PgDrawText**(*), **PgDrawTextChars**(*)

```c
if(PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
    perror("Unable to find font");
} else {
    PgSetFont( Helvetica18 );
    PgSetTextColor( Pg_WHITE );
    PgSetFillColor( Pg_PURPLE );
    PgSetUnderline( Pg_RED, Pg_TRANSPARENT, 0 );
    PgDrawText( s, strlen( s ), &p, Pg_BACK_FILL );
    PgSetUnderline( Pg_TRANSPARENT, Pg_TRANSPARENT, 0 );
}
```

The above code draws:

Hello World!

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgDrawMultiTextArea*()`, `PgDrawString*()`, `PgFlush*()`, `PgSetFillColor*()`, `PgSetFillDither*()`, `PgSetFillTransPat*()`, `PgSetFont*()`, `PgSetTextColor*()`, `PgSetTextDither*()`, `PgSetTextTransPat*()`, `PgSetTextXORColor*()`, `PgSetUnderline*()`, `PhPoint_t`

“Text” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgDrawTextArea(), PgDrawTextAreaCx()**

*Draw text within an area*

**Synopsis:**

```c
int PgDrawTextArea( char const *ptr,
                   int len,
                   PhRect_t const *rect,
                   int flags );

int PgDrawTextAreaCx( void *dc,
                       char const *ptr,
                       int len,
                       PhRect_t const *rect,
                       int flags );
```

**Library:**

ph

**Description:**

These functions draw text within an area, using the font specified by a previous call to `PgSetFont*()`. This area is clipped to the dimensions of the rectangle specified by the `PhRect_t` structure pointed to by `rect`.

By default, the text is left-aligned (Pg_TEXT_LEFT), and the text is drawn with its baseline centered inside the drawing area.

The `flags` can be a combination of:

- **Pg_BACK_FILL**
  
  Fill the text extent with fill-color parameters.

- **Pg_TEXT_WIDECHAR**
  
  The text is specified as wide characters. Each character is represented by 16 bits.

- **Pg_TEXT_ELLIPSIS**
  
  If the text doesn’t fit into the specified rectangle, draw an ellipsis (…) instead of part of the text, in accordance with the text’s alignment. For example, if the text is left-aligned, draw
\textbf{PgDrawTextArea()}, \textbf{PgDrawTextAreaCx()}\par

the ellipsis instead of the end of the text; if the text is right-aligned, draw the ellipsis instead of the beginning.

\textbf{Pg\_TEXT\_ELLIPSIS\_MIDDLE}

Draw the ellipsis in the middle of the string. You must also set \textbf{Pg\_TEXT\_ELLIPSIS}.

\textbf{Pg\_TEXT\_ELLIPSIS\_INVERT}

Invert the ellipsis location. You must also set \textbf{Pg\_TEXT\_ELLIPSIS}.

\textbf{Pg\_TEXT\_LEFT}

Align text to left edge of \textit{rect} (rect->ul.l).

\textbf{Pg\_TEXT\_RIGHT}

Align text to right edge of \textit{rect} (rect->lr.r).

\textbf{Pg\_TEXT\_CENTER}

Center text horizontally within \textit{rect}.

\textbf{Pg\_TEXT\_TOP}

Align text to top edge of \textit{rect} (rect->ul.y).

\textbf{Pg\_TEXT\_BOTTOM}

Align text to bottom edge of \textit{rect} (rect->lr.y).

\textbf{Pg\_TEXT\_MIDDLE}

Center text vertically within \textit{rect}.

\textit{PgDrawTextArea()} works on the current draw context, while you can specify the draw context \textit{dc} for \textit{PgDrawTextAreaCx()}.\par

\textbf{Returns:}\par

0 Success.

-1 An error occurred.
**PgDrawTextArea(), PgDrawTextAreaCx()**

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**Caveats:**

`PgDrawTextArea()` doesn’t work in any context that involves the render library, such as printing or Phindows. If your application needs to use the render library, you should:

1. Calculate the position at which to print the text, based on the extent of the text and the desired alignment inside the drawing area.
2. Set a clipping rectangle with `PgSetUserClip*()` or `PgSetUserClipAbsolute*()`.
3. Call `PgDrawText()` to display the text.

**See also:**

`PgDrawMultiTextArea*()`, `PgDrawString*()`, `PgDrawText*()`, `PgFlush*()`, `PgSetFillColor*()`, `PgSetFillDither*()`, `PgSetFillTransPat*()`, `PgSetFont*()`, `PgSetTextColor*()`, `PgSetTextDither*()`, `PgSetTextTransPat*()`, `PgSetTextXORColor*()`, `PgSetUnderline()`, `PhRect_t`

“Text” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PgDrawTImage( void const *ptr,
                  int type,
                  PhPoint_t const *pos,
                  PhDim_t const *size,
                  int bpl,
                  long tag,
                  void const *TransPtr,
                  int TransBPL);

int PgDrawTImagev( void const *ptr,
                   int type,
                   PhPoint_t const *pos,
                   PhDim_t const *size,
                   int bpl,
                   long tag,
                   void const *TransPtr,
                   int TransBPL);

int PgDrawTImageCx( void *dc,
                    void const *ptr,
                    int type,
                    PhPoint_t const *pos,
                    PhDim_t const *size,
                    int bpl,
                    long tag,
                    void const *TransPtr,
                    int TransBPL);

int PgDrawTImageCxv( void *dc,
                     void const *ptr,
                     int type,
                     PhPoint_t const *pos,
                     PhDim_t const *size,
                     int bpl,
                     long tag,
                     void const *TransPtr,
                     int TransBPL);
```

*Draw an image with a transparency mask*
**Library:**

ph

**Description:**

These functions build a command in the draw buffer to draw an image with a transparency mask. These functions take the same parameters as `PgDrawImage*()` with two additions, `TransPtr` and `TransBPL`.

Instead of using these functions, we recommend using a `PhImage_t` structure and calling a `PgDrawPhImage*()` function. These functions automatically handle palettes, transparency, and so on.

The `TransPtr` argument points to a bitmap that’s `TransBPL` bytes wide. This defines a bitmap that only allows the image to draw where there’s a value of 1 in the bitmap. Any value of 0 in the bitmap prevents the image from drawing. The leftmost pixel corresponds to the top bit of the first byte in the mask.

`PgDrawTImage()` and `PgDrawTImagev()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawTImageCx()` and `PgDrawTImageCxv()`.

**Returns:**

0 Successful completion
-1 The draw buffer is too small to hold the current draw state, the draw command, and one pixel line of the image. Increase the size of the draw buffer or decrease the width of the image.

**Classification:**

Photon
PgDrawTImage(), PgDrawTImagev(), PgDrawTImageCx(), PgDrawTImageCxv()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgDrawImage*(), PgDrawPhImage*(), PgDrawRepPhImage*(), PhDim_t, PhImage_t, PhMakeTransBitmap(), PhMakeTransparent(), PhPoint_t

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PgDrawTrend( short const *ptr,
                 PhPoint_t const *pos,
                 int num,
                 int delta,
                 int buflen,
                 int bufoff,
                 unsigned flags );

int PgDrawTrendv( short const *ptr,
                  PhPoint_t const *pos,
                  int num,
                  int delta,
                  int buflen,
                  int bufoff,
                  unsigned flags );

int PgDrawTrendCx( void *dc,
                   short const *ptr,
                   PhPoint_t const *pos,
                   int num,
                   int delta,
                   int buflen,
                   int bufoff,
                   unsigned flags );

int PgDrawTrendCxv( void *dc,
                    short const *ptr,
                    PhPoint_t const *pos,
                    int num,
                    int delta,
                    int buflen,
                    int bufoff,
                    unsigned flags );
```

**Library:**

`ph`
DESCRIPTION:

These functions build a command in the draw buffer to draw a trend graph.

The ptr argument points to an array of short values. Each of these values is used as either the x or the y coordinate; the other coordinate is calculated by adding delta to the previous value.

For example, if flags is Pg_TREND_VERT, the coordinates would be calculated as follows:

\[
\begin{align*}
\text{pos.x} &+ \text{*(ptr+0)}), \text{pos.y} + (\text{delta} \times 0) \\
\text{pos.x} &+ \text{*(ptr+1)}, \text{pos.y} + (\text{delta} \times 1) \\
\text{pos.x} &+ \text{*(ptr+2)}, \text{pos.y} + (\text{delta} \times 2) \\
\ldots
\text{pos.x} &+ \text{*(ptr+num-1)}, \text{pos.y} + (\text{delta} \times (\text{num}-1))
\end{align*}
\]

The pos argument defines the origin of the trend graph and the num argument controls the number of values to be drawn.

The flags argument controls how the trend will be drawn. It must be one of the following:

- **Pg_TREND_HORIZ**
  
  Draw a horizontal graph. The ptr values become the y axis and the delta values are added to the x coordinate.

- **Pg_TREND_VERT**
  
  Draw a vertical graph. The ptr values become the x axis and the delta values are added to the y coordinate.

The buflen and bufoff arguments aren’t currently used, and must be set to 0 for future compatibility.
If you call the “v” forms of this function, the data isn’t physically copied into the draw buffer. Instead, a pointer to the array is stored until the draw buffer is flushed. Make sure you call `PgFlush()` before you modify the array.

If the data is in shared memory, the mx form of this function will automatically pass a shared memory reference instead of the array.

`PgDrawTrend()` and `PgDrawTrendv()` work on the current draw context, while you can specify the draw context `dc` for `PgDrawTrendCx()` and `PgDrawTrendCxv()`.

**Returns:**

0   Success.

-1   The draw buffer is too small to hold the current draw state, the draw command and the data. Increase the size of the draw buffer, or decrease the number of points.

**Examples:**

The following example:

```c
void HTrend() {
    short data[512];
    PhPoint_t p = { 0, 80 };
    int i;

    for (i=0; i<512; i++) data[i] = (i & 127) - 64;
    PgSetStrokeColor( Pg_WHITE );
    PgDrawTrend( &data, &p, 256, 2, 0, 0,
                  Pg_TREND_HORIZ );
}
```
\textbf{Classification:}

Photon

\begin{tabular}{l|l}
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}

\textbf{See also:}

\texttt{PgDrawPolygon*()}, \texttt{PgSetStrokeColor*()}, \texttt{PgSetStrokeWidth*()}, \texttt{PgFlush*()}, \texttt{PhPoint\_t}

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon \textit{Programmer’s Guide}
**Synopsis:**

```c
PhRect_t *PgExtentMultiText( PhRect_t *extent,
    PhPoint_t *pos,
    char *font,
    char *str,
    unsigned n,
    int linespacing );
```

**Library:**

`ph`

**Description:**

This function determines the extent that would be occupied if `str` were rendered at `pos`, using `font` (which you should create by calling `PfGenerateFontName()`). The extent information is stored in the `PhRect_t` structure pointed to by `extent`.

The function calculates the extent of the first `n` characters of the string. To calculate the extent of the entire string, set `n` to zero. If you pass `pos` as NULL, it’s assumed to be (0,0).

The `linespacing` is in pixels. A positive linespacing has the obvious effect: increased spacing between lines, and a taller extent. A negative linespacing causes the function to compute an extent for overlapping lines. Larger negative linespacing make the extent decrease in height. The minimum height of the extent is the height of `font`.

**Returns:**

The same pointer as `extent`, or NULL if an error occurred.

**Examples:**

The following fragment determines the extent of a multiline string drawn in 14-point Helvetica. The characters in neighboring lines overlap by 4 pixels.

```c
PhRect_t extent;
char Helvetica14[MAX_FONT_TAG];
```
if(PfGenerateFontName("Helvetica", 0, 14, Helvetica14) == NULL) {
    perror("Unable to find font");
} else {
    if (PgExtentMultiText( &extent, NULL, Helvetica14, 
      "First line\nSecond line\nThird line", 0, -4))
        printf("Width: %d Height: %d\n", 
            extent.lr.x - extent.ul.x + 1, 
            extent.lr.y - extent.ul.y + 1);
    else
        printf("Error.\n");
}

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

_PfGenerateFontName(), PgDrawMultiTextArea(), PgDrawTextArea(), PgExtentText(), PhPoint_t, PhRect_t_

“Text” in the Raw Drawing and Animation chapter of the Photon
_Programmer’s Guide_
**Synopsis:**

```c
PhRect_t *PgExtentText( PhRect_t *extent,
                       PhPoint_t const *pos,
                       char const *font,
                       char const *str,
                       unsigned n );
```

**Library:**

ph

**Description:**

This function determines the extent that would be occupied if `str` were rendered at `pos`, using `font` (which you should create by calling `PfGenerateFontName()`). The extent information is stored in the `PhRect_t` structure pointed to by `extent`. If you pass `extent` as NULL, the function returns NULL.

The function calculates the extent of the first `n` characters of the string. To have the function calculate the extent of the entire string, set `n` to zero. If you pass `pos` as NULL, it’s assumed to be (0,0).

**Returns:**

The same as pointer `extent`, or NULL if an error occurred.

**Examples:**

The following fragment determines the extent of a string drawn in 18-point, bold, italic Helvetica:

```c
PhRect_t extent;
char Helvetica18bi[MAX_FONT_TAG];

if(PfGenerateFontName("Helvetica",
                      PF_STYLE_BOLD | PF_STYLE_ITALIC, 18,
                      Helvetica18bi) == NULL) {
    perror("Unable to find font");
} else {
    if( PgExtentText( &extent, NULL, Helvetica18bi,
                      "Hello World!", 0 ) ) {
        printf( "Ascent: %d Descent: %d Width: %d\n",
```
**PgExtentText()**

```c
extent.ul.y, extent.lr.y,
extent.lr.x - extent.ul.x + 1 );
} else {
    printf( "Error.\n" );
}
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfGenerateFontName(), PgDrawMultiTextArea(), PgDrawText(), PgExtentMultiText(), PgSetFont(), PhPoint_t, PhRect_t*

“Text” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgFlush()**, **PgFFlush()**, **PgFlushCx()**, **PgFFlushCx()**

Explicitly flush the current draw buffer

**Synopsis:**

```
int PgFlush( void );
int PgFFlush( unsigned int flags );
int PgFlushCx( void *dc );
int PgFFlushCx( void *dc, unsigned int flags );
```

**Library:**

ph

**Description:**

These functions flush the current draw buffer and then clear it. Most applications call `PgFlush*()`; you need to call `PgFFlush*()` only if you need to specify the type of draw event.

Normally, you set `flags` to `Ph_NORMAL_DRAW`; this is the same as calling `PgFlush*()`. But if your application has received an expose event with a subtype of `Ph_CAPTURE_EXPOSE`, you should encapsulate your drawing events by calling

```
PgFFlush( Ph_START_DRAW );
```

before drawing anything, and

```
PgFFlush( Ph_DONE_DRAW );
```

after the entire exposed region has been redrawn. If you’re using widgets, `PtEventHandler()` makes both these calls automatically.

Any function that builds a command in the draw buffer calls `PgFlush*()` automatically when the draw buffer becomes full.

`PgFlush()` and `PgFFlush()` work on the current draw context, while you can specify the draw context `dc` for `PgFlushCx()` and `PgFFlushCx()`.
**PgFlush(), PgFFlush(), PgFlushCx(), PgFFlushCx()**

**Returns:**

0  Successful completion.
-1  An error occurred.

**Examples:**

```c
/*
 * Place a group of 3 lines in the draw buffer
 */
PgDrawILine( 100, 100, 200, 300 );
PgDrawILine( 200, 300, 700, 700 );
PgDrawILine( 700, 700, 0, 0 );

/*
 * Emit a draw event that contains those 3 lines
 */
PgFlush();
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgClearDrawBuffer*(), *PgSetDrawBufferSize*(), *PhEvent_t*
**PgGetColorModel()**, **PgGetColorModelCx()**

Get the current color model

**Synopsis:**

```c
PgColorModel_t * PgGetColorModel( void );

PgColorModel_t * PgGetColorModelCx (PhGC_t *gc);
```

**Arguments:**

- `gc`  
  *PgGetColorModelCx() only. A pointer to a graphics context, as returned by PgCreateGC() or PgGetGC().

**Library:**

ph

**Description:**

These functions get the current color model. For descriptions of the currently supported color models, see PgColor_t.

*PgGetColorModel()* works on the current graphics context, while you can specify the graphics context for *PgGetColorModelCx().*

**Returns:**

The current color model.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PgColor_t, PgSetColorModel*()
**PgGetGC(), PgGetGCCx()**

*Get current graphics context*

**Synopsis:**

```c
PhGC_t *PgGetGC( void );

PhGC_t *PgGetGCCx(void *dc);
```

**Arguments:**

- `dc` for `PgGetGCCx()` only. A pointer to a draw context.

**Library:**

`ph`

**Description:**

These functions get the current graphics context. `PgGetGC()` works on the current draw context, while you can specify the draw context for `PgGetGCCx()`.

**Returns:**

A pointer to the current graphics context.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PgCreateGC(), PgDestroyGC(), PgSetGC*()
Synopsis:

```c
int PgGetGraphicsHWCaps (PgHWCaps_t *caps);
```

Arguments:

- `caps` A pointer to a `PgHWCaps_t` structure that the function can fill with the hardware capabilities; see below.

Library:

- ph

Description:

This function determines the hardware capabilities and fills out the structure pointed to by `caps`.

- You must call `PhAttach()` before calling this function.
- If you have more than one card in your system, you must target this function at a specific card by calling `PdSetTargetDevice()`.
- `PgGetGraphicsHWCaps()` blocks until the operation is complete.

---

`PgHWCaps_t`

The `PgHWCaps_t` structure includes at least the following members:

- `unsigned short current_video_mode`
- `unsigned char current_rate`
- `unsigned char current_mode_flags`
- `unsigned long rasteriser_version`
- `unsigned long driver_version`
- `unsigned long total_video_ram`
- `unsigned long total_crtc_ram`
- `unsigned long total_non_crtc_ram`
- `unsigned long currently_available_video_ram`
- `unsigned long currently_available_crtc_ram`
- `unsigned long currently_available_non_crtc_ram`
- `unsigned long card_capabilities`
- `unsigned short min_pitch`
- `unsigned short max_pitch`
PgGetGraphicsHWCaps()

unsigned short mult_pitch;
unsigned short reserved;
unsigned char chip_name[40];

The members are as follows:

current_video_mode

The number of the mode the video card is currently in.

current_rrate

Current refresh rate, in Hz. A value of 0 means the refresh rate is the default or not supported.

current_mode_flags

Current flags for the mode. The only value currently is Pg_CM_DOUBLE_BUFFERED, meaning that double buffering is being used.

rasteriser_version

The version of the device-independent portion of the driver.

driver_version

The version of the device-dependent portion of the driver.

total_video_ram

The amount of video ram on this card.

total_crtc_ram

The amount of RAM allocated to the CRTC-safe area.

total_non_crtc_ram

The amount of RAM allocated to the non-CRTC-safe area.

currently_available_video_ram

The total currently available video memory.

currently_available_crtc_ram

The currently available CRTC-safe video memory.
PgGetGraphicsHWCaps()

currently_available_non_crtc_ram
    The currently available non-CRTC-safe video memory.

card_capabilities
    The capabilities of this video card — see below.

min_pitch
    The minimum number of bytes per scan line for any offscreen context. The driver already makes sure that any request for offscreen memory is at least this big; this member is here just to help with debugging and to help applications make efficient use of the video memory (for example, some chips have a minimum pitch of 1024)

max_pitch
    The largest number of bytes per scan line that the driver accepts.

mult_pitch
    The number of bytes per scan line must be a multiple of this value (the driver ensures this, too).

chip_name[40]
    The name of this chipset.

The pitch values can change for each mode; you should use them only after changing to the desired mode.

The general hardware feature set (card capabilities) are:

Pg_2D_ACCELERATOR
    This video card has a 2D accelerator.

Pg_VIDEO_OVERLAY
    This video card has video-overlay support.

Pg_OFFSCREEN
    This video card can use offscreen video memory.
PgGetGraphicsHWCaps()

Pg_LINEAL_FRAME_BUFFER_CAPABLE

This video card can use a linear frame buffer.

These fields only say that it is possible to get these features on this video card. You still need to check the mode capabilities to see if they’re available in any given mode.

Returns:

0 Success.

-1 An error occurred.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdSetTargetDevice(), PgGetVideoMode(), PgGetVideoModeInfo(), PgGetVideoModeList(), PgSetVideoMode()

“Video modes” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgGetLayerCaps()**

*Query the capabilities of a layer*

**Synopsis:**

```c
int PgGetLayerCaps( int layer,
                    int format_index,
                    PgLayerCaps_t *const caps );
```

**Arguments:**

- `layer`: The layer index, which must be 0 or greater.
- `format_index`: The format index, which must be 0 or greater.
- `caps`: A pointer to a `PgLayerCaps_t` structure that the function fills with the layer capabilities. Any existing content of the structure is discarded. This argument must not be NULL.

**Library:**

`ph`

**Description:**

`PgGetLayerCaps()` queries the capabilities of the given layer, in the given format. You can use the `format_index` to iterate through all of the available layer formats. An error is returned when `format_index` exceeds the number of supported formats.

You must target this function at a device by calling `PdSetTargetDevice()`.

**Returns:**

- `0`: Success.
- `-1`: An error occurred (`errno` is set).
**PgGetLayerCaps()**

**Errors:**

- **EFAULT** The `caps` argument is NULL.
- **EINVAL** The layer index is invalid, or the `format_index` if greater than the number of formats.
- **ENXIO** The layer doesn’t exist.
- **EOPNOTSUPP** The driver doesn’t support layers.

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PdSetTargetDevice()`, `PgLayerCaps_t`
- “Layers” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgGetOverlayChromaColor()**

*Return the color used for video overlay chroma-key operations*

**Synopsis:**

```c
PgColor_t PgGetOverlayChromaColor( void );
```

**Library:**

`ph`

**Description:**

This function returns the standard color that’s used for Video Overlay chroma-keying operations.

**Returns:**

A composite color.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgConfigScalerChannel()`, `PgColor_t`, `PgCreateVideoChannel()`, `PgDestroyVideoChannel()`, `PgGetScalerCapabilities()`, `PgNextVideoFrame()`, `PgScalerCaps_t`, `PgScalerProps_t`, `PgVideoChannel_t`

“Video overlay” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgGetPalette()**

*Query for current color palette*

**Synopsis:**

```c
int PgGetPalette( PgColor_t *palette );
```

**Library:**

`ph`

**Description:**

This function queries the graphics driver for the current palette. A palette is an array of \_Pg\_MAX\_PALETTE RGB (`PgColor_t`) colors. The `palette` parameter must be a pointer to a suitably-sized array, which will be filled with the color values representing the current palette.

This function is useful for graphical image utilities such as `pv`, which can perform improved dithering with knowledge of the graphics palette.

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgGetPalette()` blocks until the operation is complete.

**Returns:**

- 0  Successful completion.
- -1  An error occurred.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
</table>

*continued...*
### PgGetPalette()

#### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

#### See also:

- `PdSetTargetDevice()`, `PgColor_t`
- “Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgGetRegion(), PgGetRegionCx()**

*Get the ID of the region that emits draw events*

**Synopsis:**

```c
PhRid_t PgGetRegion( void );

PhRid_t PgGetRegionCx( void *dc );
```

**Arguments:**

*dc*  
*PgGetRegionCx() only.* A void pointer to any type of draw context. Examples of draw contexts are:

- A *PhDrawContext_t* returned by *PhDCCreate()*
- A *PmMemoryContext_t* returned by *PmMemCreateMC()*
- A *PpPrintContext_t* returned by *PpCreatePC()*
- A *PdOffscreenContext_t* returned by *PdCreateOffscreenContext()*

**Library:**

ph

**Description:**

These functions get the ID of the region that currently emits draw events.

*PgGetRegion()* works on the current draw context, while you can specify the draw context for *PgGetRegionCx().*

**Returns:**

The ID of the region.

**Classification:**

Photon
PgGetRegion(), PgGetRegionCx()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgSetRegion*()
**PgGetScalerCapabilities()**

Get the capabilities of a video overlay scaler

**Synopsis:**

```c
int PgGetScalerCapabilities(
    PgVideoChannel_t *channel,
    int format_index,
    PgScalerCaps_t *vcaps );
```

**Library:**

`ph`

**Description:**

This function gets the capabilities of the Video Overlay scaler specified by `channel` for the video data format specified by `format_index`. The capabilities are stored in the `PgScalerCaps_t` structure pointed to by `vcaps`.

To find out about all available video data formats, call with increasing values for `format_index`, starting at zero, until -1 is returned.

**Returns:**

- 0 Success.
- -1 An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PgGetScalerCapabilities()

See also:

PgConfigScalerChannel(), PgCreateVideoChannel(),
PgDestroyVideoChannel(), PgGetOverlayChromaColor(),
PgNextVideoFrame(), PgScalerCaps_t, PgScalerProps_t,
PgVideoChannel_t

“Video overlay” in the Raw Drawing and Animation chapter of the
Photon Programmer’s Guide
**PgGetVideoMode()**

*Get the current video mode*

**Synopsis:**

```c
int PgGetVideoMode( PgDisplaySettings_t *settings );
```

**Library:**

`ph`

**Description:**

This function gets the current video mode, storing information about it in the `PgDisplaySettings_t` structure pointed to by `settings`, which includes at least the following:

- **unsigned mode**
  - The number of the current mode for the video card.
- **int refresh**
  - The refresh rate, in Hz. A value of 0 indicates that the default refresh rate for this mode (usually 60Hz) is being used.
- **unsigned flags**
  - There are currently no flags defined.

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgGetVideoMode()` blocks until the operation is complete.

**Returns:**

- **0** Success.
- **-1** An error occurred.
**PgGetVideoMode()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdSetTargetDevice(), PgGetGraphicsHWCaps(), PgGetVideoModeInfo(), PgGetVideoModeList(), PgSetVideoMode()*

“Video modes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgGetVideoModeInfo()**

*Get information about a video mode*

**Synopsis:**

```c
int PgGetVideoModeInfo(
    unsigned short  mode_number,
    PgVideoModeInfo_t *mode_info);
```

**Library:**

`ph`

**Description:**

This function requests detailed information on the video mode specified by the `mode_number` (taken from a `PgVideoModes_t` structure — see `PgGetVideoModeList()`).

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgGetVideoModeInfo()` blocks until the operation is complete.

The information is copied into the `PgVideoModeInfo_t` structure pointed to by `mode_info`. This structure has at least the following members:

- `unsigned short width`
  The width of the mode in pixels.

- `unsigned short height`
  The height of the mode in pixels.

- `unsigned short bits_per_pixel`
  The number of bits for each pixel.

- `unsigned short bytes_per_scanline`
  The number of bytes for each scan line.

- `unsigned long type`
  The image type. This corresponds to the Pg_IMAGE_XXX types — see `PhImage_t`.  

---

630  Chapter 8  Pg—Graphics
unsigned long mode_capabilities1
    General capabilities while in this mode - see below.

unsigned long mode_capabilities2
    General 2D hardware capabilities while in this mode - see below.

unsigned long mode_capabilities3
    Reserved.

unsigned long mode_capabilities4
    Reserved.

unsigned long mode_capabilities5
    Reserved.

unsigned long mode_capabilities6
    Reserved.

unsigned short refresh_rates[20]
    Refresh rates supported in this mode. A zero means you’ve reached the end of the list.

The general graphics card capabilities (mode_capabilities1) are:

PgVM_MODE_CAP1_OFFSCREEN
    This driver supports offscreen mode.

PgVM_MODE_CAP1_2D_ACCEL
    The 2D accelerator works in this mode.

PgVM_MODE_CAP1_VIDEO_OVERLAY
    Video overlay works in this mode.

PgVM_MODE_CAP1_LINEAR
    This mode’s memory is linear (not banked switched).

PgVM_MODE_CAP1_DOUBLE_BUFFER
    This mode can be double buffered.
PgGetVideoModeInfo()

PgVM_MODE_CAP1_TRIPLE_BUFFER
This mode can be triple buffered.

PgVM_MODE_CAP1_REFRESH_RATE
This mode supports multiple refresh rates.

The 2D acceleration capabilities (mode_capabilities2) are:

PgVM_MODE_CAP2_BITBLT
Hardware bit-blits.

PgVM_MODE_CAP2_RECTANGLE
Hardware rectangles.

PgVM_MODE_CAP2_LINES
Hardware lines.

PgVM_MODE_CAP2_POLYGONS
Hardware polygons.

PgVM_MODE_CAP2_FULL_ROPS
All 256 raster operations in hardware.

PgVM_MODE_CAP2_PATTERN
Hardware patterns (transparent or not).

PgVM_MODE_CAP2_CHROMA
Hardware chroma key capabilities.

PgVM_MODE_CAP2_ALPHA_BLEND
Hardware alpha blending capabilities.

PgVM_MODE_CAP2_PLANE_MASK
Hardware plane masking.

PgVM_MODE_CAP2_SCALED_BLIT
Hardware accelerated scaled blit
**PgGetVideoModeInfo()**

PgVM_MODE_CAP2_DWORD
   Accelerated DWORD-aligned data.
PgVM_MODE_CAP2_WORD
   Accelerated WORD-aligned data.
PgVM_MODE_CAP2_BYTE
   Accelerated BYTE-aligned data.
PgVM_MODE_CAP2_SYSTEM_RAM
   Transfers to and from system RAM using the hardware.

**Returns:**

0   Success.
-1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PdSetTargetDevice(), PgGetGraphicsHWCaps(), PgGetVideoMode(), PgGetVideoModeList(), PgSetVideoMode()*

“Video modes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgGetVideoModeList()**

*Query a graphics driver for a list of its supported video modes*

**Synopsis:**

```c
int PgGetVideoModeList( PgVideoModes_t *mode_list );
```

**Library:**

`ph`

**Description:**

This function queries the graphics driver for a list of video modes that it supports. The list is an array of `short` entries that represent mode numbers that the drivers use to identify a video mode.

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgGetVideoModeList()` blocks until the operation is complete.

The `mode_info` argument is a pointer to a `PgVideoModes_t` structure in which to store the mode information. This structure has at least the following members:

```
unsigned short num_modes
    The number of modes in the list.

unsigned short mode[127]
    The mode list.
```

**Returns:**

`0` Success.

`-1` An error occurred.

**Classification:**

Photon
**PgGetVideoModeList()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdSetTargetDevice(), PgGetGraphicsHWCaps(), PgGetVideoMode(), PgGetVideoModeInfo(), PgSetVideoMode()*

“Video modes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgGray()**

*Generate the RGB value for a shade of gray*

**Synopsis:**

```c
PgColor_t PgGray( int level );
```

**Arguments:**

- `level` The gray level, which ranges from 0 (black) to 255 (white). Intermediate values produce various shades of gray.

**Library:**

`ph`

**Description:**

This macro converts a gray value into a `PgColor_t` structure. For a complete description of composite color values, see `PgSetFillColor()`.

**Returns:**

A composite color value.

**Examples:**

<table>
<thead>
<tr>
<th>Color</th>
<th>Gray value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td><code>PgGray( 0 );</code></td>
</tr>
<tr>
<td>White</td>
<td><code>PgGray( 255 );</code></td>
</tr>
<tr>
<td>Medium Gray</td>
<td><code>PgGray( 160 );</code></td>
</tr>
</tbody>
</table>

**Classification:**

Photon
**PgGray()**

**Safety**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgBlueValue(), PgColor_t, PgCMY(), PgGreenValue(), PgHSV(), PgRedValue(), PgRGB(), PgSetFillColor(), PgSetFillDither()*

“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgGrayValue()**
*Extract color brightness*

**Synopsis:**

```c
int PgGrayValue( PgColor_t color );
```

**Arguments:**

- `color` A composite color value, of type `PgColor_t`.

**Library:**

`ph`

**Description:**

This macro converts a composite color into its corresponding level of gray. The calculation is based on 30% red, 59% green, and 11% blue, resulting in a value between 0 and 255.

**Returns:**

The gray component of the color.

**Examples:**

```c
// Convert pal[i] into monochrome
pal[i] = PgGray( PgGrayValue( pal[i] ) );
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

---

638  Chapter 8  •  Pg—Graphics
See also:

PgBlueValue(), PgCMY(), PgColor_t, PgGreenValue(), PgHSV(), PgRedValue(), PgRGB(), PgSetColor(), PgSetFillDither()

“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PgGreenValue()

Extract the green component from a color value

Synopsis:

\[
\text{int PgGreenValue( PgColor}_t \text{ color );}
\]

Arguments:

\(color\) A composite color value, of type \(\text{PgColor}_t\).

Library:

\(\text{ph}\)

Description:

This macro extracts the green color component from a composite color value. The result is between 0 and 255.

Classification:

Photon

Safety

\begin{center}
\begin{tabular}{l|c}
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}
\end{center}

Caveats:

\(\text{PgGreenValue()}\) is a macro.

See also:

\(\text{PgAlphaValue(), PgRGB(), PgBlueValue(), PgCMY(), PgColor}_t,\)
\(\text{PgHSV(), PgRedValue(), PgRGB(), PgSetFillColor(),}\)
\(\text{PgSetFillDither()}\)
PgGreenValue()

“Color” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**PgHSV()**

*Convert hue, saturation, and value to composite color format*

**Synopsis:**

```
PgColor_t PgHSV( unsigned H, int S, int V );
```

**Library:**

`ph`

**Description:**

This function converts hue, saturation, and value into a `PgColor_t` structure.

A color circle is divided into 65536 gradations (called *binary gradations* or *bi-grads*). Hue is in bi-grads, starting with red at 0 (0 degrees), green at 0x5555 (120 degrees), and blue at 0xAAAA (240 degrees):

<table>
<thead>
<tr>
<th>Hue</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>Red</td>
</tr>
<tr>
<td>0x2AAA</td>
<td>Yellow</td>
</tr>
<tr>
<td>0x5555</td>
<td>Green</td>
</tr>
<tr>
<td>0x8000</td>
<td>Cyan</td>
</tr>
<tr>
<td>0xAAAA</td>
<td>Blue</td>
</tr>
<tr>
<td>0xD555</td>
<td>Magenta</td>
</tr>
<tr>
<td>0xFFFF</td>
<td>Almost red</td>
</tr>
</tbody>
</table>

The values for saturation and value range from 0 to 255:

<table>
<thead>
<tr>
<th>Saturation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Gray</td>
</tr>
</tbody>
</table>

*continued...*
\textbf{PgHSV()}

<table>
<thead>
<tr>
<th>Saturation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Muted</td>
</tr>
<tr>
<td>255</td>
<td>Pure color</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>128</td>
<td>Dark</td>
</tr>
<tr>
<td>255</td>
<td>Full brightness</td>
</tr>
</tbody>
</table>

\textbf{Returns:}

A composite color value.

\textbf{Examples:}

<table>
<thead>
<tr>
<th>Color</th>
<th>HSV value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>PGHSV( 0, 0, 0 );</td>
</tr>
<tr>
<td>White</td>
<td>PGHSV( 0, 0, 255 );</td>
</tr>
<tr>
<td>Red</td>
<td>PGHSV( 0, 255, 255 );</td>
</tr>
<tr>
<td>Green</td>
<td>PGHSV( 0x5555, 255, 255 );</td>
</tr>
<tr>
<td>Blue</td>
<td>PGHSV( 0xAAAA, 255, 255 );</td>
</tr>
<tr>
<td>Orange</td>
<td>PGHSV( 0x1400, 255, 255 );</td>
</tr>
<tr>
<td>Slate Blue</td>
<td>PGHSV( 0xA000, 121, 134 );</td>
</tr>
</tbody>
</table>
PgHSV()

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgBlueValue(), PgCMY(), PgColor_t, PgGreenValue(), PgHSV2RGB(), PgRedValue(), PgRGB(), PgRGB2HSV(), PgSetFillColor(), PgSetFillDither()

“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**
```
PgColor_t PgHSV2RGB( PgColorHSV_t hsv_color );
```

**Library:**
```
ph
```

**Description:**
This function converts hue-saturation-value colors to composite color values.

If you write a color selection function that allows the user to change RGB and HSV values, you should maintain the RGB and HSV values separately. If the user changes the HSV value, you would use `PgHSV2RGB()` to calculate the new RGB value. For example:

```
RGBvalue = PgHSV2RGB( HSVvalue );
```

**Returns:**
A composite color value.

**Classification:**
```
Photon
```

**Safety**
<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PgHSV2RGB()

See also:

PgColor_t, PgColorHSV_t, PgHSV(), PgRGB2HSV(), PgSetFillColor()

“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
typedef struct {
    unsigned int format;
    long owner;
    unsigned int caps;
    unsigned int alpha_caps;
    unsigned int alpha_combine_caps;
    unsigned int chroma_caps;
    int max_data_width;
    int max_data_height;
    int max_src_width;
    int max_src_height;
    int max_dst_width;
    int max_dst_height;
    int min_dst_width;
    int min_dst_height;
    int max_x_scale;
    int max_y_scale;
    int min_x_scale;
    int min_y_scale;
    unsigned int reserved[4];
} PgLayerCaps_t;
```

**Description:**

The `PgLayerCaps_t` structure describes the capabilities of a layer. To fill in this structure for a layer, call `PgGetLayerCaps()`. To set the capabilities, call `PgSetLayerArg()`.

The members of the `PgLayerCaps_t` structure include:

- **format**: The layer format that corresponds to the format index that you pass to `PgGetLayerCaps()`; one of:
  - `Pg_LAYER_FORMAT_PAL8`
### PgLayerCaps_t

- `Pg_LAYER_FORMAT_ARGB1555`
- `Pg_LAYER_FORMAT_RGB565`
- `Pg_LAYER_FORMAT_RGB888`
- `Pg_LAYER_FORMAT_ARGB8888`
- `Pg_LAYER_FORMAT_YUV2`
- `Pg_LAYER_FORMAT_UYVY`
- `Pg_LAYER_FORMAT_YVYU`
- `Pg_LAYER_FORMAT_V422`
- `Pg_LAYER_FORMAT_YVU9`
- `Pg_LAYER_FORMAT_YV12`
- `Pg_LAYER_FORMAT_YUV420`

**owner**
The Photon client ID of the owner, if the surface is locked.

**caps**
General capabilities; a combination of the following:

- `Pg_LAYER_CAP_DISABLE` — the layer can be hidden.
- `Pg_LAYER_CAP_PAN_SOURCE` — the offset of the displayed image from the source image can be changed.
- `Pg_LAYER_CAP_PAN_DEST` — the layer’s window on the screen can be moved.
- `Pg_LAYER_CAP_SIZE_DEST` — the layer’s window on the screen can be resized.
- `Pg_LAYER_CAP_EDGE_CLAMP` — the last pixel of the source data can be replicated downwards/rightwards if the source viewport is scrolled past the bottom/right of the source data.
- `Pg_LAYER_CAP_EDGE_WRAP` — the source data can be wrapped from the top/left if the source viewport is scrolled past the bottom/right of the source data.
PgLayerCaps_t

- `Pg_LAYER_CAP_SET_BRIGHTNESS` — the layer’s brightness can be changed.
- `Pg_LAYER_CAP_SET_CONTRAST` — the layer’s contrast can be changed.
- `Pg_LAYER_CAP_SET_SATURATION` — the layer’s saturation can be changed.
- `Pg_LAYER_CAP_ALPHA_WITH_CHROMA` — this layer supports simultaneous alpha blending and chroma-keying.
- `Pg_LAYER_CAP_SCALE_REPLICATE` — scaling by pixel replication is supported.
- `Pg_LAYER_CAP_FILTER` — filtering is supported.
- `Pg_LAYER_CAP_MAIN_DISPLAY` — this layer is the screen’s primary display. This layer’s destination viewport is fixed to the physical screen size. This layer’s format is fixed to the current video mode.

*alpha_caps*

Alpha capabilities. Only global and per-pixel alpha blending are currently supported. Alpha maps aren’t supported for blending between layers.

- `Pg_LAYER_ALPHA_M1_SRC_PIXEL_ALPHA` — The primary alpha multiplier “M1” comes from the source pixels alpha component. Not valid if the source pixel format does not include an alpha component.
- `Pg_LAYER_ALPHA_M1_DST_PIXEL_ALPHA` — The primary alpha multiplier “M1” comes from the destination pixels alpha component. Not valid if the destination pixel format does not include an alpha component.
- `Pg_LAYER_ALPHA_M1_GLOBAL` — The primary alpha multiplier “M1” comes from global multiplier 1.
- \texttt{Pg\_LAYER\_ALPHA\_M2\_SRC\_PIXEL\_ALPHA} — The secondary alpha multiplier “M2” comes from the source pixels alpha component. Not valid if the source pixel format does not include an alpha component.

- \texttt{Pg\_LAYER\_ALPHA\_M2\_DST\_PIXEL\_ALPHA} — The secondary alpha multiplier “M2” comes from the destination pixels alpha component. Not valid if the destination pixel format does not include an alpha component.

- \texttt{Pg\_LAYER\_ALPHA\_M2\_GLOBAL} — The secondary alpha multiplier “M2” comes from global multiplier 2.

- \texttt{Pg\_LAYER\_BLEND\_SRC\_M1\_ALPHA} — Specifies how Ms, the value that will be multiplied with the source pixel data to produce the resultant image, is derived. Ms = M1

- \texttt{Pg\_LAYER\_BLEND\_SRC\_ONE\_MINUS\_M1\_ALPHA} — Specifies how Ms, the value that will be multiplied with the source pixel data to produce the resultant image, is derived. Ms = 1 - M1

- \texttt{Pg\_LAYER\_BLEND\_SRC\_M2\_ALPHA} — Specifies how Ms, the value that will be multiplied with the source pixel data to produce the resultant image, is derived. Ms = M2

- \texttt{Pg\_LAYER\_BLEND\_SRC\_ONE\_MINUS\_M2\_ALPHA} — Specifies how Ms, the value that will be multiplied with the source pixel data to produce the resultant image, is derived. Ms = 1 - M2

- \texttt{Pg\_LAYER\_BLEND\_DEST\_M1\_ALPHA} — Specifies how Md, the value that will be multiplied with the destination pixel data to produce the resultant image, is derived. Md = M1
**alpha_combine_caps**

Alpha combine capabilities.

- **Pg_LAYER_ALPHA_COMBINE_CAP_SPP_WITH_DG** — source per-pixel alpha blending with destination global alpha multiplier supported.

- **Pg_LAYER_ALPHA_COMBINE_CAP_SG_WITH_DPP** — destination per-pixel alpha blending with source global alpha multiplier supported.

- **Pg_LAYER_ALPHA_COMBINE_CAP_SPP_WITH_DPP** — source per-pixel with alpha blending with destination per-pixel alpha blending supported.

**chroma_caps**

Chroma key capabilities:

- **Pg_LAYER_CHROMAKEY_CAP_SRC_SINGLE** — Chroma-keying based on an exact match
between the source pixel value, and a single key color, is supported.

- **Pg_LAYER_CHROMAKEY_CAP_DST_SINGLE** — Chroma-keying based on an exact match between the destination pixel value, and a single key color, is supported.

- **Pg_LAYER_CHROMAKEY_CAP_SHOWTHROUGH** — Indicates that the layer can be configured so that when a chroma-key comparison is made, and the colors match, the displayed pixel will come from the layer(s) behind. If the colors do not match, the displayed pixel will come from this layer.

- **Pg_LAYER_CHROMAKEY_CAP_BLOCK** — Indicates that the layer can be configured so that when a chroma-key comparison is made, and the colors match, the displayed pixel will come from this layer. If the colors do not match, the pixel will come from the layer(s) behind.

- **max_data_width, max_data_height**
  The maximum size of a layer surface, in pixels.

- **max_src_width, max_src_height**
  The maximum size of the source viewport, in pixels.

- **max_dst_width, max_dst_height**
  The maximum size of the destination viewport, in pixels.

- **min_dst_width, min_dst_height**
  The minimum size of the destination viewport, in pixels.

- **max_x_scale**
  The maximum scaling factor for scaling up in the horizontal direction:
• 1 — you can’t scale up.

• > 1 — the width of the destination viewport can be up to $max_x\_scale$ times the width of the source viewport.

Values less than 1 are invalid.

$max_y\_scale$ The maximum scaling factor for scaling up in the vertical direction:

• 1 — you can’t scale up.

• > 1 — the height of the destination viewport can be up to $max_y\_scale$ times the height of the source viewport.

Values less than 1 are invalid.

$min_x\_scale$ The maximum scaling factor for scaling down in the horizontal direction:

• 1 — you can’t scale down.

• > 1 — the width of the source viewport can be up to $min_x\_scale$ times the width of the destination viewport.

Values less than 1 are invalid.

$min_y\_scale$ The maximum scaling factor for scaling down in the vertical direction:

• 1 — you can’t scale down.

• > 1 — the height of the source viewport can be up to $min_y\_scale$ times the height of the destination viewport.

Values less than 1 are invalid.
PgLayerCaps_t

Classification:
Photon

See also:

PgGetLayerCaps(), PgSetLayerArg(), PhImage_t
“Layers” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**Synopsis:**

```c
int PgLockLayer(int layer);
```

**Arguments:**

*layer*  
The layer index, which must be 0 or greater.

**Library:**

ph

**Description:**

*PgLockLayer()* acquires exclusive use of a layer by an application. To unlock a layer, call *PgUnlockLayer()*.

Other applications may not use *PgSetLayerSurface()* or *PgSetLayerArg()* on a locked surface.

You must target this function at a device by calling *PdSetTargetDevice()*.

Your application should unlock its layers before it exits. You can lock a layer multiple times, but need to unlock it only once.

**Returns:**

0  
Success.

-1  
An error occurred (errno is set).

**Errors:**

*EBUSY*  
The specified layer is locked by another application.

*EINVAL*  
Any other error.

*ENXIO*  
The layer doesn’t exist.

*EOPNOTSUPP*  
The operation isn’t supported.
PgLockLayer()

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdSetTargetDevice(), PgSetLayerArg(), PgSetLayerSurface(), PgUnlockLayer()

“Layers” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
typedef struct _Pg_map {
    PhDim_t     dim;
    short unsigned bpl;
    short unsigned bpp;
    char *map;
} PgMap_t;
```

**Description:**

The `PgMap_t` structure defines an alpha blend map. Its members include:

- `dim`: A `PhDim_t` structure that defines the size of area covered by the map, in pixels.
- `bpl`: The number of bytes per line.
- `bpp`: The number of bits per pixel.
- `map`: A pointer to the map itself.

**Classification:**

Photon

**See also:**

- `PgAlphaOff()`, `PgAlphaOn()`, `PgColor_t`, `PgSetAlpha()`, `PgSetAlphaBlend()`, `PhDim_t`
- “Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgNextVideoFrame()**

*Get the index of the next video buffer to fill*

**Synopsis:**

```c
int PgNextVideoFrame( PgVideoChannel_t *channel );
```

**Library:**

`ph`

**Description:**

`PgNextVideoFrame()` returns the index of the video buffers into which your application should copy the video frame data. Call this function before transferring a frame of video data to the video overlay scaler.

**Returns:**

The index of the video buffer, or -1 if an error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgConfigScalerChannel()`, `PgCreateVideoChannel()`, `PgDestroyVideoChannel()`, `PgGetOverlayChromaColor()`, `PgGetScalerCapabilities()`, `PgScalerCaps_t`, `PgScalerProps_t`, `PgVideoChannel_t`

“Video overlay” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgReadScreen()**  
Read an image from the screen

**Synopsis:**

```c
PhImage_t *PgReadScreen( PhRect_t *rect,
                        void *buffer);
```

**Library:**

`ph`

**Description:**

This function retrieves an image from the screen by querying the local graphics driver(s), using the highest bit depth possible.

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgReadScreen()` blocks until the operation is complete.

The `PhRect_t` structure pointed to by `rect` specifies the area to capture and is in absolute Photon coordinates.

The `buffer` argument points to a buffer in which to store the image structure and its associated data, palette, and so on. This must be shared memory, since the graphics driver needs to be able to access it.

Use `PgReadScreenSize()` to determine an adequate size for the buffer. If `buffer` is NULL, a shared memory object is created for you.

Note the following restrictions:

- There must be a graphics driver present that completely encompasses the rectangle being captured.

- The graphics driver must be running on the same node as the calling process, since shared memory is used to pass data.

- All data associated with the image (structure, palette, data) is stored in one contiguous shared memory object. Release this memory by calling `PgShmemDestroy()`; don’t call `PhReleaseImage()` for an image acquired using `PgReadScreen()`.

If you plan to use the acquired image in an environment where this restriction proves inconvenient, then you should make a copy of
**PgReadScreen()**

the image using `PiDuplicateImage()`, after which you may free the original as outlined above.

**Returns:**

A pointer to the `PhImage_t` structure that defines the image, or NULL if the operation failed (`errno` is set).

**Errors:**

- **ENOMEM** Insufficient memory to perform the operation.
- **ENXIO** There was no graphics driver present to capture the specified rectangle.

See the description of `shm_open()` in the QNX Neutrino Library Reference for further errors that may occur.

**Examples:**

```c
PhImage_t *image;
PhRect_t rect = { { 0,0 }, {100,100 } };  

if(image = PgReadScreen(&rect,buffer))
{
    /* Manipulate the image */
    ...

    /* Free the memory */
    PgShmemDestroy(image);
}
```

This example uses `PgReadScreenState()` to determine the amount of shared memory to allocate for the buffer:

```c
PhImage_t *image;
PhRect_t rect = { { 0,0 }, { 31,31 } };  

if((image = PgShmemCreate(PgReadScreenState(&rect),NULL)) &&
   PgReadScreen(&rect,image))
{
    /* Manipulate the image */
    ...

    /* Free the memory */
    PgShmemDestroy(image);
}
```
PgReadScreen()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdSetTargetDevice(), PgReadScreenSize(), PgShmemDestroy(), PhImage_t, PhRect_t, PiDuplicateImage()

shm_open() in the QNX Neutrino Library Reference
**PgReadScreenSize()**

*Determine the memory requirements for reading an image from the screen*

**Synopsis:**

```c
unsigned long PgReadScreenSize( PhRect_t *rect );
```

**Library:**

`ph`

**Description:**

This function determines how much memory is required to store an image that would be generated by a corresponding call to `PgReadScreen()`. This function is useful only if you plan to allocate an image storage buffer yourself.

The `PhRect_t` structure pointed to by `rect` specifies the target area (in absolute Photon coordinates).

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgReadScreenSize()` blocks until the operation is complete.

**Returns:**

The number of bytes required to store the image, including palette data (if applicable), or 0 if an error occurred.

**Errors:**

See `PgReadScreen()`.

**Classification:**

Photon

---

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>continued...</td>
<td></td>
</tr>
</tbody>
</table>
PgReadScreenSize()

Safety

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgReadScreen(), PhRect_t
**PgRedValue()**  
*Extract the red component from a color value*

**Synopsis:**

```c
int PgRedValue( PgColor_t color );
```

**Arguments:**

- `color`: A composite color value, of type `PgColor_t`.

**Library:**

`ph`

**Description:**

This macro extracts the red color component from a composite color value. The result is between 0 and 255.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**Caveats:**

`PgRedValue()` is a macro.

**See also:**

- `PgAlphaValue()`, `PgARGB()`, `PgBlueValue()`, `PgCMY()`, `PgColor_t`,  
- `PgGreenValue()`, `PgHSV()`, `PgRGBO`, `PgSetFillColor()`,  
- `PgSetFillDither()`
“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgRGB()**

*Convert red, green, and blue values to composite color format*

**Synopsis:**

```c
PgColor_t PgRGB( int R, int G, int B );
```

**Library:**

`ph`

**Description:**

This macro converts red, green, and blue values into a composite color value (a `PgColor_t` type). The values for red, green, and blue range from 0 to 255. If you set all three arguments to 0, the color is black; if you set all three to 255, the color is white.

This macro doesn’t support alpha; for colors involving alpha, use `PgARGB()`.

**Examples:**

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>PgRGB( 0, 0, 0 );</td>
</tr>
<tr>
<td>White</td>
<td>PgRGB( 255, 255, 255 );</td>
</tr>
<tr>
<td>Red</td>
<td>PgRGB( 255, 0, 0 );</td>
</tr>
<tr>
<td>Green</td>
<td>PgRGB( 0, 255, 0 );</td>
</tr>
<tr>
<td>Blue</td>
<td>PgRGB( 0, 0, 255 );</td>
</tr>
<tr>
<td>Orange</td>
<td>PgRGB( 255, 165, 0 );</td>
</tr>
<tr>
<td>Slate Blue</td>
<td>PgRGB( 80, 95, 134 );</td>
</tr>
</tbody>
</table>

**Classification:**

Photon
Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

\( \text{PgARGB()}, \text{PgBlueValue()}, \text{PgColor\_t}, \text{PgCMY()}, \text{PgGreenValue()}, \text{PgHSV()}, \text{PgRedValue()}, \text{PgSetFillColor()}, \text{PgSetStrokeColor()}, \text{PgSetTextColor()} \)

“Color” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgRGB2HSV()**

*Convert RGB colors to HSV*

**Synopsis:**

```c
PgColorHSV_t PgRGB2HSV( PgColor_t rgb_color );
```

**Library:**

```c
ph
```

**Description:**

This function converts composite color values into hue-saturation-value.

If you write a color selection function that allows the user to change RGB and HSV values, you should maintain the RGB and HSV values separately. So if the user changes the RGB value, you would use `PgRGB2HSV()` to calculate the new HSV value. For example:

```c
HSVvalue = PgRGB2HSV( RGBvalue );
```

When you convert RGB values into HSV, colors close to black, white, or gray might not convert to correct hue values.

This function doesn’t copy any alpha value from the RGB color to the HSV color.

**Returns:**

A hue-saturation-value value.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

668 Chapter 8 • Pg—Graphics
See also:

PgColor_t, PgColorHSV_t, PgHSV(), PgHSV2RGB(),
PgSetFillColor()

“Color” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**PgScalerCaps_t**

*Data structure that describes video overlay scaler capabilities*

**Synopsis:**

```c
typedef struct scaler_caps {
    unsigned size;
    unsigned format;
    unsigned flags:
    short src_max_x;
    short src_max_y;
    short max_mag_factor_x;
    short max_mag_factor_y;
    short max_shrink_factor_x;
    short max_shrink_factor_y;
    unsigned reserved[10];
} PgScalerCaps_t;
```

**Description:**

The **PgScalerCaps_t** structure describes video overlay scaler capabilities. It includes at least:

- **size**: The size of this structure. The application should set this to `sizeof (PgScalerCaps_t)`.

- **format**: The video data format to which the capabilities in this structure pertain. It can be one of the following formats, which are defined in `<Pg.h>`:

  - `Pg_VIDEO_FORMAT_RGB555`
  - `Pg_VIDEO_FORMAT_RGB565`
  - `Pg_VIDEO_FORMAT_RGB8888`
  - `Pg_VIDEO_FORMAT_IYU1`
  - `Pg_VIDEO_FORMAT_IYU2`
  - `Pg_VIDEO_FORMAT_UYYVY`
  - `Pg_VIDEO_FORMAT_YUY2`
  - `Pg_VIDEO_FORMAT_YUVY`
  - `Pg_VIDEO_FORMAT_V422`
  - `Pg_VIDEO_FORMAT_CLJR`
PgScalerCaps_t

- Pg_VIDEO_FORMAT_YUV9
- Pg_VIDEO_FORMAT_YUV12
- Pg_VIDEO_PLANAR_YUV_FORMAT_CLPL
- Pg_VIDEO_PLANAR_YUV_FORMAT_VBPL

flags

The flags include:

- Pg_SCALER_CAP_DST_CHROMA_KEY — the driver can perform chroma-key testing on the desktop surface. That is, video can be made to appear only on the desktop where pixels drawn with the chroma-key color are present.
- Pg_SCALER_CAP_DOUBLE_BUFFER — video output can be double-buffered. That is, two video buffers are provided, and the application can copy each frame into the alternative buffer. Thus, one frame can be displayed, while video data is being copied into the alternative frame. This eliminates flickering and tearing artifacts.
- Pg_SCALER_CAP_BRIGHTNESS_ADJUST — the application can control the brightness of the video viewport.
- Pg_SCALER_CAP_CONTRAST_ADJUST — the application can control the contrast of the video viewport.

src_max_x, src_max_y

The maximum width and height of a video data frame before scaling.

max_mag_factor_x, max_mag_factor_y

The maximum upward scaling factor in the horizontal and vertical directions.

max_shrink_factor_x, max_shrink_factor_y

The maximum downward scaling factor in the horizontal and vertical directions. Some scalers can’t perform downward scaling, in which case these fields are set to 1.
PgScalerCaps_t

Classification:

Photon

See also:

PgConfigScalerChannel(), PgCreateVideoChannel(),
PgDestroyVideoChannel(), PgGetOverlayChromaColor(),
PgGetScalerCapabilities(), PgNextVideoFrame(),
PgScalerProps_t, PgVideoChannel_t

“Video overlay” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
typedef struct {
    unsigned size;
    unsigned flags;
    PgColor_t color_key;
    unsigned reserved0;
    PgColor_t color_key_mask;
    PhRect_t viewport;
    PhDim_t src_dim;
    unsigned format;
    int brightness;
    int contrast;
    PgVideoAlpha_t alpha[4];
    unsigned reserved[10];
} PgScalerProps_t;
```

**Description:**

The `PgScalerProps_t` structure describes video overlay scaler properties. It includes:

- **size**
  
  The size of this structure; set this to:
  
  ```c
  sizeof (PgScalerProps_t)
  ```

- **flags**
  
  The flags include:
  
  - `Pg_SCALER_PROP_CHROMA_ENABLE` — enable chroma keying. When chroma keying is enabled, video output appears only where pixels drawn in the chroma-key color are present.
  
  - `Pg_SCALER_PROP_CHROMA_SPECIFY_KEY_MASK` — use the value of the `color_key` member, instead of the default returned by `PgGetOverlayChromaColor()`, as the chroma-key color.
PgScalerProps_t

- **Pg_SCALER_PROP_DISABLE_FILTERING** — attempt to disable interpolation or filtering algorithms when scaling; scaling is instead performed using simple replication (scaling upwards) or dropping (scaling downwards) algorithms.
- **Pg_SCALER_PROP_DOUBLE_BUFFER** — turn on double-buffering of video frames.
- **Pg_SCALER_PROP_SCALER_ENABLE** — enable the video scaler output viewport.
- **Pg_SCALER_PROP_TO_BACK** — tell the driver to put the scaler region behind all other scaler regions.
- **Pg_SCALER_PROP_TO_FRONT** — tell the driver to put the scaler region in front of all other scaler regions.
- **Pg_SCALER_PROP_DRAW_TARGETABLE** — allow Photon drawing output to be directed to the video scaler buffer. Note that only RGB data format is supported for targeting in this manner.

**color_key**  The chroma-key color (of type PgColor_t) to use when Pg_SCALER_PROP_CHROMA_ENABLE and Pg_SCALER_PROP_CHROMA_SPECIFY_KEY_MASK are set in the flags member.

**color_key_mask**  Not implemented.

**viewport**  A PhRect_t structure that stores the location and dimension, in desktop coordinates, of the video scaler output viewport.

**src_dim**  A PhDim_t structure that defines the width and height of the video data frames before scaling.
**PgScalerProps_t**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>The format of the video frame data. This is analogous to the format member of the PgScalerCaps_t structure.</td>
</tr>
<tr>
<td>brightness</td>
<td>The brightness of the video output viewport. The range is 127 to -127, where 0 specifies normal brightness.</td>
</tr>
<tr>
<td>contrast</td>
<td>The contrast of the video output viewport. The range is 127 to -127, where 0 specifies normal contrast.</td>
</tr>
</tbody>
</table>

**Classification:**

Photon

**See also:**

PgColor_t, PgConfigScalerChannel(), PgCreateVideoChannel(), PgDestroyVideoChannel(), PgGetOverlayChromaColor(), PgGetScalerCapabilities(), PgNextVideoFrame(), PgScalerCaps_t, PgVideoChannel_t, PhDim_t, PhRect_t

“Video overlay” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetAlpha(), PgSetAlphaCx()**

Set the parameters for alpha blending in detail

**Synopsis:**

```c
void PgSetAlpha(
    unsigned long const alpha_op,
    PgMap_t const * const src_alpha_map,
    PgGradient_t const * const src_alpha_gradient,
    char unsigned const src_global_alpha,
    char unsigned const dst_global_alpha);

void PgSetAlphaCx(
    PhGC_t *gc,
    unsigned long const alpha_op,
    PgMap_t const * const src_alpha_map,
    PgGradient_t const * const src_alpha_gradient,
    char unsigned const src_global_alpha,
    char unsigned const dst_global_alpha);
```

**Arguments:**

- **gc** *PgSetAlphaCx()* only. A pointer to a graphics context, as returned by *PgCreateGC()* or *PgGetGC()*.
- **alpha_op** The current operation; a combination of:
  - One or more operation flags. If
    ```c
    Pg_ALPHA_OP_SRC* or
    Pg_ALPHA_OP_DST_GLOBAL isn't set, the alpha value from the source pixel (As) is used.
    And:
    - A source multiplier flag — if no flag is set, the default *Pg_BLEND_SRC_0* is used.
    - A destination multiplier flag — if no flag is set, the default *Pg_BLEND_DST_0* is used.
    Or:
    - The alpha test operation flag
      *Pg_ALPHA_OP_TEST*.
    - A test type flag — if no flag is set, the default *Pg_TEST_NEVER* is used.
**PgSetAlpha(), PgSetAlphaCx()**

`src_alpha_map`

A pointer to the alpha map to use as the source; NULL means no map. This argument is used if `alpha_op` has `Pg_ALPHA_OP_SRC_MAP` set. This map, of type `PgMap_t`, indicates the alpha blending to be applied to each individual pixel. The map is “pinned” to the origin of your draw command and is tiled if the dimensions of the map are smaller than the dimension of the drawing operation.

`src_alpha_gradient`

A pointer to the gradient alpha to use as the source. It isn’t currently supported; set it to NULL.

`src_global_alpha`

The constant source alpha value, which is used if you set `Pg_ALPHA_OP_SRC_GLOBAL`.

`dst_global_alpha`

The constant destination alpha value, which is used if you set `Pg_ALPHA_OP_DST_GLOBAL`.

**Library:**

ph

**Description:**

These functions set the parameters for an alpha-blending or alpha-test operation. `PgSetAlpha()` works on the current graphics context, while you can specify the graphics context for `PgSetAlphaCx()`.

The basic formula for alpha blending is:

\[
\begin{align*}
\text{source multiplier} & \text{ is one of } Pg\_BLEND\_SRC^* \\
\text{destination multiplier} & \text{ is one of } Pg\_BLEND\_DST^*
\end{align*}
\]

\[
\begin{align*}
Sm &= \text{source pixel} * \text{ source multiplier} \\
Dm &= \text{destination pixel} * \text{ destination multiplier} \\
\text{destination pixel} &= Sm + Dm
\end{align*}
\]
You can OR only one source and only one destination multiplier flag into alpha_op. You can’t combine source or destination flags.

**Operation flags**

The following operation flags are defined:

- **Pg_ALPHA_OP_SRC_GLOBAL**
  The source alpha value (As) is src_global_alpha.

- **Pg_ALPHA_OP_SRC_MAP**
  The source alpha is src_alpha_map. This bit can’t be used in conjunction with Pg_ALPHA_OP_SRC_GLOBAL.

- **Pg_ALPHA_OP_DST_GLOBAL**
  The destination alpha (Ad) value is dst_global_alpha.

- **Pg_ALPHA_OP_TEST**
  Perform an alpha test rather than an alpha blend. See the test flags below.

**Multiplier flags**

Flags are defined for source and destination multipliers. In the descriptions below, all channels are represented in a range from 0 to 1; that is, they have been normalized from their actual range of 0-255. As RGB colors, (1,1,1) is white, (0,0,0) is black, (0.5,0.5,0.5) is grey, (1, 0, 0) is full red, and so on.

The following descriptions assume an RGB or ARGB color model for the formulas, so As is the source pixel’s alpha channel, Rs is the source pixel’s red channel, Ad is the destination pixel’s alpha channel, Rd is the destination pixel’s red channel, and so on. For more information about color models, see PgColor_t.
Source multiplier flags

Pg_BLEND_SRC_0
Multiply the source pixel by 0: Sm = (As, Rs, Gs, Bs) * (0,0,0,0).

Pg_BLEND_SRC_1
Multiply the source pixel by 1 (no change): Sm = (As, Rs, Gs, Bs) * (1,1,1,1).

Pg_BLEND_SRC_D
Multiply the source pixel by D: Sm = (As, Rs, Gs, Bs) * (Ad, Rd, Gd, Bd).

Pg_BLEND_SRC_1mD
Multiply the source pixel by 1-D: Sm = (As, Rs, Gs, Bs) * (1-Ad, 1-Rd, 1-Gd, 1-Bd).

Pg_BLEND_SRC_As
The source pixel is multiplied by As: Sm = (As, Rs, Gs, Bs) * (As, As, As, As).

Pg_BLEND_SRC_1mAs
Multiply the source pixel by 1-As: Sm = (As, Rs, Gs, Bs) * (1-As, 1-As, 1-As, 1-As).

Pg_BLEND_SRC_Ad
Multiply the source pixel by Ad: Sm = (As, Rs, Gs, Bs) * (Ad, Ad, Ad, Ad).

Pg_BLEND_SRC_1mAd
Multiply the source pixel by 1-Ad: Sm = (As, Rs, Gs, Bs) * (1-Ad, 1-Ad, 1-Ad, 1-Ad).

Pg_BLEND_SRC_A1As
Multiply the source pixel’s alpha channel by 1, and the rest of the pixel by As: Sm = (As, Rs, Gs, Bs) * (1,As,As,As).

Pg_BLEND_SRC_1mA1As
Multiply the source pixel’s alpha channel by 0, and the rest of the pixel by 1-As: Sm = (As, Rs, Gs, Bs) * (1-1,1-As,1-As,1-As).
**PgSetAlpha(), PgSetAlphaCx()**

Pg_BLEND_SRC_A1Ad

Multiply the source pixel’s alpha channel by 1, and the rest of the pixel by Ad: Sm = (As, Rs, Gs, Bs) * (1, Ad, Ad, Ad).

Pg_BLEND_SRC_1mA1Ad

Multiply the source pixel’s alpha channel by 0, and the rest of the pixel by 1-Ad: Sm = (As, Rs, Gs, Bs) * (1-1, 1-Ad, 1-Ad, 1-Ad).

Pg_BLEND_SRC_A0As

Multiply the source pixel’s alpha channel by 0, and the rest of the pixel by As: Sm = (As, Rs, Gs, Bs) * (0, As, As, As).

Pg_BLEND_SRC_1mA0As

Multiply the source pixel’s alpha channel by 1, and the rest of the pixel by 1-As: Sm = (As, Rs, Gs, Bs) * (1-0, 1-As, 1-As, 1-As).

Pg_BLEND_SRC_A0Ad

Multiply the source pixel’s alpha channel by 0, and the rest of the pixel by Ad: Sm = (As, Rs, Gs, Bs) * (0, Ad, Ad, Ad).

Pg_BLEND_SRC_1mA0Ad

Multiply the source pixel’s alpha channel by 1, and the rest of the pixel by 1-Ad: Sm = (As, Rs, Gs, Bs) * (1-0, 1-Ad, 1-Ad, 1-Ad).

Pg_BLEND_SRC_SRC_ALPHA

Deprecated; use Pg_BLEND_SRC_As.

Pg_BLEND_DST_ONE_MINUS_SRC_ALPHA

Deprecated; use Pg_BLEND_DST_1mAAs.

**Destination multiplier flags**

Pg_BLEND_DST_0

Multiply the destination pixel by 0: Dm = (Ad, Rd, Gd, Bd) * (0, 0, 0, 0).

Pg_BLEND_DST_1

Multiply the destination pixel by 1 (no change): Dm = (Ad, Rd, Gd, Bd) * (1, 1, 1, 1).
PgSetAlpha(), PgSetAlphaCx()

Pg_BLEND_DST_S
Multiply the destination pixel by S: \( Dm = (Ad,Rd,Gd,Bd) \times (As,Rs,Gs,Bs) \).

Pg_BLEND_DST_1mS
Multiply the destination pixel by 1-S: \( Dm = (Ad,Rd,Gd,Bd) \times (1-As,1-Rs,1-Gs,1-Bs) \).

Pg_BLEND_DST_As
Multiply the destination pixel by As: \( Dm = (Ad,Rd,Gd,Bd) \times (As,As,As,As) \).

Pg_BLEND_DST_1mAs
Multiply the destination pixel by 1-As: \( Dm = (Ad,Rd,Gd,Bd) \times (1-As,1-As,1-As,1-As) \).

Pg_BLEND_DST_Ad
Multiply the destination pixel by Ad: \( Dm = (Ad,Rd,Gd,Bd) \times (Ad,Ad,Ad,Ad) \).

Pg_BLEND_DST_1mAd
Multiply the destination pixel by 1-Ad: \( Dm = (Ad,Rd,Gd,Bd) \times (1-Ad,1-Ad,1-Ad,1-Ad) \).

Pg_BLEND_DST_A1As
Multiply the destination pixel’s alpha channel by 1, and the rest of the pixel by As: \( Dm = (Ad,Rd,Gd,Bd) \times (1As,As,As) \).

Pg_BLEND_DST_1mA1As
Multiply the destination pixel’s alpha channel by 0, and the rest of the pixel by 1-As: \( Dm = (Ad,Rd,Gd,Bd) \times (1-1,1-As,1-As,1-As) \).

Pg_BLEND_DST_A1Ad
Multiply the destination pixel’s alpha channel by 1, and the rest of the pixel by Ad: \( Dm = (Ad,Rd,Gd,Bd) \times (1Ad,Ad,Ad) \).
**PgSetAlpha(), PgSetAlphaCx()**

Pg_BLEND_DST_1mA1Ad  
Multiply the destination pixel’s alpha channel by 0, and the rest of the pixel by 1-Ad:  
\[ Dm = (Ad, Rd, Gd, Bd) \times (1-1, 1-Ad, 1-Ad, 1-Ad). \]

Pg_BLEND_DST_A0As  
Multiply the destination pixel’s alpha channel by 0, and the rest of the pixel by As:  
\[ Dm = (Ad, Rd, Gd, Bd) \times (0, As, As, As). \]

Pg_BLEND_DST_1mA0As  
Multiply the destination pixel’s alpha channel by 1, and the rest of the pixel by 1-As:  
\[ Dm = (Ad, Rd, Gd, Bd) \times (1-0, 1-As, 1-As, 1-As). \]

Pg_BLEND_DST_A0Ad  
Multiply the destination pixel’s alpha channel by 0, and the rest of the pixel by Ad:  
\[ Dm = (Ad, Rd, Gd, Bd) \times (0, Ad, Ad, Ad). \]

Pg_BLEND_DST_1mA0Ad  
Multiply the destination pixel’s alpha channel by 1, and the rest of the pixel by 1-Ad:  
\[ Dm = (Ad, Rd, Gd, Bd) \times (1-0, 1-Ad, 1-Ad, 1-Ad). \]

**Alpha test flags:**

Pg_TEST_NEVER  
Never write source pixels to the destination. The destination remains unmodified.

Pg_TEST_ALWAYS  
Always write source pixels to the destination.

Pg_TEST_LESS_THAN  
If As is less than Ad, write the source pixel to the destination.

Pg_TEST_LESS_THAN_OR_EQUAL  
If As is less than or equal to Ad, write the source pixel to the destination.
PgTEST_EQUAL

If As is equal to Ad, write the source pixel to the destination.

PgTEST_GREATER_THAN_OR_EQUAL

If As is greater than or equal to Ad, write the source pixel to the destination.

PgTEST_GREATER_THAN

If As is greater than Ad, write the source pixel to the destination.

PgTEST_NOT_EQUAL

If As is not equal to Ad, write the source pixel to the destination.

Examples:

To use the alpha test option:

```
PgSetAlpha(Pg_ALPHA_OP_SRC_MAP | Pg_ALPHA_OP_DST_GLOBAL | 
            Pg_ALPHA_OP_TEST | Pg_TEST_LESS_THAN, &alpha_map, 
            NULL, 0, 0x80);
PgAlphaOn();
PgSetFillColor(Pg_WHITE);
PgDrawRect(&canvas_size, Pg_DRAW_FILL);
PgAlphaOff();
```

In this example, when the graphics driver is rendering the white rectangle:

- anywhere in the alphamap that is less than 128 (0x80) a white pixel is drawn
- anywhere that is equal to or greater than 128 in the alphamap the destination pixel is unmodified.

To use the alpha map option:

```
unsigned char alphamapdata[8][8]= {
    {0x00, 0x40, 0x80, 0xC0, 0xFF, 0xC0, 0x80, 0x40 },
    {0x40, 0x80, 0xC0, 0xFF, 0xC0, 0x80, 0x40, 0x00 },
    {0x80, 0xC0, 0xFF, 0xC0, 0x80, 0x40, 0x00, 0x40 },
    {0xC0, 0xFF, 0xC0, 0x80, 0x40, 0x00, 0x40, 0x80 },
    {0xFF, 0xC0, 0x80, 0x40, 0x00, 0x40, 0x80, 0xC0 }
}
```
PgSetAlpha(), PgSetAlphaCx()

\[
\begin{align*}
\{0xC0, 0x80, 0x40, 0x00, 0x40, 0x80, 0xC0, 0xFF \}, \\
\{0x80, 0x40, 0x00, 0x40, 0x80, 0xC0, 0xFF, 0xC0 \}, \\
\{0x40, 0x00, 0x40, 0x80, 0xC0, 0xFF, 0xC0, 0x80 \}
\end{align*}
\]

PgMap_t alphamap;
alphamap.dim.w=8;
alphamap.dim.h=8;
alphamap.bpl=8;
alphamap.bpp=8;
alphamap.map=alphamapdata;

PgSetAlpha (Pg_ALPHA_OP_SRC_MAP | Pg_BLEND_SRC_As |
            Pg_BLEND_DST_1mAs,
            &alphamap, NULL, 0, 0);

Classification:
Photon

_Safety_

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgAlphaOff*(), PgAlphaOn*(), PgMap_t, PgColor_t, PgSetAlphaBlend*, PhDim_t

“Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon _Programmer’s Guide_
**PgSetAlphaBlend(), PgSetAlphaBlendCx()**

Set the parameters for alpha blending simply

**Synopsis:**

```c
void PgSetAlphaBlend(
    PgMap_t const * const src_alpha_map,
    unsigned char const src_alpha_value);

void PgSetAlphaBlendCx(
    PhGC_t *gc,
    PgMap_t const * const src_alpha_map,
    unsigned char const src_alpha_value);
```

**Library:**

ph

**Description:**

These functions set the parameters for an alpha-blending operation.

Calling `PgSetAlphaBlend*()` is equivalent to calling `PgSetAlpha*()` with Pg_BLEND_SRC_As and Pg_BLEND_DST_1mAs as the source and destination multipliers.

The `src_alpha_map` argument is a pointer to the alpha map to be used in the blending operation. If this is NULL, the global blending factor, `src_alpha_value`, is used. For more information about the `PgMap_t` structure, see `PgMap_t`.

`PgSetAlphaBlend()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetAlphaBlendCx()`.

**Examples:**

```c
// Draw a purple rectangle blended (at 25%) over top of
// whatever is under it.
PgSetAlphaBlend(NULL, 0x40); // 64 / 256 = 0.25 or 25%
PgSetFillColor(Pg_PURPLE);
PgAlphaOn();
PgDrawIRect(0,0,99,99,Pg_DRAW_FILL);
PgAlphaOff();
PgFlush();
```
PgSetAlphaBlend(), PgSetAlphaBlendCx()

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:
PgAlphaOff(), PgAlphaOn(), PgMap_t, PgSetAlpha()

“Alpha Blending Support” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
\textit{PgSetChroma()}, \textit{PgSetChromaCx()}

\textit{Set the chroma color and operation}

**Synopsis:**

\begin{verbatim}
void PgSetChroma( PgColor_t ChromaColor,
                 unsigned long ChromaOp );

void PgSetChromaCx( PhGC_t *gc,
                     PgColor_t ChromaColor,
                     unsigned long ChromaOp );
\end{verbatim}

**Library:**

ph

**Description:**

These functions set the current chroma color and defines the chroma operation.

\textit{PgSetChroma()} works on the current graphics context, while you can specify the graphics context \textit{gc} for \textit{PgSetChromaCx()}.

\textit{ChromaColor} is the color to test against for chroma operations. 
\textit{ChromaOp} is made up of the following bits:

\begin{itemize}
  \item \texttt{Pg\_CHROMA\_SRC\_MATCH}  
    Test against the source data.
  \item \texttt{Pg\_CHROMA\_DEST\_MATCH}  
    Test against the destination data.
  \item \texttt{Pg\_CHROMA\_DRAW}  
    If the test pixel matches the chroma color, draw it in the destination.
  \item \texttt{Pg\_CHROMA\_NODRAW}  
    If the test pixel matches the chroma color, don’t draw it in the destination.
\end{itemize}
**PgSetChroma(), PgSetChromaCx()**

**Examples:**

```c
// Draw an image using a chroma color of full green

PhImage_t *image;
PhPoint_t p={0,0};

// Code to load the image...
;

PgChromaOn();
// Set the chroma operation to copy every pixel in the source to
// the destination, except the color 0x0000FF00 (bright green)
PgSetChroma(0x0000FF00, Pg_CHROMA_SRC_MATCH | Pg_CHROMA_NODRAW);
PgDrawPhImage(&p,image,0);
PgChromaOff();
PgFlush();
```

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgChromaOff*(), *PgChromaOn*(), *PgColor_t*,

“Chroma key support” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetClipping(), PgSetClippingCx()**

*Limit the extent of drawing*

**Synopsis:**

```c
void PgSetClipping( unsigned short n,
                    PhRect_t const *rects );

void PgSetClippingCx( PhGC_t *gc,
                       unsigned short n,
                       PhRect_t const *rects );
```

**Library:**

```
ph
```

**Description:**

These functions let you limit the extent of drawing by specifying which rectangles to draw in. Note that you can never draw outside of the current region.

*PgSetClipping()* works on the current graphics context, while you can specify the graphics context *gc* for *PgSetClippingCx()*.

The *n* argument contains the number of clip rectangles; *rects* is an array of *n* clip rectangles, relative to the origin of the current region. To reset the clipping rectangle to the full size of the region, you should pass 0 rectangles.

All subsequent draws will be clipped to the intersection of the clipping rectangles set by *PgSetClipping()*(*), *PgSetMultiClip()*(*), and *PgSetUserClip()*(*), and *PgSetRegion()*(*).

```
PhAttach(), PhReattach(), and PgSetRegion*() reset the clipping rectangle to the full size of the region.
```

Don’t call *PgSetClipping()* in a widget’s draw function; use *PgSetMultiClip()*(*), *PtClipAdd()*(*), and *PtClipRemove()* instead.

```
This function flushes the draw buffer.
```
**PgSetClipping(), PgSetClippingCx()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgFlush*(), *PgSetMultiClip*(), *PgSetRegion*(), *PgSetUserClip*(),
*PhAttach*, *PhReattach* PhRect_t, PtClipAdd, PtClipRemove*
Synopsis:

const PgColorModel_t * PgSetColorModel(
    PgColorModel_t const * model );

const PgColorModel_t * PgSetColorModelCx(
    PhGC_t *gc,
    PgColorModel_t const * model );

Library:

ph

Description:

These functions change the interpretation of colors represented by
PgColor_t by the Photon graphics library and io-graphics (see
the Utilities Reference).

PgSetColorModel() works on the current graphics context, while you
can specify the graphics context gc for PgSetColorModelCx().

For descriptions of the currently supported color models, see
PgColor_t.

Returns:

The previous color model.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

- `PgColor_t`, `PgGetColorModel`()
- `io-graphics` in the *Utilities Reference*
**Synopsis:**

```c
int PgSetDPMSMode (int mode);
```

**Library:**

`ph`

**Description:**

This function sets the display power-saving mode based on VESA Display Power Management System standards.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>On</td>
<td>The display is in full operation (Video: Active, Horizontal: Sync Pulses, Vertical: Sync Pulses)</td>
</tr>
<tr>
<td>1</td>
<td>Stand-by</td>
<td>An optional state of minimal power reduction with shortest display recovery time (Video: Blanked, Horizontal Sync: off, Vertical Sync: Pulses)</td>
</tr>
<tr>
<td>2</td>
<td>Suspend</td>
<td>A state with substantial power reduction, but the display recovery time can be longer than that of the stand-by state (Video: Blanked, Horizontal Sync: Pulses, Vertical Sync: off)</td>
</tr>
<tr>
<td>4</td>
<td>Off</td>
<td>The display is consuming the lowest level of power and is nonoperational (Video: Blanked, Horizontal Sync: off, Vertical Sync: off)</td>
</tr>
</tbody>
</table>

**Returns:**

- 0  Success.
- -1 An error occurred; `errno` is set.
PgSetDPMSMode()

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PgSetDrawBufferSize(), PgSetDrawBufferSizeCx()**

*Resize a draw buffer*

**Synopsis:**

```c
int PgSetDrawBufferSize(  
    unsigned short cmd_buf_len );

int PgSetDrawBufferSizeCx(  
    void *dc,  
    unsigned short cmd_buf_len );
```

**Library:**

ph

**Description:**

These functions resize the current draw buffer. The default size, allocated with every PhAttach(), is at least 4K. If `cmd_buf_len` is less than 1K, then 4K is allocated. The draw buffer stores all drawing data except for data stored in shared memory.

If the draw buffer contains unflushed data when these functions are called, the function will flush the data before reallocating the buffer.

`PgSetDrawBufferSize()` works on the current draw context, while you can specify the draw context `dc` for `PgSetDrawBufferSizeCx()`.

**Returns:**

- 0 Success.
- -1 An error occurred.

**Examples:**

```c
// Allocate a 16K draw buffer
PgSetDrawBufferSize( 16 * 1024 );
```

To reduce the memory requirements of the graphics driver, you should limit draw buffers to 16K.
**PgSetDrawBufferSize(), PgSetDrawBufferSizeCx()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PgClearDrawBuffer*, PgFlush*, PhAttach, PhGetMsgSize
**Synopsis:**

```c
int PgSetDrawMode( int mode );

int PgSetDrawModeCx( PhGC_t *gc,
                     int mode );
```

**Library:**

ph

**Description:**

These functions control how pixels are combined with video memory. 

*PgSetDrawMode()* works on the current graphics context, while you 
can specify the graphics context *gc* for *PgSetDrawModeCx()*.

**Photon 1.14 and earlier**

You can set *mode* to one of the following:

*Pg_DRAWMODE_OPAQUE*

Use default draw mode (the draw overwrites the screen).

*Pg_DRAWMODE_XOR*

XOR drawn pixels with the screen.

*Pg_DRAWMODE_AND*

AND drawn pixels with the screen.

*Pg_DRAWMODE_OR*

OR drawn pixels with the screen.
The effect of these functions depends on the physical video mode. If the video mode is “true color,” the RGB value being drawn will modify the RGB value of the pixel that’s in video memory. If the video mode is palette based, the palette index of the draw color will modify the palette index of the pixel that’s in video memory.

To facilitate XOR drawing, you can use the special draw color `Pg_INVERT_COLOR`. This color remains highly visible regardless of video mode (see `PgSetFillColor()`). If the video mode is true color, the graphics driver will XOR the screen pixels with pure white. If the video mode is palette based, the driver will invert the pixel index.

**Photon for QNX Neutrino**

Photon supports 256 raster operations. Operations can be done using a combination of source pixel data, destination pixel data, and color expanded monochrome pattern pixel data. Extended raster operations are set the same way the normal raster operations were set, using `PgSetDrawMode()`.

The extended raster operations are pervasive, meaning that they affect all subsequent drawing operations, including bit-blit operations and images. The old style raster operations still exist and behave as described above.

The extended raster operations are defined as `Pg_DrawModecharacters`, in reverse notation, where the characters are chosen from the following:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Pattern</td>
</tr>
<tr>
<td>S</td>
<td>Source</td>
</tr>
<tr>
<td>D</td>
<td>Destination</td>
</tr>
</tbody>
</table>

*continued…*
**PgSetDrawMode(), PgSetDrawModeCx()**

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>OR</td>
</tr>
<tr>
<td>a</td>
<td>AND</td>
</tr>
<tr>
<td>n</td>
<td>NOT</td>
</tr>
<tr>
<td>x</td>
<td>XOR</td>
</tr>
</tbody>
</table>

For example:

Pg_DrawModeS  
   Copy all source data.

Pg_DrawModePs  
   Logically OR the source data with the pattern data.

For a complete list of all raster operations available, see `<photon/Pg.h>`.

**Returns:**

The previous drawing mode.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PgSetDrawMode()**, **PgSetDrawModeCx()**

See also:

*PgDefaultFill*(), *PgDefaultMode*(), *PgSetFillColor*(),
*PgSetFillDither*(), *PgSetFillTransPat*(), *PgSetFillXORColor*(),
*PgSetStrokeColor*(), *PgSetStrokeDither*(), *PgSetStrokeTransPat*(),
*PgSetStrokeXORColor*(), *PgSetTextColor*(), *PgSetTextDither*(),
*PgSetTextTransPat*(), *PgSetTextXORColor*()

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetFillColor(), PgSetFillColorCx()**

*Set the fill color*

**Synopsis:**

```c
PgColor_t PgSetFillColor( PgColor_t color );

PgColor_t PgSetFillColorCx( PhGC_t *gc,
                           PgColor_t color );
```

**Library:**

`ph`

**Description:**

These functions set the fill color used for subsequent draws. If the driver doesn’t support 24-bit color, it selects the nearest color available to the one requested.

*PgSetFillColor() works on the current graphics context, while you can specify the graphics context `gc` for *PgSetFillColorCx()*.*

These functions override the color defined by *PgSetFillDither*().

You don’t need to set the fill color if you’re using widgets; the drawing attributes are set based on the widgets’ definitions and resources.

However, in all other cases you should set the fill color before you begin drawing. The defaults are undefined and drawing before setting the relevant attributes may produce unexpected results.

**Returns:**

The previous color.

**Examples:**

```c
/* Set the draw color to white. */
PgSetFillColor( Pg_WHITE );
```
**PgSetFillColor(), PgSetFillColorCx()**

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgARGB(), PgCMY(), PgColor_t, PgDefaultFill*(), PgGray(), PgHSV(), PgRGB(), PgSetDrawMode*(), PgSetFillDither*(), PgSetFillTransPat*(), PgSetFillXORColor*(), PgSetStrokeColor*(), PgSetTextColor*(*)

“Drawing attributes” and “Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetFillDither(), PgSetFillDitherCx()**

Set the dither pattern and colors for fills

**Synopsis:**

```c
void PgSetFillDither( PgColor_t c1,
                      PgColor_t c0,
                      PgPattern_t pat );

void PgSetFillDitherCx( PhGC_t *gc,
                         PgColor_t c1,
                         PgColor_t c0,
                         PgPattern_t pat );
```

**Library:**

`ph`

**Description:**

These functions combine two colors according to the pattern defined by `pat` and applies the pattern to fills.

The `c1` argument represents the color used for “on” bits in the dither pattern and `c0` represents the color used for “off” bits. The driver always selects the colors closest to `c1` and `c0`.

The dither pattern is an array of 8 bytes, aligned with the upper-left corner of the application’s region. This pattern repeats itself every 8 pixels horizontally and every 8 pixels vertically.

These functions override the color defined by the appropriate `PgSetFillColor()` function. For basic colors, see `PgColor_t`.

At least the following patterns are defined in `<photon/Pg.h>`:
**PgSetFillDither(), PgSetFillDitherCx()**

Predefined dither patterns.

*PgSetFillDither()* works on the current graphics context, while you can specify the graphics context *gc* for *PgSetFillDitherCx()*.
Examples:

// Set the fill to be black with every 8th vertical line being white
PgSetFillDither( Pg_WHITE, Pg_BLACK, Pg_PAT_VERT8 );

// Set the fill to be red bricks with dark gray mortar
PgSetFillDither(Pg_DGRAY, Pg_RED, "\x20\x20\xFF\x02\x02\x02\xFF\x20" );

Here’s the code that produced the sample of predefined dither patterns:

typedef struct {
    char *name;
    PgPattern_t p;
} DithersListStruct;

DithersListStruct DithersList[] = {
    "Pg_PAT_DEFAULT", Pg_PAT_DEFAULT,
    "Pg_PAT_HALF", Pg_PAT_HALF,
    "Pg_PAT_BACK_HALF", Pg_PAT_BACK_HALF,
    "Pg_PAT_CHECKB8", Pg_PAT_CHECKB8,
    "Pg_PAT_CHECKB4", Pg_PAT_CHECKB4,
    "Pg_PAT_DIAMOND", Pg_PAT_DIAMOND,
    "Pg_PAT_HORIZ8", Pg_PAT_HORIZ8,
    "Pg_PAT_HORIZ4", Pg_PAT_HORIZ4,
    "Pg_PAT_HORIZ2", Pg_PAT_HORIZ2,
    "Pg_PAT_VERT8", Pg_PAT_VERT8,
    "Pg_PAT_VERT4", Pg_PAT_VERT4,
    "Pg_PAT_VERT2", Pg_PAT_VERT2,
    "Pg_PAT_DIAGF8", Pg_PAT_DIAGF8,
    "Pg_PAT_DIAGF4", Pg_PAT_DIAGF4,
    "Pg_PAT_DIAGB8", Pg_PAT_DIAGB8,
    "Pg_PAT_DIAGB4", Pg_PAT_DIAGB4,
    "Pg_PAT_BRICK", Pg_PAT_BRICK,
    "Pg_PAT_WEAVE", Pg_PAT_WEAVE,
    "Pg_PAT_RXHATCH8", Pg_PAT_RXHATCH8,
    "Pg_PAT_RXHATCH4", Pg_PAT_RXHATCH4,
    "Pg_PAT_DXHATCH2", Pg_PAT_DXHATCH2,
    "Pg_PAT_DXHATCH8", Pg_PAT_DXHATCH8,
    "Pg_PAT_DXHATCH4", Pg_PAT_DXHATCH4,
};

#define DithersListNum \
    (sizeof( DithersList ) / sizeof( DithersListStruct ) )
#define DithersListCHeight 20
#define DithersListWinY (DithersListNum*DithersListCHeight)

Dithers();
DithersListStruct *DLPtr = DithersList;
PhPoint_t p;
PhRect_t r;
int i, y;
char Helvetica14b[MAX_FONT_TAG];

if(PfGenerateFontName(“Helvetica”, PF_STYLE_BOLD, 14,
                      Helvetica14b) == NULL) {
    perror(“Unable to find font”);
} else {
    PgSetFont( Helvetica14b );
}
PgSetTextColor( Pg_BLACK );
PgSetStrokeColor( Pg_BLACK );
for (y=i=0; i<DithersListNum;
     i++, y+=DithersListCHeight, DLPtr++) {
    p.x = 2;
p.y = y+14;
PgDrawText( DLPtr->name,
               strlen( DLPtr->name ), &p, 0 );
    PgSetFillDither( Pg_WHITE, Pg_DBLUE, DLPtr->p);
    r.ul.x = 160; r.lr.x = 320;
r.ul.y = y;    r.lr.y = y+DithersListCHeight;
PgDrawRect( &r, Pg_DRAW_FILL_STROKE );
}

Classification:

Photon

Safety

Interrupt handler  No
Signal handler No
Thread  No

See also:

PgARGB(), PgCMY(), PgColor_t, PgDefaultFill*, PgGray(),
PgHSV(), PgRGB(), PgSetFillColor*, PgSetFillTransPat*,
PgSetFillXORColor*, PgSetStrokeDither*, PgSetTextDither*()
PgSetFillDither(), PgSetFillDitherCx()

“Drawing attributes” and “Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide.
**PgSetFillTransPat(), PgSetFillTransPatCx()**

Set the draw transparency for fills

**Synopsis:**

```c
void PgSetFillTransPat( PgPattern_t pat );

void PgSetFillTransPatCx( PhGC_t *gc,
                           PgPattern_t pat );
```

**Library:**

ph

**Description:**

These functions set a masking pattern and applies it to fills. You should use them in combination with `PgSetFillColor*()` or `PgSetFillDither*()`.

`PgSetFillTransPat()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetFillTransPatCx()`.

These functions use the same patterns as `PgSetFillDither*()`. To disable transparency and draw normally, specify the `Pg_PAT_DEFAULT` pattern.

Because of speed considerations, some graphics drivers don’t draw strokes with a transparency mask and, as a result, ignore the stroke transparency pattern.

**Examples:**

```c
// let background show through for half the pixels
PgSetFillTransPat( Pg_PAT_HALF );

// disable transparency mask, draw normally
PgSetFillTransPat( Pg_PAT_DEFAULT );
```

**Classification:**

Photon
**PgSetFillTransPat()**, **PgSetFillTransPatCx()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgDefaultFill*(), *PgSetDrawMode*(), *PgSetFillColor*(),
*PgSetFillDither*(), *PgSetFillXORColor*(), *PgSetStrokeTransPat*(),
*PgSetTextTransPat*()

“Drawing attributes” and “Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetFillXORColor()**, **PgSetFillXORColorCx()**

*Set the fill color for XOR drawing*

**Synopsis:**

```c
void PgSetFillXORColor( PgColor_t frgd, 
                       PgColor_t bkgd );

void PgSetFillXORColorCx( PhGC_t *gc, 
                           PgColor_t frgd, 
                           PgColor_t bkgd );
```

**Library:**

`ph`

**Description:**

These functions set the draw color for fills. When an application XORs this color with the color `bkgd`, the result is the color `frgd`.

Since XOR is a reflexive function, `frgd` and `bkgd` may be reversed. **PgSetFillXORColor()** works on the current graphics context, while you can specify the graphics context `gc` for **PgSetFillXORColorCx()**.

**Examples:**

```c
DrawXOR() {
    char *s = "Hello World!";
    PhPoint_t p = { 8, 30 };
    PhRect_t r;
    char Helvetica18[MAX_FONT_TAG];

    if(PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
        perror ("Unable to find font");
    } else {
        PgSetFont( Helvetica18 );
        PgSetTextColor( Pg_YELLOW );
        PgSetFillColor( Pg_PURPLE );
        PgDrawText( s, strlen( s ), &p, Pg_BACK_FILL );

        PgExtentText( &r, &p, Helvetica18, s, strlen( s ) );
        r.lr.x -= (r.lr.x - r.ul.x) / 2;
        PgSetDrawMode( Pg_DRAWMODE_XOR );
        PgSetFillXORColor( Pg_YELLOW, Pg_PURPLE );
        PgDrawRect( &r, Pg_DRAW_FILL );
        PgSetDrawMode( Pg_DRAWMODE_OPAQUE );
    }
}
```

710 Chapter 8 • Pg—Graphics
\texttt{PgSetFillXORColor()}, \texttt{PgSetFillXORColorCx()} \)

The above code draws:

\begin{center}
\textbf{Hello World!}
\end{center}

**Classification:**

Photon

\begin{center}
\begin{tabular}{l c}
\textbf{Safety} & \\
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}
\end{center}

**See also:**

\texttt{PgColor_t}, \texttt{PgDefaultFill*()}, \texttt{PgSetDrawMode*()}, \texttt{PgSetFillColor*()}, \texttt{PgSetFillDither*()}, \texttt{PgSetFillTransPat*()}, \texttt{PgSetStrokeXORColor*()}, \texttt{PgSetTextXORColor*()}

“Drawing attributes” and “Arcs, ellipses, polygons, and rectangles” in the Raw Drawing and Animation chapter of the Photon \textit{Programmer’s Guide}
**PgSetFont(), PgSetFontCx()**

*Set the text font*

**Synopsis:**

```c
void PgSetFont( char const *ff );

void PgSetFontCx( PhGC_t *gc,
                     char const *ff );
```

**Library:**

`ph`

**Description:**

These functions set the font for text subsequently drawn with `PgDrawText*()` or `PgDrawString*()`. The `ff` argument is a pointer to a font identifier, which you should create by calling `PfGenerateFontName()`.

`PgSetFont()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetFontCx()`.

**Examples:**

```c
char font_name[MAX_FONT_TAG];

// Use Helvetica, 12p, Normal:
if (PfGenerateFontName( "Helvetica", 0, 12,
                        font_name ) != NULL) {
    PgSetFont( font_name );
}

// Use Helvetica, 14p, Bold Italic:
if (PfGenerateFontName( "Helvetica",
                        PF_STYLE_BOLD | PF_STYLE_ITALIC,
                        14, font_name ) != NULL ) {
    PgSetFont( font_name );
}
```
**PgSetFont(), PgSetFontCx()**

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfGenerateFontName(), PgDrawString*(), *PgDrawText*(), *PgSetFillColor*(), *PgSetFillDither*(), *PgSetFillTransPat*(), *PgSetUnderline*()

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetGC(), PgSetGCCx()**  
Set current graphics context

**Synopsis:**

```c
PhGC_t *PgSetGC( PhGC_t *GC );
```

```c
PhGC_t *PgSetGCCx( void *dc, 
                  PhGC_t *GC );
```

**Arguments:**

- `dc` **PgSetGCCx()** only. A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- `GC` A pointer to a graphics context, as returned by `PgCreateGC()`.

**Library:**

`ph`

**Description:**

These functions set the current graphics context to `GC`.

`PgSetGC()` works on the current draw context, while you can specify the draw context for `PgSetGCCx()`.

**Returns:**

A pointer to the previous graphics context.

**Classification:**

Photon
**PgSetGC(), PgSetGCCx()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgCreateGC(), PgDestroyGC(), PgGetGC*()
**Synopsis:**

```c
int PgSetLayerArg( int layer,
                   int arg,
                   void *data,
                   int data_len );
```

**Arguments:**

- `layer` The layer index, which must be 0 or greater.
- `arg` One of:
  - `Pg_LAYER_ARG_ACTIVE`
  - `Pg_LAYER_ARG_FORMAT_INDEX`
  - `Pg_LAYER_ARG_DST_VIEWPORT`
  - `Pg_LAYER_ARG_SRC_VIEWPORT`
  - `Pg_LAYER_ARG_BRIGHTNESS`
  - `Pg_LAYER_ARG_SATURATION`
  - `Pg_LAYER_ARG_CONTRAST`
  - `Pg_LAYER_ARG_CHROMA`
  - `Pg_LAYER_ARG_ALPHA`
  - `Pg_LAYER_ARG_EDGE_MODE`
  - `Pg_LAYER_ARG_FILTER_MODE`
  - `Pg_LAYER_ARG_LIST_BEGIN`
  - `Pg_LAYER_ARG_LIST_END`
  
  For more information, see “Layer arguments,” below.
- `data` A pointer to an argument-dependent value.
- `data_len` The size of the data.
**PgSetLayerArg()**

**Library:**

ph

**Description:**

`PgSetLayerArg()` configures the specified layer argument. This function replaces the existing value of the argument with the new value. It doesn’t free dynamically allocated data. If this function fails, then the value of the argument is undefined.

You must target this function at a device by calling `PdSetTargetDevice()`.

You can change layer arguments only after a call to `PgSetLayerArg()` with an argument of `Pg_LAYER_ARG_LIST_BEGIN`, and before another call with an argument of `Pg_LAYER_ARG_LIST_END`. For example:

```c
int fmt_idx = 5;
PgSetLayerArg( layer, Pg_LAYER_ARG_LIST_BEGIN, NULL, 0 );
PgSetLayerArg( layer, Pg_LAYER_ARG_FORMAT_INDEX,
              &fmt_idx, sizeof(int) );
PgSetLayerSurface( layer, 0, osc0 );
PgSetLayerSurface( layer, 1, osc1 );
PgSetLayerSurface( layer, 2, osc2 );
PgSetLayerArg( layer, Pg_LAYER_ARG_BRIGHTNESS, ... );
PgSetLayerArg( layer, ... );
...
PgSetLayerArg( layer, Pg_LAYER_ARG_LIST_END, NULL, 0 );
```

**Layer arguments**

The arguments for a layer are as follows:

**Pg_LAYER_ARG_ACTIVE**

Whether the layer is active (shown) or hidden.
**PgSetLayerArg()**

Pg\_LAYER\_ARG\_FORMAT\_INDEX

The layer format, by index. The index corresponds to the format\_index passed to *PgGetLayerCaps()*.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>≥ 0</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Pg\_LAYER\_ARG\_DST\_VIEWPORT

The rectangle on the screen where the source data is displayed.

<table>
<thead>
<tr>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhArea_t</td>
<td>The maximum area allowed by the driver</td>
</tr>
</tbody>
</table>

Pg\_LAYER\_ARG\_SRC\_VIEWPORT

The rectangle within the source data that’s displayed by the layer.

<table>
<thead>
<tr>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhArea_t</td>
<td>The same as the destination viewport, or the maximum area allowed by the driver, if smaller</td>
</tr>
</tbody>
</table>

Pg\_LAYER\_ARG\_BRIGHTNESS

The brightness level.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-128 to 127</td>
<td>0</td>
</tr>
</tbody>
</table>
PgLayerArg()

Pg_LAYER_ARG_SATURATION
The saturation level.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-128 to 127</td>
<td>0</td>
</tr>
</tbody>
</table>

Pg_LAYER_ARG_CONTRAST
The contrast level.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-128 to 127</td>
<td>0</td>
</tr>
</tbody>
</table>

Pg_LAYER_ARG_CHROMA
Chroma information, stored in a PgChroma_t structure:

```c
typedef struct _Pg_chroma {
    unsigned int color;
    unsigned long op;
} PgChroma_t;
```

The members include:
- **color** — a 32-bit color.
- **op** — the chroma operation; one of the Pg_CHROMA_* operations described for PgSetChroma().

<table>
<thead>
<tr>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgChroma_t</td>
<td>op = 0, color = N/A</td>
</tr>
</tbody>
</table>

Pg_LAYER_ARG_ALPHA
Alpha blending for the layer, stored in a PgLayerAlpha_t structure:

```c
typedef struct {
```
The members include:

- **op** — a bitwise OR of `Pg_LAYER_ALPHA_*` and `Pg_LAYER_BLEND_*` values, as described for `PgGetLayerCaps()`.
- **salpha** — the source alpha value (0 to 0xFF).
- **dalpha** — the destination alpha value (0 to 0xFF).

The **salpha** and **dalpha** members are ignored unless the **op** specifies that they should be used.

<table>
<thead>
<tr>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PgLayerAlpha_t op</code></td>
<td>0</td>
</tr>
</tbody>
</table>

**Pg_LAYER_ARG_EDGE_MODE**

How to behave if the source viewport is larger than the extent of the source data.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td><code>Pg_LAYER_EDGE_WRAP</code>, <code>Pg_LAYER_EDGE_CLAMP</code>, or 0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Pg_LAYER_ARG_FILTER_MODE**

Enable or disable viewport filtering.
**PgSetLayerArg()**

Bracket a set of calls to `PgSetLayerArg()` or `PgSetLayerSurface()` with these to queue up register updates.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Returns:**

- 0  Success.
- -1 An error occurred (`errno` is set).

**Errors:**

- EBUSY The layer is locked by another application.
- EINVAL The specified layer doesn’t exist, the layer doesn’t support the given argument, or the data specified for the argument is invalid.
- EOPNOTSUPP The layer doesn’t support this argument.

**Examples:**

**Classification:**

- Photon

**Safety**

- Interrupt handler: No

*continued...*
**PgSetLayerArg()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PdSetTargetDevice(), PgGetLayerCaps(), PgSetChroma(), PgSetLayerSurface(), PhArea_t*

“Layers” in the Raw Drawing and Animation chapter of the Photon
*Programmer’s Guide*
**Synopsis:**

```c
int PgSetLayerSurface( int layer,
                        int surface_index,
                        PdOffscreenContext_t *surface );
```

**Arguments:**

- `layer` The layer index, which must be 0 or greater.
- `surface_index` The surface index, which must be 0 or greater.
- `surface` A pointer to the `PdOffscreenContext_t` structure for the offscreen context, created by a call to `PgCreateLayerSurface()`, that you want to display. This argument must not be NULL.

**Library:**

`ph`

**Description:**

`PgSetLayerSurface()` tells a layer to fetch image data from the specified surface. If another surface was previously assigned to the layer for the same `surface_index`, it’s replaced but not destroyed.

You must target this function at a device by calling `PdSetTargetDevice()`.

You must configure the layer’s format before calling `PgSetLayerSurface()`. If a layer reads data from more than one surface, every surface read must have the same width and height.

You must use this function after a call to `PgSetLayerArg()` with an argument of `Pg_LAYER_ARG_LIST_BEGIN`, and before another call with an argument of `Pg_LAYER_ARG_LIST_END`. For example:

```c
int fmt_idx = 5;
PgSetLayerArg( layer, Pg_LAYER_ARG_LIST_BEGIN, NULL, 0 );
PgSetLayerArg( layer, Pg_LAYER_ARG_FORMAT_INDEX, 
               fmt_idx );
PgSetLayerArg( layer, Pg_LAYER_ARG_LIST_END, NULL, 0 );
```
PgSetLayerSurface()

&fmt_idx, sizeof(int) );
PgSetLayerSurface( layer, 0, osc0 );
PgSetLayerSurface( layer, 1, osc1 );
PgSetLayerSurface( layer, 2, osc2 );
PgSetLayerArg( layer, Pg_LAYER_ARG_BRIGHTNESS, ... );
PgSetLayerArg( layer, ... );
...  
PgSetLayerArg( layer, Pg_LAYER_ARG_LIST_END, NULL, 0 );

Returns:

0  Success.

-1  An error occurred (errno is set).

Errors:

EBUSY  The layer is locked by another application.

EINVAL  The specified layer or surface doesn’t exist, the osc argument is NULL, or the specified offscreen context is incompatible with the layer and/or surface.

EOPNOTSUPP  The operation isn’t supported.

EFAULT  The function couldn’t access the specified offscreen context.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 8 • Pg—Graphics
See also:

- `PdOffscreenContext_t`, `PdSetTargetDevice()`, `PgCreateLayerSurface()`, `PgSetLayerArg()`
- “Layers” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetMultiClip()**, **PgSetMultiClipCx()**

*Set a list of rectangles to clip drawing*

**Synopsis:**

```c
int PgSetMultiClip( int num,
                   PhRect_t const *clip_list );

int PgSetMultiClipCx( PhGC_t *gc,
                       int num,
                       PhRect_t const *clip_list );
```

**Library:**

`ph`

**Description:**

These functions set a list of rectangles (stored in `PhRect_t` structures) to clip subsequent drawing operations. The rectangles are always relative to the origin of the current region. To disable this clipping, set `clip_list` to NULL or `num` to 0.

All subsequent drawing operations will be clipped to the intersection of the clipping rectangles set by `PgSetClipping*()`, `PgSetMultiClip*()`, and `PgSetUserClip*()`.

*PhAttach(), PhReattach(), and PgSetRegion*() reset the clipping rectangle to the full size of the region.

This function emits a draw command.

`PgSetMultiClip()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetMultiClipCx()`.

**Returns:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>-1</td>
<td>Unable to allocate enough memory (using <code>malloc()</code>) to store the clipping rectangles.</td>
</tr>
</tbody>
</table>
Classification:

Photons

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PgSetClipping*(), *PgSetUserClip*(), *PhRect_t*
**PgSetPalette()**, **PgSetPaletteCx()**

*Set the color palette*

**Synopsis:**

```c
int PgSetPalette( PgColor_t const *palette,
                 long palette_id,
                 short first_color,
                 short num_colors,
                 int flags,
                 long tag);

int PgSetPaletteCx( PhGC_t *gc,
                    PgColor_t const *palette,
                    long palette_id,
                    short first_color,
                    short num_colors,
                    int flags,
                    long tag);
```

**Library:**

`ph`

**Description:**

These functions set the palette for subsequent draw commands. The palette can be either the graphics driver’s palette, or a private hardware or software palette. The `palette` argument points to a static buffer containing the palette; `first_color` denotes the first color to set, and `num_colors` defines how many palette entries to set.

The graphics driver uses the `palette_id` tag for caching. If the `palette_id` is 0, the `tag` will be used, as long as it is not 0 as well. If `palette_id` and `tag` are both 0, your region’s unique number is used as the palette ID. To have the graphics driver release a cached palette, set `num_colors` to -1.

A palette can operate in one of several modes. To determine the mode, set `flags` to one of the following:

- `Pg_PALSET_HARD`

  Used primarily for palette-based images; see `PhImage_t`.
Setting this palette type changes the physical palette. All colors set with a `PgSet...Color()` function will be chosen from this palette, for this process only. Other processes will continue to choose colors from the global palette and may appear incorrect. When you release the hardware palette, the other processes will return to their previous colors without being redrawn. You should always release the hardware palette when your window loses focus.

Direct-color and fixed-color graphics drivers will change this palette type to `Pg_PALSET_SOFT`.

**Pg_PALSET_HARDINACTIVE**

Same as `Pg_PALSET_HARD`, but doesn’t change the physical palette. You can use this to restore the global palette.

**Pg_PALSET_SOFT**

Used primarily for palette-based images; see `PhImage_t`. Since this type is completely handled by software in the graphics driver, it doesn’t affect the driver’s physical palette. Colors set with this palette type are unique to your graphics context.

**Pg_PALSET_HARDLOCKED**

Used for palette cycling or hardware-based flashing colors. Setting this type of palette prevents set colors from being involved in automatic color selection. To access these locked colors, you should OR the index with `Pg_INDEX_COLOR` when setting a color value; see `PgColor_t`.

To ensure that no other process is currently using the specified colors, you can OR this type with `Pg_PALSET_FORCE_EXPOSE`—this causes the screen to redraw.

**Pg_PALSET_GLOBAL**

Changes the physical palette. To ensure that all processes look correct, you can OR this type with `Pg_PALSET_FORCE_EXPOSE` to force all windows to redraw.
You can OR the above palette types with
Pg_PALSET_FORCE_EXPOSE to force an expose from the graphics
driver. This is useful when changing palettes; the expose causes all
applications to redraw with the new palette.

The tag argument is used for data caching by programs such as
phrelay (see the QNX Neutrino Utilities Reference). To calculate
the tag, use PtCRC(). This argument is ignored if you set it to 0.

PgSetPalette() works on the current graphics context, while you can
specify the graphics context gc for PgSetPaletteCx().

Returns:

0   Success.

-1   The draw buffer is too small to hold the current draw state and
     the draw command.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PgColor_t, PgDrawImage*(), PgFillColor*(),
PgSetStrokeColor*(), PgSetTextColor*(), PhImage_t, PtCRC()

“Drawing attributes” in the Raw Drawing and Animation chapter of
the Photon Programmer’s Guide
**Synopsis:**

```c
unsigned long PgSetPlaneMask( unsigned long mask );

unsigned long PgSetPlaneMaskCx( PhGC_t *gc, unsigned long mask );
```

**Library:**

`ph`

**Description:**

These functions protect planes of video memory from being modified. Each bit in the specified mask corresponds to a plane of video memory: a value of 0 enables access to the plane, a value of 1 protects the plane.

The effect of these functions depends on the physical video mode. If the video mode is “true color,” the mask will protect parts of the RGB value of the pixel that’s in video memory. If the video mode is palette based, the mask will protect parts of the palette index of the pixel that’s in video memory.

These functions work only on some 8-bit drivers.

*PgSetPlaneMask()* works on the current graphics context, while you can specify the graphics context `gc` for *PgSetPlaneMaskCx()*.

**Returns:**

The previous `mask`.

**Classification:**

Photon
**PgSetPlaneMask(), PgSetPlaneMaskCx()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDefaultMode(), PgSetFillColor*(), *PgSetStrokeColor*(),
*PgSetTextColor*()

“Drawing attributes” in the Raw Drawing and Animation chapter of the Photon *Programmer's Guide*
**PgSetRegion(), PgSetRegionCx()**

Specify which region will emit draw events

**Synopsis:**

```c
void PgSetRegion( PhRid_t rid );

void PgSetRegionCx( void *dc,
                    PhRid_t rid );
```

**Arguments:**

- `dc`  
  `PgSetGCCx()` only. A void pointer to any type of draw context. Examples of draw contexts are:
  - a `PhDrawContext_t` returned by `PhDCCreate()`
  - a `PmMemoryContext_t` returned by `PmMemCreateMC()`
  - a `PpPrintContext_t` returned by `PpCreatePC()`
  - a `PdOffscreenContext_t` returned by `PdCreateOffscreenContext()`

- `rid`  
  The region ID

**Library:**

`ph`

**Description:**

These functions specify which region will emit subsequent draw events.

If `rid` is the current region, this function does nothing.

If `rid` isn’t the current region, this function:

- Specifies `rid` as the region which will emit draw events.
- Resets the clipping rectangle to the full size of the region. Note that all draws are clipped to the region that emits them.
- Flushes the data before changing the current region, if the draw buffer contains unflushed data.

`PgSetRegion()` works on the current draw context, while you can specify the draw context for `PgSetRegionCx()`.
**PgSetRegion(), PgSetRegionCx()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgFlush*(), PgGetRegion*(), PgSetClipping*()`
Synopsis:

```c
int PgSetStrokeCap( int cap );

int PgSetStrokeCapCx( PhGC_t *gc, int cap );
```

Library:

`ph`

Description:

These functions determine how the ends of thick lines are drawn. You can set `cap` to one of the following:

- `Pg_BUTT_CAP` — the default.
- `Pg_POINT_CAP`
- `Pg_ROUND_CAP`
- `Pg_SQUARE_CAP`

```
Pg_BUTT_CAP

Pg_POINT_CAP

Pg_ROUND_CAP

Pg_SQUARE_CAP
```

*Styles for capping lines.*
PgSetStrokeCap(), PgSetStrokeCapCx()

The dotted lines in the above examples were added to illustrate how the caps relate to the original lines; they don’t normally appear.

*PgSetStrokeCap*() works on the current graphics context, while you can specify the graphics context gc for *PgSetStrokeCapCx*().

**Returns:**

The previous *cap* value.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDefaultStroke*(), *PgDrawEllipse*(), *PgDrawLine*(),
*PgDrawPolygon*(), *PgDrawRect*(), *PgDrawRoundRect*(),
*PgSetDrawMode*(), *PgSetStrokeDash*(), *PgSetStrokeDither*(),
*PgSetStrokeJoin*(), *PgSetStrokeTransPat*(), *PgSetStrokeWidth*(),
*PgSetStrokeXORColor*(),

**PgSetStrokeColor()**, **PgSetStrokeColorCx()**

*Set the stroke color*

**Synopsis:**

```c
PgColor_t PgSetStrokeColor( PgColor_t color );

PgColor_t PgSetStrokeColorCx( PhGC_t *gc, PgColor_t color );
```

**Library:**

ph

**Description:**

These functions set the stroke color used for subsequent draws. If the driver doesn’t support 24-bit color, it selects the nearest color available to the one requested.

These functions override the color defined by `PgSetStrokeDither*()`.

`PgSetStrokeColor()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetStrokeColorCx()`.

You don’t need to set stroke color if you’re using widgets; the drawing attributes are set based on the widgets’ definitions and resources.

However, in all other cases you should set the stroke color before you begin drawing. The defaults are undefined and drawing before setting the relevant attributes may produce unexpected results.

**Returns:**

The previous color.

**Classification:**

Photon
**PgSetStrokeColor()**, **PgSetStrokeColorCx()**

### Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

*PgARGB()*, *PgCMY()*, *PgColor_t*, *PgDefaultStroke*(), *PgGray()*, *PgHSV()*, *PgRGB()*, *PgDrawMode*(), *PgFillColor*(), *PgDefaultStroke*(), *PgSetStrokeCap*(), *PgSetStrokeDash*(), *PgSetStrokeDither*(), *PgSetStrokeJoin*(), *PgSetStrokeTransPat*(), *PgSetStrokeWidth*(), *PgSetStrokeXORColor*(), *PgTextColor*()

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetStrokeDash(), PgSetStrokeDashCx()**

*Set dashed lines*

**Synopsis:**

```c
void PgSetStrokeDash( unsigned char const *DashList,
                      int ListLen,
                      long DashScale );

void PgSetStrokeDashCx( PhGC_t *gc,
                        unsigned char const *DashList,
                        int ListLen,
                        long DashScale );
```

**Library:**

`ph`

**Description:**

These functions define a dash list that’s used to draw lines. The `DashList` argument points to an array of up to 16 characters. The values alternate between stroke values and space values: the first value defines a stroke length, the next defines a space length, the next after that defines a stroke length, and so on.

The values in `DashList` are scaled by the `DashScale` argument, which is 16.16 fixed point. The upper 16 bits are the integer part and the lower 16 the fractional part. The fractional part is in 65536ths, not 10ths, 100ths, etc.

For example, to specify a decimal scaling factor of 1.5:

1. Put 1 in the upper 16 bits.
2. Put 0.5 * 65536 = 32768 (i.e. 0x8000) in the lower 16 bits.

The resulting scaling parameter is 0x00018000.

To specify a decimal scaling of 47.75:

1. Put 47 (i.e. 0x2F) in the upper 16 bits.
2. Put 0.75 * 65536 = 49152 (i.e. 0xC000) in the lower 16 bits.

The resulting scaling parameter is 0x002FC000.
**PgSetStrokeDash()**, **PgSetStrokeDashCx()**

`PgSetStrokeDash()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetStrokeDashCx()`.

**Examples:**

```c
typedef struct {
    char *name;
    int l;
    char *p;
} DashListStruct;

/* NOTE: dash patterns are in octal */

DashListStruct DashList[] = {
    "solid", 0, NULL,
    "dotted", 1, "\1",
    "bigger dots", 1, "\2",
    "dashed", 2, "\10\4",
    "long dash", 2, "\40\4",
    "dash dot dot", 6, "\40\2\1\2\1\2",
    "long pattern", 16,
        "\3\2\5\2\10\2\13\2\15\2\13\2\10\2\5\2",
    "complex", 7, "\20\1\14\2\11\3\6",
};

#define DashListNum sizeof( DashList ) / sizeof( DashListStruct )
#define DashListCHeight 20
#define DashListWinY (DashListNum*DashListCHeight)

Dashes() {
    DashListStruct *DLPtr = DashList;
    PhPoint_t p;
    PhRect_t r;
    int i, y;
    char Helvetica14b[MAX_FONT_TAG];

    if(PfGenerateFontName("Helvetica", PF_STYLE_BOLD, 14, Helvetica14b) != NULL)
        PgSetFont( Helvetica14b );

    PgTextColor( Pg_WHITE );
    PgSetTextColor( Pg_WHITE );
    for (y=i=0; i<DashListNum; i++,
        y+=DashListCHeight, DLPtr++) {
        p.x = 2;
        p.y = y+14;
        PgDrawText( DLPtr->name, strlen( DLPtr->name ),
            &p, 0 );
        PgSetStrokeDash( DLPtr->p, DLPtr->l, 0x10000 );
    }
}
```

740  Chapter 8  *Pg—Graphics*
The above code draws:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>solid</td>
<td>____________________________</td>
</tr>
<tr>
<td>dotted</td>
<td>____________________________</td>
</tr>
<tr>
<td>bigger dots</td>
<td>__________________</td>
</tr>
<tr>
<td>dashed</td>
<td>_________________________</td>
</tr>
<tr>
<td>long dash</td>
<td>____________</td>
</tr>
</tbody>
</table>
| dash dot dot | ___________
| long pattern | ___________
| complex | ________

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PgDefaultStroke*(), *PgDrawLine*(), *PgDrawPolygon*(),
*PgDrawRect*(), *PgSetDrawMode*(), *PgSetStrokeCap*(),
*PgSetStrokeDither*(), *PgSetStrokeJoin*(), *PgSetStrokeTransPat*(),
*PgSetStrokeWidth*(), *PgSetStrokeXORColor*()

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetStrokeDither(), PgSetStrokeDitherCx()**

*Set the stroke dither pattern*

**Synopsis:**
```c
void PgSetStrokeDither( PgColor_t c1,
                        PgColor_t c0,
                        PgPattern_t pat );

void PgSetStrokeDitherCx( PhGC_t *gc,
                           PgColor_t c1,
                           PgColor_t c0,
                           PgPattern_t pat );
```

**Library:**

ph

**Description:**

These functions combine two colors according to the pattern defined by `pat` and applies the pattern to outlines.

The `c1` argument represents the color used for “on” bits in the dither pattern and `c0` represents the color used for “off” bits. The driver always selects the colors closest to `c1` and `c0`.

The dither pattern is an array of 8 bytes, aligned with the upper-left corner of the application’s region. This pattern repeats itself every 8 pixels horizontally and every 8 pixels vertically. For a sample of dither patterns, see `PgSetFillDither*()`.

Because of speed considerations, some graphics drivers don’t dither strokes. If a driver doesn’t support dithering, it uses `c1` to draw strokes.

These functions override the color defined by `PgSetStrokeColor*()`. For basic colors, see `PgColor_t`.

`PgSetStrokeDither()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetStrokeDitherCx()`.
**PgSetStrokeDither(), PgSetStrokeDitherCx()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PgARGB(), PgCMY(), PgColor_t, PgDefaultStroke*(), PgGray(), PgHSV(), PgRGB(), PgSetDrawMode*(), PgSetFillDither*(), PgSetStrokeCap*(), PgSetStrokeColor*(), PgSetStrokeDash*(), PgSetStrokeJoin*(), PgSetStrokeTransPat*(), PgSetStrokeWidth*(), PgSetStrokeXORColor*(), PgSetTextDither*()

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PgSetStrokeJoin( int join );

int PgSetStrokeJoinCx( PhGC_t *gc, int join );
```

**Library:**

`ph`

**Description:**

These functions determine how thick lines are connected. You can set `join` to one of the following:

- `Pg_BEVEL_JOIN`
- `Pg_BUTT_JOIN`
- `Pg_MITER_JOIN` — the default.
- `Pg_QROUND_JOIN` — a quick simulated rounded joint.
- `Pg_ROUND_JOIN`
Styles for joining lines.
**PgSetStrokeJoin(), PgSetStrokeJoinCx()**

The dotted lines in the above examples were added to illustrate how the joints relate to the original lines; they don’t normally appear.

*PgSetStrokeJoin()* works on the current graphics context, while you can specify the graphics context *gc* for *PgSetStrokeJoinCx()*.

**Returns:**

The previous *join* value.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PgDefaultStroke*(), *PgDrawEllipse*(), *PgDrawLine*(),
*PgDrawPolygon*(), *PgDrawRect*(), *PgDrawRoundRect*(),
*PgSetDrawMode*(), *PgSetStrokeCap*(), *PgSetStrokeColor*(),
*PgSetStrokeDash*(), *PgSetStrokeDither*() *PgSetStrokeTransPat*(),
*PgSetStrokeWidth*(), *PgSetStrokeXORColor*(),

**PgSetStrokeTransPat(), PgSetStrokeTransPatCx()**

Set the draw transparency for strokes

**Synopsis:**

```c
void PgSetStrokeTransPat( PgPattern_t pat );

void PgSetStrokeTransPatCx( PhGC_t *gc,
                           PgPattern_t pat );
```

**Library:**

`ph`

**Description:**

These functions set a masking pattern and applies it to outlines. You should use it in combination with `PgSetStrokeColor*()` or `PgSetStrokeDither*()`. These functions use the same patterns as `PgSetFillDither*()`. To disable transparency and draw normally, specify the `Pg_PAT_DEFAULT` pattern.

Because of speed considerations, some graphics drivers don’t draw strokes with a transparency mask and, as a result, ignore the stroke transparency pattern.

`PgSetStrokeTransPat()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetStrokeTransPatCx()`.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 8 • Pg—Graphics 747
See also:

*PgDefaultStroke*(), *PgSetDrawMode*(), *PgSetFillTransPat*(),
*PgSetStrokeCap*(), *PgSetStrokeColor*(), *PgSetStrokeDash*(),
*PgSetStrokeDither*(), *PgSetStrokeJoin*(), *PgSetStrokeWidth*(),
*PgSetStrokeXORColor*(), *PgSetTextTransPat*()

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw
Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PgSetStrokeWidth( int width );

long PgSetStrokeFWidth( long width );

int PgSetStrokeWidthCx( PhGC_t *gc, int width );

long PgSetStrokeFWidthCx( PhGC_t *gc, long width );
```

**Library:**

`ph`

**Description:**

These functions set the thickness of lines. If you call `PgSetStrokeWidth*()` , the `width` argument takes an integer that indicates the width of the line in pixels. But if you call `PgSetStrokeFWidth*()` , the `width` argument takes a pixel width multiplied by 65,536 (`0x10000`). For example, specifying a value of `0x80000` will set the line width to 8 pixels.

The minimum line width for `PgSetStrokeFWidth*()` is one pixel.

`PgSetStrokeWidth()` and `PgSetStrokeFWidth()` work on the current graphics context, while you can specify the graphics context `gc` for `PgSetStrokeWidthCx()` and `PgSetStrokeFWidthCx()`.

**Returns:**

The previous width.

**Classification:**

Photon
**PgSetStrokeWidth(), PgSetStrokeFWidth(), PgSetStrokeWidthCx(), PgSetStrokeFWidthCx()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**Caveats:**

We don’t recommend using a line width greater than one pixel. Some graphics drivers might give unexpected results.

**See also:**

`PgDefaultStroke*(), PgDrawEllipse*(), PgDrawLine*(), PgDrawPolygon*(), PgDrawRect*(), PgDrawRoundRect*(), PgSetDrawMode*(), PgSetStrokeCap*(), PgSetStrokeColor*(), PgSetStrokeDash*(), PgSetStrokeDither*(), PgSetStrokeJoin*(), PgSetStrokeTransPat*(), PgSetStrokeXORColor*()`

**PgSetStrokeXORColor(),**
**PgSetStrokeXORColorCx()**

*Set the stroke color for XOR drawing*

**Synopsis:**

```c
void PgSetStrokeXORColor( PgColor_t frgd,
                          PgColor_t bkgd );

void PgSetStrokeXORColorCx( PhGC_t *gc,
                            PgColor_t frgd,
                            PgColor_t bkgd );
```

**Library:**

ph

**Description:**

These functions set the draw color for outlines. When an application XORs this color with the color `bkgd`, the result is the color `frgd`.

Since XOR is a reflexive function, `frgd` and `bkgd` may be reversed.

`PgSetStrokeXORColor()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetStrokeXORColorCx()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgColor_t`, `PgDefaultStroke*()`, `PgSetDrawMode*()`,
`PgSetFillXORColor*()`, `PgSetStrokeCap*()`, `PgSetStrokeColor*()`,
`PgSetStrokeDash*()`, `PgSetStrokeDither*()`, `PgSetStrokeJoin*()`,
**PgSetStrokeXORColor(), PgSetStrokeXORColorCx()**

*PgSetStrokeTransPat*(), *PgSetStrokeWidth*(), *PgSetTextXORColor*(

“Drawing attributes” and “Lines, pixels, and pixel arrays” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

\[
PgColor_t \text{ PgSetTextColor( PgColor_t color )};
\]

\[
PgColor_t \text{ PgSetTextColorCx( PhGC_t *gc, PgColor_t color )};
\]

**Library:**

ph

**Description:**

These functions set the color used for text and bitmaps in subsequent draws. If the driver doesn’t support 24-bit color, it selects the nearest color available to the one requested.

This function overrides the color defined by \textit{PgSetTextDither*()}. \textit{PgSetTextColor()} works on the current graphics context, while you can specify the graphics context \textit{gc} for \textit{PgSetTextColorCx()}.

**Returns:**

The previous color.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PgARGB(), PgCMY(), PgColor_t, PgDefaultText*,
PgDrawString*, PgDrawText*, PgDrawTextArea*,
PgExtentMultiText*, PgExtentText*, PgGray(), PgHSV(),
PgRGB(), PgSetDrawMode*, PgSetFillColor*, PgSetFont*,
PgSetStrokeColor*, PgSetTextDither*, PgSetTextTransPat*,
PgSetTextXORColor*, PgSetUnderline*

“Drawing attributes” and “Text” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetTextDither(), PgSetTextDitherCx()**

Set the dither pattern for text and bitmap

**Synopsis:**

```c
void PgSetTextDither( PgColor_t c1,
                      PgColor_t c0,
                      PgPattern_t pat );

void PgSetTextDitherCx( PhGC_t *gc,
                         PgColor_t c1,
                         PgColor_t c0,
                         PgPattern_t pat );
```

**Library:**

ph

**Description:**

These functions combine two colors according to the pattern defined by `pat` and applies the pattern to text and bitmaps.

The `c1` argument represents the color used for “on” bits in the dither pattern and `c0` represents the color used for “off” bits. The driver always selects the colors closest to `c1` and `c0`.

The dither pattern is an array of 8 bytes, aligned with the upper-left corner of the application’s region. This pattern repeats itself every 8 pixels horizontally and every 8 pixels vertically. For a sample of dither patterns, see `PgSetFillDither*()`.

These functions override the color defined by `PgSetTextColor*()`. For basic colors, see `PgColor_t`.

`PgSetTextDither()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetTextDitherCx()`.

**Classification:**

Photon
**PgSetTextDither(), PgSetTextDitherCx()**

### Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PgRGB()`, `PgColor_t`, `PgDefaultText*()`, `PgDrawString*()`,
- `PgDrawText*()`, `PgDrawTextArea*()`, `PgExtentMultiText*()`,
- `PgExtentText*()`, `PgGray()`, `PgHSV()`, `PgRGB()`, `PgSetDrawMode*()`,
- `PgSetFillDither*()`, `PgSetFont*()`, `PgSetStrokeDither*()`,
- `PgSetTextColor*()`, `PgSetTextTransPat*()`, `PgSetTextXORColor*()`,
- `PgSetUnderline*()`

“Drawing attributes” and “Text” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetTextTransPat(), PgSetTextTransPatCx()**

*Set the draw transparency for text and bitmaps*

**Synopsis:**

```c
void PgSetTextTransPat( PgPattern_t pat );

void PgSetTextTransPatCx( PhGC_t *gc,
                           PgPattern_t pat );
```

**Library:**

`ph`

**Description:**

These functions set a masking pattern and applies it to text and bitmaps. You should use it in combination with *PgSetTextColor*() or *PgSetTextDither*().

These functions use the same patterns as *PgSetFillDither*(). To disable transparency and draw normally, specify the `Pg_PAT_DEFAULT` pattern.

Because of speed considerations, some graphics drivers don’t draw strokes with a transparency mask and, as a result, ignore the stroke transparency pattern.

*PgSetTextTransPat() works on the current graphics context, while you can specify the graphics context `gc` for *PgSetTextTransPatCx()*.*

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 8 • Pg—Graphics 757
**PgSetTextTransPat(), PgSetTextTransPatCx()**

See also:

*PgDefaultText*(), *PgDrawString*(), *PgDrawText*(),
*PgDrawTextArea*(), *PgExtentMultiText*(), *PgExtentText*(),
*PgGray()*, *PgHSV()*, *PgRGB()*, *PgSetDrawMode*(),
*PgSetFillTransPat*(), *PgSetFont*(), *PgSetStrokeTransPat*(),
*PgSetTextColor*(), *PgSetTextDither*(), *PgSetTextXORColor*(),
*PgSetUnderline*()

“Drawing attributes” and “Text” in the Raw Drawing and Animation chapter of the Photon **Programmer’s Guide**
**PgSetTextXORColor(), PgSetTextXORColorCx()**

Set the text and bitmap color for XOR drawing

### Synopsis:

```c
void PgSetTextXORColor( PgColor_t frgd,
                        PgColor_t bkgd );

void PgSetTextXORColorCx( PhGC_t *gc,
                          PgColor_t frgd,
                          PgColor_t bkgd );
```

### Library:

ph

### Description:

These functions set the draw color for text and bitmaps. When an application XORs this color with the color `bkgd`, the result is the color `frgd`.

Since XOR is a reflexive function, `frgd` and `bkgd` may be reversed. `PgSetTextXORColor()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetTextXORColorCx()`.

### Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

### See also:

`PgColor_t, PgDefaultText*(), PgDrawString*(), PgDrawText*(), PgDrawTextArea*(), PgExtentMultiText(), PgExtentText(), PgSetDrawMode*(), PgSetFillXORColor*(), PgSetFont*()`
PgSetTextXORColor(), PgSetTextXORColorCx()

PgSetStrokeXORColor*(), PgSetTextColor*(), PgSetTextDither*(), PgSetTextTransPat*(), PgSetUnderline*()

“Drawing attributes” and “Text” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetTranslation(), PgSetTranslationCx()**

*Translate draw commands horizontally and vertically*

**Synopsis:**

```c
void PgSetTranslation ( PhPoint_t const *translation,
                        int flags );

void PgSetTranslationCx ( PhGC_t *gc,
                          PhPoint_t const *translation,
                          int flags );
```

**Library:**

*ph*

**Description:**

These functions cause all subsequent draw commands to be translated by `translation->x` pixels horizontally and `translation->y` pixels vertically. The default translation is (0,0). You can set `flags` to:

- **0** The translation is absolute, and replaces the current one.
- **Pg_RELATIVE** The translation is relative to the current translation, and is added to it.

`PgSetTranslation()` works on the current graphics context, while you can specify the graphics context `gc` for `PgSetTranslationCx()`.

**Examples:**

Draw a square from (100,100) to (200,200):

```c
PhPoint_t translation;
PgSetFillColor ( Pg_RED );
translation.x = translation.y = 100;
PgSetTranslation ( &translation, Pg_RELATIVE );
PgDrawIRect ( 0, 0, 100, 100, Pg_DRAW_FILL );
```
PgSetTranslation(), PgSetTranslationCx()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgClearTranslation*(), PhPoint_t

“PtRaw widget” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PgSetUnderline(), PgSetUnderlineCx()**

Set colors for underlining text

Synopsis:

```c
void PgSetUnderline( PgColor_t c1,
                     PgColor_t c2,
                     int flags );

void PgSetUnderlineCx( PhGC_t *gc,
                        PgColor_t c1,
                        PgColor_t c2,
                        int flags );
```

Library:

`ph`

Description:

These functions set the color or colors used for underlining text:

<table>
<thead>
<tr>
<th><code>c1</code></th>
<th><code>c2</code></th>
<th>Underline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pg_TRANSPARENT</td>
<td>N/A</td>
<td>Disabled</td>
</tr>
<tr>
<td>Any color</td>
<td>Pg_TRANSPARENT</td>
<td>Single underline</td>
</tr>
<tr>
<td>Any color</td>
<td>Any color</td>
<td>Double underline</td>
</tr>
</tbody>
</table>

You should find double underlining useful for scored underlining (where `c2` is a shadow color) or for thick underlining (where both `c1` and `c2` are the same color).

No flags are currently defined.

These functions affect only the drawing operations that involve text:

- *PgDrawMultiTextArea*()
- *PgDrawString*()
- *PgDrawText*()
- *PgDrawTextArea*()
**PgSetUnderline(), PgSetUnderlineCx()**

*PgSetUnderline()* works on the current graphics context, while you can specify the graphics context gc for *PgSetUnderlineCx()*.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgColor_t`, `PgDefaultText()`, `PgDrawMultiTextArea*()`, `PgDrawString*()`, `PgDrawText*()`, `PgDrawTextArea*()`, `PgExtentMultiText*()`, `PgExtentText()`, `PgSetDrawMode*()`, `PgSetFont*()`, `PgSetTextColor*()`, `PgSetTextDither*()`, `PgSetTextTransPat*()`, `PgSetTextXORColor*()`

“Drawing attributes” and “Text” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgSetUserClip(), PgSetUserClipAbsolute(), PgSetUserClipCx(), PgSetUserClipAbsoluteCx()**

Restrict subsequent draws

### Synopsis:

```c
void PgSetUserClip( PhRect_t const *ClipRect );
void PgSetUserClipAbsolute( PhRect_t const *ClipRect );
void PgSetUserClipCx( PhGC_t *gc, PhRect_t const *ClipRect );
void PgSetUserClipAbsoluteCx( PhGC_t *gc, PhRect_t const *ClipRect );
```

### Library:

`ph`

### Description:

These functions restrict all subsequent draws to the area defined by the `PhRect_t` pointed to by `ClipRect`. To disable the user clipping rectangle, pass `ClipRect` as NULL.

`PgSetUserClip*()` sets the user clipping rectangle relative to the current translation whereas `PgSetUserClipAbsolute*()` sets the rectangle independent of the current translation.

The user clipping area is set independent of the clipping that’s set by `PgSetClipping*()` and `PgSetMultiClip*()`.

All subsequent draws will be clipped to the intersection of the clipping rectangles set by `PgSetClipping*()` , `PgSetMultiClip*()` , and `PgSetUserClip*()`.

Unlike `PgSetClipping*()` , these functions don’t flush the draw buffer.

---

`PhAttach()`, `PhReattach()`, and `PgSetRegion()` reset the clipping rectangle to the full size of the region.
These functions emit a draw command.

`PgSetUserClip()` and `PgSetUserClipAbsolute()` work on the current graphics context, while you can specify the graphics context `gc` for `PgSetUserClipCx()` and `PgSetUserClipAbsoluteCx()`.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgClearTranslation()`, `PgSetClipping()`, `PgSetMultiClip()`, `PgSetRegion()`, `PgSetTranslation()`, `PhAttach()`, `PhReattach()`, `PhRect_t`
**Synopsis:**

```c
int PgSetVideoMode( PgDisplaySettings_t *settings );
```

**Library:**

```
ph
```

**Description:**

This function sets the current video mode to the settings given in the `PgDisplaySettings_t` structure pointed to by `settings`, which includes at least the following:

- `unsigned mode`
  The number of the current mode for the video card.
- `int refresh`
  The refresh rate, in Hz. A refresh rate of 0 requests the default rate for this mode (usually 60Hz).
- `unsigned flags`
  There are currently no flags defined.

You must target this function at a specific card by calling `PdSetTargetDevice()`. `PgSetVideoMode()` blocks until the operation is complete.

**Returns:**

- 0  Success.
- -1  An error occurred.

**Classification:**

Photon
**PgSetVideoMode()**

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgGetGraphicsHWCaps(), PgGetVideoMode(), PgGetVideoModeInfo(), PgGetVideoModeList()*

“Video modes” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PgShmemAttach( char const *name,
                   unsigned long size,
                   void *addr );
```

**Library:**

`ph`

**Description:**

This function records a reference to an existing block of shared memory (that is, a block created with `shm_open()`), sized with `ftruncate()`, and mapped into the process’s address space with `mmap()` — see the QNX Neutrino Library Reference).

The Photon library uses `atexit()` to arrange for `PgShmemCleanup()` to be called when your program terminates normally. If your program terminates abnormally, it should call `PgShmemCleanup()` explicitly.

**Returns:**

- 0 Successful completion.
- -1 An error occurred ( `errno` is set).

**Errors:**

See the errors for `shm_open()` in the QNX Neutrino Library Reference.

**Classification:**

Photon

**Safety**

- Interrupt handler    No

  *continued...
**PgShmemAttach()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PgDrawBitmapmx(), PgDrawImagemx(), PgShmemCleanup(), PgShmemCreate(), PgShmemDestroy(), PgShmemDetach()

atexit(), errno, ftruncate(), mmap(), shm_open() in the QNX Neutrino Library Reference
Synopsis:

```c
void PgShmemCleanup();
```

Library:

```
ph
```

Description:

This function removes all shared memory references that you defined with `PgShmemCreate()` and `PgShmemAttach()`. If you created the block with `PgShmemCreate()`, the block is unlinked.

The Photon library uses `atexit()` to arrange for `PgShmemCleanup()` to be called when your program terminates normally. If your program terminates abnormally, it should call `PgShmemCleanup()` explicitly.

Examples:

This code fragment shows how you can use `PgShmemCleanup()` in a signal handler:

```c
void ExitCleanup( int sig ) {
    sig = sig;
    PgShmemCleanup();
    _exit( 1 );
}
```

```c
main( ... ) {
    ...
    signal( SIGTERM, ExitCleanup );
    signal( SIGHUP, ExitCleanup );
    signal( SIGQUIT, ExitCleanup );
    signal( SIGINT, ExitCleanup );

    /* main loop */
    ...
}
```
**PgShmemCleanup()**

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- *PgShmemAttach(), PgShmemCreate()*
- *atexit() in the QNX Neutrino Library Reference*
Synopsis:

void *PgShmemCreate( unsigned long size,
                     char const *name );

Library:

ph

Description:

This function creates a block of shared memory. The size argument determines the size of the block.

If you pass name as NULL, this function generates a unique name in the form Pg############; this is the preferred mode of operation. If you pass a name, make sure that it isn’t already in use.

You must use the “mx” form of a draw function to pass the shared memory reference. Otherwise, the data is copied into the draw event.

The Photon library uses atexit() to arrange for PgShmemCleanup() to be called when your program terminates normally. If your program terminates abnormally, it should call PgShmemCleanup() explicitly.

Returns:

A local pointer to shared memory. If an error occurs, it returns NULL and sets errno.

Errors:

See the errors for shm_open() in the QNX Neutrino Library Reference.

Classification:

Photon
**PgShmemCreate()**

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgDrawBitmapmx()*, *PgDrawImagemx()*, *PgShmemAttach()*, *PgShmemCleanup()*, *PgShmemDestroy()*, *PgShmemDetach()*

*shm_open()* in the QNX Neutrino Library Reference
**PgShmemDestroy()**

*Remove a block of shared memory*

**Synopsis:**

```c
int PgShmemDestroy( void *addr );
```

**Library:**

`ph`

**Description:**

This function removes a block of shared memory created with `PgShmemCreate()`. The block is referenced by the address returned from `PgShmemCreate()`.

**Returns:**

- 0    Successful completion.
- -1   An error occurred (`errno` will be set).

**Errors:**

See the errors for `shm_unlink()` in the QNX Neutrino *Library Reference*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PgShmemDestroy()**

See also:

*PgShmemCreate()*
**Synopsis:**

```c
int PgShmemDetach( void *addr );
```

**Library:**

`ph`

**Description:**

This function removes a shared memory reference previously attached with `PgShmemAttach()`.

The shared memory object persists until no other applications refer to it. Don’t use the same name for another shared memory object, especially right after detaching the first one.

**Returns:**

- 0 Success.
- -1 An error occurred (`errno` is set).

**Errors:**

See the errors for `shm_unlink()` in the QNX Neutrino Library Reference.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PgShmemDetach()

See also:

PgShmemAttach()
**Synopsis:**

```c
int PgSwapDisplay( PdOffscreenContext_t *osc,
                   unsigned long flags );

int PgSwapDisplayCx( PhGC_t *gc,
                      PdOffscreenContext_t *osc,
                      unsigned long flags );
```

**Library:**

ph

**Description:**

These functions point the CRT of the video display at the context indicated by `osc`. These functions can be used for double and triple buffering. They’re available only in direct mode.

The `flags` argument is a combination of the following bits:

- **Pg_SWAP_BLIT**
  - Blit the contents of the new target to the old one.

- **Pg_SWAP_VSYNC**
  - Wait for a Vsync to occur before continuing to parse the draw stream.

To guarantee that you can point the CRT at this target, you should create it with the flag `Pg_OSC_CRTC_SAFE`.

**Returns:**

- 0  Success.
- -1 An error occurred.
**PgSwapDisplay(), PgSwapDisplayCx()**

**Examples:**

This example of double buffering assumes we’re in direct mode already:

```c
PdOffscreenContext_t *buf[2];
int cur_buf=1;

// Create an offscreen context from the current screen:
buf[0] = PdCreateOffscreenContext(0,0,0,Pg_OSC_MAIN_DISPLAY);

// Duplicate the buffer:
buf[1] = PdDupOffscreenContext(buf[0],Pg_OSC_CRTC_SAFE);

while (not_done)
{
    PhDCSetCurrent(buf[cur_buf]);
    RenderMyFrame();
    PgSwapDisplay(buf[cur_buf],0);
    PgFlush();
    cur_buf = cur_buf ? 0 : 1;
}
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PdCreateOffscreenContext(), PdDupOffscreenContext(), PdGetOffscreenContextPtr(), PdOffscreenContext_t, PgContextBlit()*

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PgUnlockLayer()**

**Unlock a layer**

**Synopsis:**

```c
int PgUnlockLayer( int layer );
```

**Arguments:**

- `layer`  The layer index, which must be 0 or greater.

**Library:**

```
ph
```

**Description:**

`PgUnlockLayer()` releases a layer from exclusive use by an application. To lock a layer, call `PgLockLayer()`.

Other applications may not use `PgSetLayerSurface()` or `PgSetLayerArg()` on a locked surface.

You must target this function at a device by calling `PdSetTargetDevice()`.

Your application should unlock its layers before it exits. You can lock a layer multiple times, but need to unlock it only once.

**Returns:**

- `0`  Success.
- `-1`  An error occurred (`errno` is set).

**Errors:**

- `EBUSY`  The specified layer is locked by another application.
- `EINVAL`  No such layer, or any other error.
- `ENXIO`  The layer doesn’t exist.
- `EOPNOTSUPP`  The operation isn’t supported.
**PgUnlockLayer()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdSetTargetDevice(*), *PgLockLayer(*), *PgSetLayerArg(*),
*PgSetLayerSurface(*)

“Layers” in the Raw Drawing and Animation chapter of the Photon
*Programmer’s Guide*
**PgVideoChannel_t**

*Data structure that describes a video overlay channel*

**Synopsis:**

```
typedef struct pg_scaler_channel {
    PdOffscreenContext_t    *yplane1;
    PdOffscreenContext_t    *uplane1;
    PdOffscreenContext_t    *vplane1;
    PdOffscreenContext_t    *yplane2;
    PdOffscreenContext_t    *uplane2;
    PdOffscreenContext_t    *vplane2;
    unsigned               flags1;
    int                    chid1;
} PgVideoChannel_t;
```

**Description:**

This data structure describes a video overlay channel. It includes at least:

- **yplane1**  
  A pointer to an offscreen context describing the primary video buffer, if the format is not a Planar YUV format, or the primary Y data buffer for a Planar YUV format.

- **uplane1**  
  A pointer to an offscreen context describing the primary U data buffer for a Planar YUV format. It’s used only for Planar YUV formats.

- **vplane1**  
  A pointer to an offscreen context describing the primary V data buffer for a Planar YUV format. It’s used only for Planar YUV formats.

- **yplane2**  
  A pointer to an offscreen context describing the secondary video buffer, if the format is not a Planar YUV format, or the secondary Y data buffer for a Planar YUV format.

- **uplane2**  
  A pointer to an offscreen context describing the secondary the secondary U data buffer for a Planar YUV format. It’s used only for Planar YUV formats.

- **vplane2**  
  A pointer to an offscreen context describing the secondary the secondary V data buffer for a Planar YUV format. It’s used only for Planar YUV formats.
**PgVideoChannel_t**

*flags*  
No flags are currently defined.

*chid*  
Not implemented.

**Classification:**  
Photon

**See also:**  
*PdOffscreenContext_t, PgConfigScalerChannel(), PgCreateVideoChannel(), PgDestroyVideoChannel(), PgGetOverlayChromaColor(), PgGetScalerCapabilities(), PgNextVideoFrame(), PgScalerCaps_t, PgScalerProps_t*

“Video overlay” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
void PgWaitDrawComplete( void );
```

Library:

```
ph
```

Description:

This function waits until all previously emitted draw streams have been processed by the primary graphics driver.

This function is useful for throttling applications that continually draw. This call prevents the applications from getting ahead of the hardware. To wait until it’s safe to reuse a shared memory image, call `PgWaitHWIdle()`.

You must target this function at a specific card by calling `PdSetTargetDevice()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PdSetTargetDevice()`, `PgWaitHWIdle()`, `PgWaitVSync()`
**PgWaitHWIdle()**

*Wait until the video driver is idle*

**Synopsis:**

```c
int PgWaitHWIdle( void );
```

**Library:**

`ph`

**Description:**

This function waits until the video card’s FIFOs are empty and the engine is idle.

You must target this function at a specific card by calling `PdSetTargetDevice()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PdSetTargetDevice()`, `PgWaitDrawComplete()`, `PgWaitVSync()`

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
Synopsis:

void PgWaitVSync( void );

void PgWaitVSyncCx( void *dc );

Arguments:

dc PgWaitVSyncCx() only. A void pointer to any type of draw context. Examples of draw contexts are:

- a PhDrawContext_t returned by PhDCCreate()
- a PmMemoryContext_t returned by PmMemCreateMC()
- a PpPrintContext_t returned by PpCreatePC()
- a PdOffscreenContext_t returned by PdCreateOffscreenContext()

Library:

ph

Description:

These functions insert a “wait for vertical sync” tag into the drawstream. The driver waits until a vertical refresh has started before continuing to render the draw stream.

These functions is available only in direct mode.

PgWaitVSync() works on the current draw context, while you can specify the draw context for PgWaitVSyncCx().

Examples:

PgSetFillColor(Pg_RED);
PgWaitVSync();
PgDrawIRect(0,0,99,99,Pg_DRAW_FILL);
PgSetFillColor(Pg_BLACK);
PgDrawIRect(9,9,89,89,Pg_DRAW_FILL);
PgFlush(); // Wait for Vsync, then draw 2 rects
**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PgWaitDrawComplete(), PgWaitHWIdle()*

“Direct mode” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
Chapter 9
Ph—Photon
These functions handle operations that directly involve the Photon Manager. Using these functions, you can:

- Open Photon channels.
- Create, destroy, or modify regions that are independent of the widget hierarchy.
- Query the Photon Manager for information about regions.
- Initiate drag operations.
- Collect or emit events.
**PhAddMergeTiles()**

*Merge two list tiles, eliminating overlap*

**Synopsis:**

```c
PhTile_t * PhAddMergeTiles( PhTile_t *tiles,
                         PhTile_t *add_tiles,
                         int *added);
```

**Arguments:**

- `tiles` A pointer to a list of tiles.
- `add_tiles` A pointer to a list of tiles that you want to merge into the `tiles` list.
- `added` NULL, or a pointer to a location that the function sets to:
  - 0 if `tiles` or `add_tiles` is NULL, or `tiles` completely covers `add_tiles`
  - 1 otherwise.

**Library:**

`ph`

**Description:**

`PhAddMergeTiles()` merges the list of tiles pointed to by `add_tiles` into the list pointed to by `tiles` and returns a pointer to the resulting list.

This function makes sure that the tiles in the merged list don’t overlap.

**Returns:**

A pointer to the merged list. This isn’t always the same as the `tiles` pointer.

---

Don’t `free()` the list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.
**PhAddMergeTiles()**

**Classification:**
 Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhClipTilings(), PhCoalesceTiles(), PhCopyTiles(), PhDeTranslateTiles(), PhFreeTiles(), PhGetTile(), PhIntersectTilings(), PhMergeTiles(), PhRectsToTiles(), PhSortTiles(), PhTile_t, PhTilesToRects(), PhTranslateTiles()*

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PhAllocPackType()**

Allocate a buffer and pack transport data into it

### Synopsis:

```c
char * PhAllocPackType( char unsigned *type,
                        char unsigned *desc,
                        int unsigned grouping_num,
                        int unsigned handle,
                        int unsigned transport,
                        char *packing_type,
                        char unsigned *data,
                        int unsigned len,
                        int *size );
```

### Arguments:

- **type**
  
  A descriptive type name, such as `image`, `text`, `filename`, or `files`. This is simply added to the header for the packed data.

- **desc**

  The specifics of what’s in the data. The extractor uses a regular-expression match against the description to determine if the data should be unpacked or discarded. This is simply added to the header for the packed data.

- **grouping_num**

  When used with Photon’s drag and drop mechanism, the `grouping_num` is used to indicate which stream is just a different representation of other data also packed into the same `PhTransportCtrl_t` structure. Only one of each `grouping_num` should be unpacked by the reader/destination.

  This value is simply added to the header for the packed data.

- **handle**

  A number that you can use to identify a transaction. This is simply added to the header for the packed data.

- **transport**

  The transport type used for the inlined data. This can be one of:
**PhAllocPackType()**

- **Ph_TRANSPORT_INLINE** — the data being transported is in memory and can be unpacked immediately.
- **Ph_TRANSPORT_FILEREF** — the data being transported is in the temporary file(s) named in the inlined data.
- **Ph_TRANSPORT_SHMEM** — the data being transported is in the temporary shared object(s) named in the inlined data.

**packing_type**  
The name of the entry in the transport registry to be used to pack the data. For more information, see *PhTransportRegEntry_t*. If you already have a pointer to the registry entry, you can call *PhPackEntry()* instead of *PhPackType()*.

**data**  
A pointer to the data to be packed.

**len**  
The size, in bytes, of the data to be packed. This size is used only for raw data.

**size**  
If this argument isn’t NULL, the size of the allocated buffer is stored in the memory it points to.

**Library:**

ph

**Description:**

This function allocates a buffer big enough to hold the packed version of the given data, and then packs the data into the buffer.

**Returns:**

A pointer to the buffer, or NULL if no data was packed.
**PhAllocPackType()**

**Classification:**

Ph Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhMallocUnpack(), PhPackEntry(), PhPackType(),
PhTransportCtrl_t, PhTransportRegEntry_t,
PhTransportType(), PhUnpack()

Drag and Drop chapter of the Photon *Programmer’s Guide*
PhArea_t
Position and dimensions of a rectangular area

Synopsis:

typedef struct Ph_area {
    PhPoint_t pos;
    PhDim_t size;
} PhArea_t;

Description:

The PhArea_t structure describes the position and dimensions of a rectangular area. It’s used extensively by the widget (Pt*) functions (see the Photon Widget Reference). This structure contains at least the following members:

- **pos**  Upper-left corner of the area.
- **size**  Size of the area.

Classification:

Ph

See also:

- PhDim_t, PhPoint_t, PhRect_t

“Geometry data types” in the Working with Code chapter of the Photon Programmer’s Guide
**PhAreaToRect()**

_Convert an area into a rectangle_

**Synopsis:**

```c
void PhAreaToRect( PhArea_t const *area,
                   PhRect_t *rect );
```

**Library:**

`ph`

**Description:**

This function converts an area (i.e. a position and dimensions) into a rectangle (i.e. upper-left and lower-right points).

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhArea_t, PhRect_t, PhRectToArea()`
PhAttach()
Open a communications channel

Synopsis:

```c
struct _Ph_ctrl *
PhAttach(
    char const *name,
    PhChannelParms_t const *parms);
```

Library:

```
ph
```

Description:

This function opens a communications channel to a Photon Manager. The channel becomes the current channel.

This is a low-level routine that you aren’t likely to call directly. Both PtInit() and PtAppInit() invoke this function. Your application must call one of these functions or PhAttach() before it calls any other Photon functions.

A Photon channel contains:

- an FD that can be used to send QNX messages to Photon
- an optional channel that the Photon Manager can send Neutrino pulses to
- a draw buffer for graphics commands.

PhAttach() doesn’t create a channel; if you need to create one, call PhChannelAttach().

The name argument contains the name registered by a Photon Manager. If you pass NULL, the function uses the PHOTON environment variable. If PHOTON isn’t set, the function uses /dev/photon instead.

The parms argument lets you fine-tune the resources of the channel. Passing NULL to this argument gets the channel defaults:

- maximum queue size (max_q_entries) of 10 events
### PhAttach()

- no special flags on the channel.

If you don’t pass NULL for *parms*, you should pass a pointer to a PhChannelParms_t structure, which contains at least:

```c
unsigned long max_q_entries;
unsigned long flags;
```

where:

- `max_q_entries`
  - The maximum number of queued events likely to be needed.
  - The Photon Manager may override this value.

- `flags`
  - Defined flags:
    - **Ph_NO_HOLD**  Don’t block the client if it overflows another application’s event queue.
    - **Ph_DYNAMIC_BUFFER**
      - If there’s a pending Photon event that’s larger than the client’s event buffer, send an event that indicates how large the client’s buffer needs to be to receive the entire event message. For more information, see *PhEventNext()*, *PhEventRead()*, and *PhGetMsgSize()*.

If you attach communications channels to multiple Photon managers, you’ll have to keep track of which regions belong to which manager.

## Returns:

A pointer to a control structure.
Examples:

```c
promiscuous_call( void )
{
    struct _ph_ctrl *ph1, *ph2, *ph3;

    ph1 = PhAttach( NULL, NULL );
    if( ph1 )
        printf( "ph1 is the current channel to: "
            "the local Photon kernel\n" );
    ph2 = PhAttach( "/dev/photon", NULL );
    if( ph2 )
        printf( "ph2 is the current channel to: "
            "the local Photon kernel\n" );
    ph3 = PhAttach( "/net/darrin/dev/photon", NULL );
    if( ph3 )
        printf( "ph3 is the current channel to: "
            "the Photon kernel on node 83\n" );
    if( !ph1 | !ph2 | !ph3 )
        return( -1 );

    PhReattach( ph1 );
    printf( "ph1 is the current channel again\n" );
    PhDetach( ph1 );
    printf( "there is no current channel\n" );
    PhReattach( ph3 );
    printf( "ph3 is the current channel again\n" );
    PhDetach( ph2 );
    PhDetach( ph3 );
    printf( "all Photon channels closed\n" );
    return( 0 );
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
*PhAttach()*

See also:

`PgSetDrawBufferSize()`, `PhChannelAttach()`, `PhDetach()`, `PhEventNext()`, `PhEventArm()`, `PhEventRead()`, `PhGetMsgSize()`, `PhReattach()`, `PtInit()`, `PtAppInit()`
PhBlit()

Blit an area within a region

Synopsis:

```c
int PhBlit( PhRid_t rid,
            const PhRect_t *rect,
            const PhPoint_t *offset );
```

Library:

ph

Description:

This function “blits” the area that is defined by the `PhRect_t` structure pointed to by `rect` and whose origin is defined by the origin of the region specified by the `PhPoint_t` structure pointed to by `rid`. The area is blitted by the given `offset`. Other windows aren’t affected by the blit.

Returns:

A nonnegative value

Success.

-1 The blit failed, possibly because `rid` was incorrect or the Photon Manager wasn’t running.

Examples:

```c
PhRect_t rect = { 10,10,20,20};
PhPoint_t offset = { -5, 5};
PhRect_t exposed = { 15, 10, 20, 15};

// Blit the area bounded by (10,10), (20,20)
// five pixels left and five pixels down.
PhBlit( PtWidgetRid( region_widget ), &rect, &offset );
PtDamageExtent( region_widget, &exposed );
```
**PhBlit()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhPoint_t, PhRect_t, PgBlit*, PtClippedBlit, PtWidgetRid()
PhBitmapCursorDescription_t

Bitmap for the cursor

Synopsis:

typedef struct Ph_bitmap_cursor descr {
    PhCursorDescription_t hdr;
    PhBitmapCursorData_t bmp;
} PhBitmapCursorDescription_t;

Description:

The PhBitmapCursorDescription_t structure defines a bitmap cursor. The members include at least:

The PhBitmapCursorDescription_t contains these members:

hdr

The structure header. This is a PhCursorDescription_t structure that is automatically filled in by the widget. You pass this instead of the PhBitmapCursorDescription_t in functions that have a cursor argument, such as PhInitDrag().

The hdr has these members:

- hdr.type — must be Ph_CURSOR_BITMAP.
- hdr.length — the size of the PhCharacterCursorDescription_t structure. For example, if the structure is called curdef, hdr.length must be equal to:

  curdef->bmp.images - (char*)curdef +
  curdef->bmp.bytesperline1 * curdef->bmp.size1.h +
  curdef->bmp.bytesperline2 * curdef->bmp.size2.h

bmp

A PhBitmapCursorData_t structure that describes the bitmap (see below).

The PhBitmapCursorData_t structure defines the bitmap used as a cursor defined by PhBitmapCursorDescription_t.

typedef struct Ph_bitmap_cursor_data {
    PhPoint_t size1;
    ...
PhBitmapCursorDescription_t

PhPoint_t offset1;
PgColor_t color1;
char bytesperline1;
PhPoint_t size2;
PhPoint_t offset2;
PgColor_t color2;
char bytesperline2;
char Spare[14];
char images[];

} PhBitmapCursorData_t;

The members are:

size1 The dimensions of the first bitmap plane, in pixels.
offset1 The position of the upper-left corner of the first plane of the bitmap, relative to the hot spot.
color1 The color of the first bitmap plane.
bytesperline1 The number of bytes per line for the first bitmap plane.

size2 The dimensions of the second bitmap plane, in pixels. If there’s only one bitplane, set this to 0.
offset2 The position of the upper-left corner of the second plane of the bitmap, relative to the hot spot.
color2 The color of the second bitplane.
bytesperline2 The number of bytes per line for the second bitmap plane. If there’s only one bitplane, set this to 0.
images The bitmap image data, as a series of 1-bit-per-pixel planes. Typically, you need to allocate an appropriate amount of memory using malloc(), and then use memcpy() to copy the bitmaps into the memory starting at bmp.images[0].
Many graphics drivers don’t support alpha in the cursor colors.

**Classification:**

Photon

**See also:**

PhCursorDescription_t
PhCancelDrag()
Cancel a drag operation

Synopsis:

```c
int PhCancelDrag(
    PhRid_t rid,
    unsigned input_group,
    unsigned flags);
```

Arguments:

- `rid`: A `PhRid_t` that must match the `rid` passed to the `PhInitDrag()` that initiated the drag operation you want to cancel.
- `flags`: Must be 0.
- `input_group`: An input-group value that must match the `input_group` passed to the `PhInitDrag()` that initiated the drag operation you want to cancel.

Library:

ph

Description:

This function cancels a drag. The application still collects a `Ph_EV_DRAG` event with a subtype of `PhEV_DRAG_COMPLETE` that describes the results of the operation. You can inspect the `button_state` member of the event `PhDragEvent_t` data structure to determine whether the `PhCancelDrag()` call or a button release caused the operation to complete.

Returns:

- `>0`: Successful completion.
- `-1`: An error occurred.
**PhCancelDrag()**

**Classification:**

<table>
<thead>
<tr>
<th>Photon</th>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt handler</td>
</tr>
<tr>
<td></td>
<td>Signal handler</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

- `PhBitmapCursorDescription_t`
- `PhCharacterCursorDescription_t`
- `PhDragEvent_t`
- `PhEvent_t`
- `PhGetData()`
- `PhDim_t`
- `PhInitDrag()`
- `PhPoint_t`
- `PhRect_t`
- `PhTranslateRect()`

“Dragging” in the Events chapter of the Photon *Programmer’s Guide*
**PhChannelAttach()**

Create or use a Neutrino channel

**Synopsis:**

```c
int PhChannelAttach( int channel,
                     int connection,
                     struct sigevent const *event);
```

**Arguments:**

- `channel`: A channel ID, or 0 to create a new channel.
- `connection`: A connection ID, or -1 to create a new connection.
- `event`: argument describes how Photon is to notify your application. If your application is using the widget library, pass NULL. For more information, see `sigevent` and `ionotify()` in the QNX Neutrino Library Reference.

**Library:**

`ph`

**Description:**

Use this function if you want the library to create a Neutrino channel or use one that you’ve already created.

**name_attach() and PtAppAddInput()**

`PtAppAddInput()` and `name_attach()` both try to create a channel with `_NTO_CHF_COID_DISCONNECT` and `_NTO_CHF_DISCONNECT` set (see the QNX Neutrino Library Reference). If your application calls both functions, you need to let Photon use the same channel as `name_attach()`. To do this, call these functions in this order:

- `name_attach()`
- `PhChannelAttach()`
- `PtAppAddInput()`
**PhChannelAttach()**

See the Examples section for a sample of code that illustrates the correct order.

If you want to create a separate channel for Photon, it doesn’t matter whether you create it and give it to `PhChannelAttach()` before or after calling `name_attach()`. But keep in mind that since certain mechanisms in Photon library expect the Photon channel to have the two DISCONNECT flags, they might not work properly if it doesn’t. One such mechanism is the detection of broken connections (see `PtConnectionClientSetError()` and `PtConnectionServerSetError()`) and anything that relies on it.

**Returns:**

A channel ID, or -1 on error (`errno` is set).

**Errors:**

- **EBUSY**: A channel is already attached and `chid` is nonzero and differs from the current channel ID, or `connection` isn’t -1 and differs from the currently used connection.
- **EINVAL**: The `channel` argument is 0, but `connection` isn’t -1.
- Other values: `ChannelCreate()` or `ConnectAttach()` failed.

**Examples:**

To create a channel and a connection:

```
PhChannelAttach( 0, -1, NULL )
```

To attach a channel `chid` and create a connection:

```
PhChannelAttach( chid, -1, NULL )
```

To attach channel `chid` and connection `coid`:

```
PhChannelAttach( chid, coid, NULL )
```

Here’s a fully working code sample that illustrates the order of `PhChannelAttach()`, `name_attach()`, and `PtAppAddInput()`:
PhChannelAttach()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <sys/iomsg.h>
#include <sys/iofunc.h>
#include <sys/dispatch.h>
#include <Pt.h>

struct my_msg
{
    short type;
    char reply[50];
};

#define NON_PHOTON_PULSE _IO_MAX+4
#define MY_SERV "my_server_name"

int non_photon_msg_func (void *data, int rcvid, void *message, size_t size);

int main( int argc, char **argv)
{
    name_attach_t *attach;
    PtWidget_t *window;

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    attach = name_attach( NULL, MY_SERV, 0);
    if (attachment == NULL)
    {
        printf("server:failed to attach name, errno %d\n", errno);
        PtExit(EXIT_FAILURE);
    }

    PhChannelAttach (attach->chid, -1, NULL);

    PtAppAddInput( NULL, 0, &non_photon_msg_func, NULL);

    window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    PtRealizeWidget ( window);

    PtMainLoop ();
    return 0;
}

int non_photon_msg_func (void *data, int rcvid, void *message, size_t size)
{
    struct my_msg *msg = ( struct my_msg * ) message;
    printf ( "Recieved a non photon message\n");

    if ( msg->type == NON_PHOTON_PULSE )
    {
        printf("server: This message is to be handled by this input handler\n");
        strcpy ( msg->reply, "I got your message" );
        MsgReply ( rcvid, OK, (char *) msg->reply, sizeof ( msg->reply ));
        return ( Pt_HALT );
    }
    else
    {
        printf("server: This message isn't for this input handler\n");
    }
```

812  Chapter 9  Ph—Photon
PhChannelAttach()

```c
return ( Pt_CONTINUE );
}
}
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtAppAddInput()*

Interprocess Communication chapter of the Photon *Programmer’s Guide*

*ionotify(), name_attach(), sigevent* in the QNX Neutrino *Library Reference*
PhCharacterCursorDescription_t
A character cursor

Synopsis:

typedef struct Ph_character_cursor_data {
    PhCursorDescription_t hdr;
    PgColor_t color;
} PhCharacterCursorDescription_t;

Description:

The PhCharacterCursorDescription_t structure is used to define a character cursor. It contains these members:

hdr
The structure header. This is a PhCursorDescription_t structure. You pass this instead of the PhBitmapCursorDescription_t in functions that have a cursor argument, such as PhInitDrag().

The hdr has these members:
- hdr.type — one of the cursor types listed below.
- hdr.length — must be equal to sizeof(PhCharacterCursorDescription_t).

color
A PgColor_t structure that describes the cursor color.

Cursor types:
- Ph_CURSOR_INHERIT — The cursor type is inherited from the parent region.
- Ph_CURSOR_POINTER —
- Ph_CURSOR_BIG_POINTER —
- Ph_CURSOR_MOVE —
- Ph_CURSOR_CROSSHAIR —
PhCharacterCursorDescription_t

- Ph_CURSOR_CLOCK, Ph_CURSOR_WAIT — 🕒
- Ph_CURSOR_NOINPUT, Ph_CURSOR_DONT — ✗
- Ph_CURSOR_FINGER — 🧠
- Ph_CURSOR_INSERT — 🟠
- Ph_CURSOR_DRAG_VERTICAL, Ph_CURSOR_DRAG_TOP, Ph_CURSOR_DRAG_BOTTOM — 📣
- Ph_CURSOR_DRAG_HORIZONTAL, Ph_CURSOR_DRAG_LEFT, Ph_CURSOR_DRAG_RIGHT — ➡️
- Ph_CURSOR_DRAG_BACKDIAG, Ph_CURSOR_DRAG_TL, Ph_CURSOR_DRAG_BR — 📌
- Ph_CURSOR_DRAG FOREDIAG, Ph_CURSOR_DRAG_TR, Ph_CURSOR_DRAG_BL — 💼
- Ph_CURSOR_LONG_WAIT — 🕒
- Ph_CURSOR_QUESTION_POINT — 🎈
- Ph_CURSOR_PASTE — 🎈

Classification:

Photon

See also:

PhCursorDescription_t, PgColor_t
PhClipboardCopyString()

Copy string-only data to the clipboard

Synopsis:

```c
int PhClipboardCopyString( unsigned short ig, 
                           const char *string );
```

Library:

ph

Description:

This function is a simple cover function for copying string-only data to the clipboard. It builds a PhClipboardHdr entry:

```c
{ "TEXT", strlen(string), string }
```

and then calls PhClipboardWrite() to perform the operation. The string must be NULL terminated.

Each input group has its own private clipboard, which can be selected through the `ig` parameter. To determine the current input group, call PhInputGroup(), passing to it the event that triggered the clipboard operation (e.g. `cbinfo->event`).

Returns:

- 0     Successful completion.
- -1    An error occurred.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

816  Chapter 9  Ph—Photon
PhClipboardCopyString()

See also:

PhClipboardHdr, PhClipboardPasteString(), PhClipboardRead(), PhClipboardWrite()
PhClipboardHdr
Clipboard header structure

Synopsis:

typedef char PhClipType[8];

typedef struct {
    PhClipType    type;
    uint32_t      length;
    void          *data;
} PhClipboardHdr;

#define Ph_CLIPBOARD_TYPE_TEXT "TEXT"

Description:

This data structure describes clipboard data. Its members include:

type The type of data — an arbitrary 8-character string (e.g. TEXT or BMAP) that you can define for your application.

length The length of the data (pointed to by data).

data A pointer to the data itself (of length bytes).

Classification:

Photon

See also:

PhClipboardCopyString(), PhClipboardPasteString(), PhClipboardRead(), PhClipboardWrite()
PhClipboardPasteString()
Paste string-only data from the clipboard

Synopsis:

char *PhClipboardPasteString( unsigned short ig );

Library:

ph

Description:

This function is a simple cover function for pasting string-only data from the clipboard. The function calls PhClipboardRead(), and requests data of type TEXT.

Each input group has its own private clipboard, which can be selected through the ig parameter. To determine the current input group, call PhInputGroup(), passing to it the event that triggered the clipboard operation (e.g. cbinfo->event).

This function allocates the resultant string with strdup(). Your application must free() this memory after use.

Returns:

A pointer to the text string extracted from the clipboard, or NULL if there was no available data or an error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PhClipboardCopyString(), PhClipboardHdr,
PhClipboardPasteString(), PhClipboardRead,
PhClipboardWrite

strdup(), free() in the QNX Neutrino Library Reference
PhClipboardRead()
Copy data from the clipboard

Synopsis:

typedef char PhClipType[8];

PhClipboardHdr *PhClipboardRead(unsigned short ig,
                                   PhClipType type);

Arguments:

ig The input group. Each input group has its own private clipboard. To determine the current input group, call PhInputGroup(), passing to it the event that triggered the clipboard operation (e.g., cbinfo->event).

type The data type to read from the clipboard.

Library:

ph

Description:

This function copies the clipboard data in that matches type from the Photon clipboard, and returns a pointer to the data populated in a PhClipboardWrite structure.

Returns:

A pointer to a populated PhClipboardWrite
   Successful completion.
NULL An error occurred.

Examples:

This callback reads data from the clipboard, and pastes it in a PtText widget named text:

```c
int paste_from_clip(PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
```
PhClipboardRead()

```c
int *cursor, ig, insertion_pt;
PhClipboardHdr *ptr;

ig=PhInputGroup(cbinfo->event);
PtGetResource(ABW_text, Pt_ARG_CURSOR_POSITION, &cursor, 0);
insertion_pt=*cursor;

if (ptr=PhClipboardRead(ig, Ph_CLIPBOARD_TYPE_TEXT))
{
    PtTextModifyText(ABW_text, insertion_pt, insertion_pt,
                     insertion_pt, ptr->data, utf8strlen(ptr->data, ptr->length, NULL) );
    free(ptr->data);
    free(ptr);
}

PtContainerGiveFocus(ABW_text, NULL);
return(Pt_CONTINUE);
```

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhClipboardCopyString(), PhClipboardHdr, PhClipboardPasteString(), PhClipboardWrite*
**PhClipboardWrite()**

Copy data to the clipboard

**Synopsis:**

```c
int32_t PhClipboardWrite(unsigned short ig,
                          uint32_t n,
                          PhClipboardHdr const clip[]);
```

**Arguments:**

- **ig**
  The input group. Each input group has its own private clipboard. To determine the current input group, call `PhInputGroup()`, passing to it the event that triggered the clipboard operation (e.g. `cbinfo->event`).

- **n**
  The number of items in the `clip` array.

- **clip**
  An array of `PhClipboardHdr` structures, that specify the information you want to save to the clipboard. Each entry includes the data, its type, and its length.

**Library:**

`ph`

**Description:**

This function copies the clipboard data in `clip` to the Photon clipboard. Each clip is saved based on the input group `ig`, data type `clip.type`, and ID of the user. This ensures that one user can’t access clipboard data saved by another user. Clipboard data is also encrypted, and can only be accessed through the clipboard API function `PhClipboardRead()`.

Multiple representations of the data may be placed on the clipboard. For example, you may want to save text and format data for the text. The number of different types is specified with the `n` parameter. Each type has a header structure in the `clip` array. For more information, see `PhClipboardHdr`. 
**PhClipboardWrite()**

**Returns:**

0   Successful completion.

-1   An error occurred.

**Examples:**

This callback copies selected text from a PtText widget named text, and saves it on the clipboard:

```c
int copy_to_clip( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
    char *selstring;
    int start, end, len, ig;
    PhClipboardHdr clip[1];

    ig=PhInputGroup(cbinfo->event);
    len = PtTextGetSelection(ABW_text, &start, &end);
    if(start!=-1 && len > 1) {
        char *text = NULL;
        PtGetResource( ABW_text, Pt_ARG_TEXT_STRING, &text, 0 );
        if( text ) {
            int s = utf8strnlen( text, start, NULL ),
                l = utf8strnlen( text + s, end - start, NULL );
            if( NULL != (selstring = malloc( l + 1 )) ) {
                // Copy text to clipboard
                memcpy( selstring, text + s, l );
                strcpy(clip[0].type,Ph_CLIPBOARD_TYPE_TEXT);
                clip[0].length=strlen(selstring);
                clip[0].data = selstring;
                PhClipboardWrite(ig, 1, clip);
                free(selstring); selstring=NULL;
            }
        }
    }
    PtContainerGiveFocus(ABW_text, NULL);
    return(Pt_CONTINUE);
}
```

**Classification:**

Photon
PhClipboardWrite()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhClipboardCopyString(), PhClipboardHdr,
PhClipboardPasteString(), PhClipboardRead(), PhClipboardHdr
**PhClipTilings()**

Clip one list of tiles from another

**Synopsis:**

```c
PhTile_t *PhClipTilings(
    PhTile_t *tiles,
    PhTile_t const * const clip_tiles,
    PhTile_t **intersection );
```

**Library:**

ph

**Description:**

This function clips the list of tiles pointed to by `clip_tiles` from the list pointed to by `tiles`. If `intersection` isn’t NULL, it’s set to point to the list of intersections that are clipped out of the `tiles` list.

The `clip_tiles` list isn’t modified.

---

Don’t `free()` a list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.

**Returns:**

A pointer to the clipped list of tiles, or NULL if `clip_tiles` encompasses `tiles`.

**Classification:**

Photon

---

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
See also:

PhAddMergeTiles(), PhCoalesceTiles(), PhCopyTiles(),
PhDeTranslateTiles(), PhFreeTiles(), PhGetTile(),
PhIntersectTilings(), PhMergeTiles(), PhRectsToTiles(),
PhSortTiles(), PhTile_t, PhTilesToRects(), PhTranslateTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of
the Photon Programmer’s Guide
**PhCoalesceTiles()**

*Combine a list of tiles*

**Synopsis:**

```c
PhTile_t * PhCoalesceTiles( PhTile_t *tiles );
```

**Library:**

```
ph
```

**Description:**

`PhCoalesceTiles()` combines the tiles in the list pointed to by `tiles` as much as possible. This function works best on a sorted, merged list of tiles.

```
Don’t free() the list of tiles; instead, use PhFreeTiles() to return the tiles to the internal pool.
```

**Returns:**

A pointer to the list, which is always the same as the pointer given.

**Classification:**

```
Photon
```

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

```
PhAddMergeTiles(), PhClipTilings(), PhCopyTiles(), PhDeTranslateTiles(), PhFreeTiles(), PhGetTile(), PhIntersectTilings(), PhMergeTiles(), PhRectsToTiles(), PhSortTiles(), PhTile_t, PhTilesToRects(), PhTranslateTiles()
```
Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PhCopyTiles()**

*Copy a list of tiles*

**Synopsis:**

```c
PhTile_t * PhCopyTiles(
    PhTile_t const * const tile);
```

**Library:**

ph

**Description:**

This function creates a copy of the list of tiles pointed to by `tile`. Don’t `free()` the list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.

**Returns:**

A pointer to the copy.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`, `PhGetTile()`, `PhIntersectTilings()`, `PhMergeTiles()`, `PhRectsToTiles()`, `PhSortTiles()`, `PhTile_t`, `PhTilesToRects()`, `PhTranslateTiles()`
“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhCreateImage()
Create a new PhImage_t structure

Synopsis:

PhImage_t *PhCreateImage( PhImage_t *buffer,
                           short width,
                           short height,
                           int type,
                           PgColor_t const *palette,
                           int ncolors,
                           int shmem );

Library:

   ph

Description:

This function creates a new Photon image and allocates space for the image data and palette (if present).

The buffer argument lets you pass in a pointer to a PhImage_t structure to fill in. If this value is NULL, the function allocates a structure for you.

The width and height specify the size of the new image, in pixels. The type specifies the type of the image. Supported types are outlined in the documentation for PhImage_t.

The palette and ncolors arguments let you have this function automatically allocate and fill in space to store the image’s palette, if needed. The ncolors argument specifies the size of the palette in terms of the number of colors, while palette points to the list of colors specifying the palette itself:

- If ncolors is 0, no palette is allocated or copied.
- If ncolors is nonzero, but palette is NULL, space is allocated for the palette, but nothing is copied.
- If ncolors is nonzero and palette is non-NULL, then space is allocated and the palette is copied automatically.

The shmem argument specifies whether or not shared memory should be allocated for the image’s data. For large images, shared memory
PhCreateImage()

facilitates faster data transfer to local graphics drivers, and hence quicker rendering of the image. If you wish to use shared memory to store the image data, set this value to 1. Otherwise, set it to 0.

Returns:

A pointer to the new image on success, or NULL if an error occurred due to lack of memory, or if the specified type isn’t a recognized Photon image type.

Classification:

Ph

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApGetImageRes(), PgColor_t, PgDrawPhImage*(), PgDrawPhImageRect*(), PgDrawRepPhImage*(), PhImage_t, PhMakeGhostBitmap(), PhMakeTransBitmap(), PhMakeTransparent(), PhReleaseImage(), PmMemCreateMC(), PmMemFlush(), PxLoadImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PhCreateTransportCtrl()**

*Allocate a PhTransportCtrl_t structure*

**Synopsis:**

```c
PhTransportCtrl_t *PhCreateTransportCtrl( );
```

**Library:**

`ph`

**Description:**

This function creates and initializes a control structure to be used when packing data to send using Photon’s transport mechanism.

**Returns:**

A pointer to the `PhTransportCtrl_t` structure created.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhGetNextInlineData()`, `PhGetTransportVectors()`, `PhPackEntry()`, `PhPackType()`, `PhReleaseTransportCtrl()`, `PhTransportCtrl_t`, `PhTransportLink_t`, `PhTransportType()`, `PtCreateTransportCtrl()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
PhCursorDef_t

Bitmap for the cursor

Synopsis:

typedef struct Ph_cursor_def {
    PhRegionDataHdr_t    hdr;
    PhPoint_t            size1;
    PhPoint_t            offset1;
    PgColor_t            color1;
    char                  bytesperline1;
    PhPoint_t            size2;
    PhPoint_t            offset2;
    PgColor_t            color2;
    char                  bytesperline2;
    char                  Spare[14];
    char                  images[1];
} PhCursorDef_t;

Description:

The PhCursorDef_t structure is used to define bitmaps to be used as the cursor. The members include at least:

- **hdr**: A pointer to a PhRegionDataHdr_t structure that defines the region data header.
- **size1**: The dimensions of the first bitmap plane, in pixels.
- **offset1**: The position of the upper-left corner of the first plane of the bitmap, relative to the hot spot.
- **color1**: The color of the first bitmap plane.
- **bytesperline1**: The number of bytes per line for the first bitmap plane.
- **size2**: The dimensions of the second bitmap plane, in pixels.
- **offset2**: The position of the upper-left corner of the second plane of the bitmap, relative to the hot spot.
- **color2**: The color of the second bitplane. You can’t have more than two bitplanes.
**PhCursorDef_t**

- **bytesperline2**  The number of bytes per line for the second bitmap plane.
- **images**  The bitmap image data, as a series of 1-bit-per-pixel planes.

Most graphics drivers don’t support alpha in the cursor colors.

**Classification:**
Photon

**See also:**

- `PgColor_t`, `PhPoint_t`, `PhRegionDataHdr_t`
- `Pt_ARG_BITMAP_CURSOR(PtWidget)` in the *Widget Reference*
PhCursorDescription_t
Cursor header structure

Synopsis:

typedef struct Ph_ev_cursor_descr {
    unsigned short type;
    unsigned short length;
} PhCursorDescription_t;

Description:

The PhCursorDescription_t structure is used as a header for these cursor description structures:

- PhCharacterCursorDescription_t — a character cursor structure
- PhBitmapCursorDescription_t — a bitmap cursor structure

In functions that call for a PhCursorDescription_t, you should pass in the address for the hdr structure of a populated PhCharacterCursorDescription_t or PhBitmapCursorDescription_t.

The members of PhCursorDescription_t are:

- type: Defines a cursor type. See the parent structure for a description.
- length: The total length of the parent cursor structure, in bytes.

Classification:

Photon

See also:

PhCharacterCursorDescription_t, PhBitmapCursorDescription_t, PhInitDrag(), PtDndSelect(), PtInitDnd()
PhDCCreate()
Create and initialize a new draw context

Synopsis:

PhDrawContext_t *PhDCCreate(
  int type,
  long flags,
  int (*flush)(int Subtype),
  int (*modify)(PhDrawContext_t *dc,
               int acquire,
               void *data ) );

Arguments:

**type**
The type of draw context:

- Ph_DRAW_TO_MEMORY_CONTEXT — the context being created will draw to memory.
- Ph_DRAW_TO_OFFSCREEN_MEMORY — emit the draw stream to offscreen memory.
- Ph_DRAW_TO_PHOTON — normal draw mode.
- Ph_DRAW_TO_PRINT_CONTEXT — create a context that’s compatible with the Pp* api and will embed the required printer control codes in the draw stream and allow you to direct the draw stream to an external source (spooler/file)
- Ph_DRAW_TO_SERVICE — emit the draw stream directly to a service provider.

**flags**
Flags that apply to the context:

- Ph_INLINE_SHMEM_OBJECTS — shared objects are inlined in the draw stream. This ensures the interpreter of the draw stream can access the data. It’s possible that the interpreter is on a different machine and doesn’t have access to the shared objects refered to or may be interpreting the draw stream weeks later, when the shared objects no longer exist.
- Ph_TEXT_EXTENTS — include bounding rectangles for drawn text. This extra information can be used by remote viewers (Phindows etc.) to find a font that matches the desired size as possible. Used internally.
PhDCCreate()

- **Ph_SUPPRESS_PARENT_CLIP** — when a widget draw cycle begins, the parents of the starting widget are traversed and their canvases are intersected to determine the clipping to be used during the draw. If this flag is set, that step is skipped. As a result, all of the starting widget is visible in the draw stream (as if it isn’t clipped). This is important (for example) for printing large widgets in small containers.

- **Ph_SYNC_GCS** — force a “sync GC” command to be placed at the beginning of each draw stream. This command causes the graphics driver to wait for a vertical retrace before processing the draw commands.

**flush**
The function to be called whenever the current draw buffer needs to be flushed. If not provided, the standard Photon graphic flush function is used.

The *Subtype* argument is what’s passed by *PgFFlush()*.

**modify**
A function that’s called whenever this draw control is to be modified. The modification is restricted by the type of the draw control (e.g. if the type is Ph_DRAW_TO_PRINTER, you can target a different printer but can’t target memory with subsequent calls to the *modify* function).

The arguments are:

- *dc* — a pointer to the draw context structure.

- *action* — what’s being done to the draw context; one of:
  - Ph_CREATED_DC
  - Ph_ACTIVATE_DC
  - Ph_DEACTIVATE_DC
  - Ph DESTROYING_DC

- *data* — a pointer to arbitrary data you want to pass to the function.
**PhDCCreate()**

**Library:**

**Description:**

This function creates a draw context structure and initializes its *flush* and *attach* functions. The *type* argument is recorded in the new draw control and prevents the DC from being used as anything but its current type.

You aren’t likely to call this function directly unless you’re creating your own type of draw context. The functions that create specific types of contexts (e.g. direct-mode, printer, and memory) call *PhDCCreate()*.

**Returns:**

A pointer to the newly allocated draw context, or NULL if there isn’t enough memory.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PdCreateDirectContext(), PdCreateOffscreenContext(),*  
*PhDCSetCurrent(), PhDCGetCurrent(), PhDCRelease(),*  
*PmMemCreateMC(), PpCreatePC()*
**Synopsis:**

\[
\text{PhDrawContext_t} \ast \text{PhDCGetCurrent ( void );}
\]

**Library:**

ph

**Description:**

This function returns a pointer to the currently active draw context, which may be a print context, memory context, or draw context.

**Returns:**

A pointer to the currently active draw context.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhDCCreate(), PhDCSetCurrent(), PhDCRelease()*
PhDCRelease()

Release a draw context

Synopsis:

```c
int PhDCRelease( PhDrawContext_t *dc );
```

Library:

```
ph
```

Description:

This function releases the provided DC (the default Photon draw context can not be released). The context is notified of its pending demise before it’s destroyed, so that it can do any necessary cleanup.

You aren’t likely to call this function directly unless you’re releasing:

- your own type of draw context
- an offscreen context, created by calling `PdCreateOffscreenContext()`. If you’ve locked the context, call `PdLockOffscreen()` to unlock it before releasing the offscreen context.

The functions that release specific types of contexts (e.g. direct-mode, printer, and memory) call `PhDCRelease()`.

Returns:

- **0**  Success.
- **-1** An error occurred (most likely you’re trying to release the Photon default DC).

Classification:

```
Photon
```
PhDCRelease()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PdReleaseDirectContext(), PdCreateOffscreenContext(),
PhDCCreate(), PhDCGetCurrent(), PhDCSetCurrent(),
PmMemReleaseMC(), PpReleasePC()

“Video memory offscreen” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PhDCSetCurrent()**

Set the currently active draw context

**Synopsis:**

```c
PhDrawContext_t *PhDCSetCurrent(
    void *draw_context);
```

**Arguments:**

`draw_context`  A pointer to the draw context to be made the default one.

**Library:**

`ph`

**Description:**

This function makes the provided `draw_context` active. Calling this function with NULL makes the default draw context active. The default draw context emits draws to graphics drivers via Photon.

A draw context is anything that defines the flow of the draw stream. Print contexts and memory contexts are types of draw contexts — it may help to think of them as specialized subclasses of the draw context.

Contexts that may be set using this function:

**Draw contexts**

There’s usually only one basic draw context per application. Draw contexts are used to deliver the draw stream to graphics drivers via Photon.

**Print contexts**

Created via `PpCreatePC()`. Print contexts are used to produce printed output from Photon applications.

**Memory contexts**

Created via `PmMemCreateMC()`. Memory contexts are used to draw into memory to build images for manipulation or display.
PhDCSetCurrent()

Returns:

The old draw context, or NULL if the new context can’t be made current (active), in which case errno has specifics of the error.

Examples:

In the following example, the print context *pc is made active by calling PpContinueJob(). PpContinueJob() returns the context that the print context is replacing. The returned context is stored to enable us to restore the context that was active at the time we decided to start printing.

```c
PhDrawContext_t *dc;
PpPrintContext_t *pc;
FmMemoryContext_t *mc;
...
if( ( dc = PpContinueJob( pc ) ) == -1 )
{
    perror( "unable to activate print context" );
}
else{
    // do print stuff

    // Then restore context which was active before we started printing. This is equivalent to doing
    // a PpSuspendJob() followed by a PmMemStart(), or PpContinueJob(), depending on what type of draw
    // context was active previously.

    PhDCSetCurrent( dc );
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
</tbody>
</table>

continued...
**PhDCSetCurrent()**

| Safety | Thread | No |

See also:

*PhDCCreate(), PhDCGetCurrent(), PhDCRelease(), PmMemCreateMC(), PpContinueJob(), PpCreatePC(), PpPrintContext_t*
Synopsis:
\[ \text{int PhDetach( struct _Ph_ctrl *Ph );} \]

Library:
\[ \text{ph} \]

Description:
This function frees all resources consumed by the Photon channel \( Ph \).
If \( Ph \) is the current channel, no current channel will exist after this function is called.

\( Ph \) is a pointer to a Photon control structure returned by a previous call to \( \text{PhAttach()} \).

Returns:
\[ \begin{align*}
0 & \quad \text{Successful completion.} \\
-1 & \quad \text{An error occurred.}
\end{align*} \]

Examples:
See \( \text{PhAttach()} \).

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PhDetach()
**PhDeTranslateRect()**

*Detranslate a rectangle (subtract offset)*

**Synopsis:**

```c
PhRect_t *PhDeTranslateRect(  
    PhRect_t *rect,           
    PhPoint_t const *delta );
```

**Library:**

`ph`

**Description:**

This convenience function subtracts `delta->x` from `rect->ul.x` and `rect->lr.x`, and subtracts `delta->y` from `rect->ul.y` and `rect->lr.y`. You’ll find this function handy for translating events, extents, or canvases so they become relative to various points.

**Returns:**

A pointer to the `rect` argument.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler No</td>
</tr>
<tr>
<td>Signal handler No</td>
</tr>
<tr>
<td>Thread No</td>
</tr>
</tbody>
</table>

**See also:**

`PhPoint_t, PhRect_t, PhTranslateRect()`

“Geometry data types” in the Working with Code chapter of the Photon *Programmer’s Guide*
**PhDeTranslateTiles()**
Subtract x and y offsets from the vertices of a list of tiles

**Synopsis:**

```c
PhTile_t * PhDeTranslateTiles(
    PhTile_t *tile,
    PhPoint_t const *point_subtract);
```

**Library:**

`ph`

**Description:**

This function subtracts the coordinates of `point_subtract` from the vertices of each tile in the list pointed to by `tile`.

Don’t `free()` the list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.

**Returns:**

The same pointer as `tile`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`,
`PhCopyTiles()`, `PhFreeTiles()`, `PhGetTile()`, `PhIntersectTilings()`,
PhDeTranslateTiles()

PhMergeTiles(), PhPoint_t, PhRectsToTiles(), PhSortTiles(), PhTile_t, PhTilesToRects(), PhTranslateTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhDim_t

Dimensions of an area

Synopsis:

```c
typedef struct Ph_dim {
    unsigned short w, h;
} PhDim_t;
```

Description:

The PhDim_t structure defines the dimensions of an area. It contains at least the following members:

- `w` Width of the area.
- `h` Height of the area.

Classification:

Photon

See also:

- PhArea_t, PhPoint_t, PhRect_t
- “Geometry data types” in the Working with Code chapter of the Photon Programmer’s Guide
Synopsis:

```c
typedef struct Ph_ev_drag_data {
    PhRect_t    rect;
    PhRid_t     rid;
    PhRect_t    boundary;
    PhDim_t     min;
    PhDim_t     max;
    PhDim_t     step;
    PhPoint_t   pos;
    unsigned long   key_mods;
    ushort_t    flags;
    unsigned short   button_state;
} PhDragEvent_t;
```

Description:

The `PhDragEvent_t` structure defines the data associated with `Ph_EV_DRAG` events (see `PhEvent_t`). It contains at least the following members:

- `rect` A `PhRect_t` structure that contains the coordinates of the initial, current, or final drag rectangle, depending on the drag-event subtype value.

- `rid` The ID of the region that initiated the drag operation. Your application needs to specify the region ID when it calls `PhInitDrag()` to initiate the dragging operation.

- `boundary` A `PhRect_t` structure that contains the coordinates of the rectangle that constrains the drag operation.

- `min`, `max` `PhDim_t` structures that define the minimum and maximum sizes of the drag rectangle, as specified in the call to `PhInitDrag()`.

- `step` A `PhDim_t` structure that defines the drag operation’s granularity, as specified in the call to `PhInitDrag()`.

- `pos` A `PhPoint_t` structure that indicates the current cursor position. This position isn’t necessarily within the `boundary` rectangle.
Your application can use the modifier keys (e.g. Shift or Num Lock) to change the meaning of a drag event. When a modifier key is pressed or released, it’s evaluated through a table, and the key_mods field is updated accordingly. This evaluation is done before the drag event is sent.

The key_mods member is a combination of the following bits:

- Pk_KM_Shift
- Pk_KM_Ctrl
- Pk_KM_Alt
- Pk_KM_AltGr
- Pk_KM_Shl3
- Pk_KM_Mod6
- Pk_KM_Mod7
- Pk_KM_Mod8
- Pk_KM_Shift_Lock
- Pk_KM_Ctrl_Lock
- Pk_KM_Alt_Lock
- Pk_KM_AltGr_Lock
- Pk_KM_Shl3_Lock
- Pk_KM_Mod6_Lock
- Pk_KM_Mod7_Lock
- Pk_KM_Mod8_Lock
- Pk_KM_Caps_Lock
- Pk_KM_Num_Lock
- Pk_KM_Scroll_Lock

If the Shift key is pressed, the Shift modifier is on; if it’s released, the Shift modifier is off. Because some keys occur twice on the keyboard, a key release doesn’t guarantee that the corresponding modifier is off — the matching key may still be pressed.
unsigned short flags

Flags that indicate which edges of the drag rectangle track the pointer. You can OR the following values into flags:

- Ph_DRAG_NOBUTTON—Allow the drag to start, even if the user isn’t holding down a button.
- Ph_DRAG_KEY_MOTION—During the drag, emit drag events with the Ph_EV_DRAG_KEY_EVENT or the Ph_EV_DRAG_MOTION_EVENT subtype (see PhEvent_t).
- Ph_DRAG_TRACK—No drag outline is drawn, and Ph_EV_DRAG_MOVE events are emitted to the initiating region. Use this flag if you want to implement your own visual interpretation of dragging operations.
- Ph_TRACK_LEFT—left edge tracks the pointer.
- Ph_TRACK_RIGHT—right edge tracks the pointer.
- Ph_TRACK_TOP—top edge tracks the pointer.
- Ph_TRACK_BOTTOM—bottom edge tracks the pointer.
- Ph_TRACK_DRAG—all edges track the pointer (the same as using all four of the above values).

unsigned short button_state

The current state of the pointing-device buttons (i.e. which buttons are currently pressed):

- Ph_BUTTON_SELECT
- Ph_BUTTON_MENU
- Ph_BUTTON_ADJUST
PhDragEvent_t

Classification:
Photon

See also:
PhDim_t, PhEvent_t, PhInitDrag(), PhPoint_t, PhRect_t

“Dragging” in the Events chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PhEmit( PhEvent_t const *event,
            PhRect_t const *rects,
            void const *data );
```

**Library:**

`ph`

**Description:**

This function emits the event described by the given `PhEvent_t` structure.

The `rects` argument points to an array of `PhRect_t` structures that define the rectangles associated with the event. If `event->num_rects` isn’t 0, then `rects` must point to an array of `event->num_rects` valid rectangles.

The `data` argument points to variable-length event-specific data. If `event->data_len` isn’t 0, then `data` must point to a buffer of at least `event->data_len` bytes.

If you set the collector ID (`event->collector.rid`) to zero, the event is enqueued to every appropriately sensitive region that intersects with the event. If you set `collector.rid` to a region ID, only that region notices the event.

The Photon library fills in the `collector` and `translation` fields in the `PhEvent_t` structure after a copy of the event is enqueued to an application.

**Returns:**

A nonnegative value

- Successful completion.
- An error occurred; check the value of `errno`.
Ph Emit()

Examples:

The following example emits an expose event from the device region. Because the event covers the entire event space, any visible part of the event space is refreshed:

```c
#include <stdio.h>
#include <time.h>
#include <Ph.h>

int main( int argc, char *argv[] )
{
    PhEvent_t event = { 0 };
    PhRect_t rect;

    if( NULL == PhAttach( NULL, NULL ) ) {
        fprintf( stderr, "Couldn't attach Photon channel.\n" );
        exit( EXIT_FAILURE );
    }
    event.type = Ph_EV_EXPOSE;
    event.subtype = 0;
    event.flags = 0;
    event.num_rects = 1;
    event.data_len = 0;
    event.emitter.rid = Ph_DEV_RID;
    rect.ul.x = rect.ul.y = SHRT_MIN;
    rect.lr.x = rect.lr.y = SHRT_MAX;
    PhEmit( &event, &rect, NULL );

    return EXIT_SUCCESS;
}
```

Classification:

Photon

Safety

- Interrupt handler: No
- Signal handler: No
- Thread: No
See also:

PhEvent_t, PhEmitmx(), PhEventNext(), PhEventPeek(), PhEventRead(), PhRect_t, PtSendEventToWidget()

**PhEmitmx()**

*Emit an event when the event-specific data isn’t contiguous in memory*

**Synopsis:**

```c
int PhEmitmx( PhEvent_t const *event,
             PhRect_t const *rects,
             int mxpath,
             iov_t *mx );
```

**Library:**

`ph`

**Description:**

This function provides an alternative to `PhEmit()`. You’ll find it useful when the event-specific data isn’t contiguous in memory.

The `mx` argument points to an array of `iov_t` entries, and `mxpath` contains the number of `mx` entries pointed to by `mx`. You should leave the first three entries of `mx` blank; these are filled in by `PhEmitmx()`. You can use the remaining entries to build a description of the data to be attached to the event. If `event->data_len` isn’t 0, then the event data must be at least `event->data_len` bytes long.

The `event` argument points to a `PhEvent_t` structure.

The `rects` argument points to an array of `PhRect_t` structures that define the rectangles associated with the event. If `event->num_rects` isn’t 0, then `rects` must point to an array of `event->num_rects` valid rectangles.

**Returns:**

A nonnegative value

Successful completion.

-1 An error occurred; check the value of `errno`.

**Examples:**

The following example emits a pointer press event. (A call to `PhEmit()` is just as efficient and slightly more convenient.)
```
#include <stdio.h>
#include <time.h>
#include <Ph.h>

int main( int argc, char *argv[] )
{
    PhEvent_t event = { 0 },
    PhRect_t rect,
    PhPointerEvent_t ptr_event;
    iov_t mx[4];

    if( NULL == PhAttach( NULL, NULL ) ) {
        fprintf( stderr,
            "Could not attach a Photon channel.\n"");
        exit( EXIT_FAILURE );
    }
    event.type = Ph_EV_BUT_PRESS;
    eventsubtype = 0;
    event.emitter.rid = Ph_DEV_RID;
    event.flags = 0;
    event.num_rects = 1;
    event.data_len = sizeof( ptr_event );
    rect.ul.x = rect.lr.x = 100;
    rect.ul.y = rect.lr.y = 200;
    ptr_event.flags = 0;
    ptr_event.buttons = Ph_BUTTON_SELECT;
    SETIOV( &mx[3], &ptr_event, sizeof( ptr_event ) );
    PhEmitmx( &event, &rect, 4, mx );

    return EXIT_SUCCESS;
}
```

**Classification:**

 Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PhEmitmx()**

**See also:**

*PgFlush(), PhEvent_t, PhEmit(), PhPointerEvent_t, PhRect_t*


*MsgSendv(), SETIOV() in the QNX Neutrino Library Reference*
**PhEvent_t**

Data structure describing an event

Synopsis:

```c
typedef struct Ph_event {
    unsigned long type;
    unsigned short subtype;
    unsigned short processing_flags;
    PhEventRegion_t emitter;
    PhEventRegion_t collector;
    unsigned short input_group;
    unsigned short flags;
    unsigned long timestamp;
    PhPoint_t translation;
    unsigned short num_rects;
    unsigned short data_len;
} PhEvent_t;
```

Description:

The `PhEvent_t` structure describes an event. It contains at least the following members:

- **type**
  One — and only one — of the predefined event types:
  - `Ph_EV_BOUNDARY`
  - `Ph_EV_BUT_PRESS`
  - `Ph_EV_BUT_RELEASE`
  - `Ph_EV_BUT_REPEAT`
  - `Ph_EV_DNDROP`
  - `Ph_EV_DRAG`
  - `Ph_EV_DRAW`
  - `Ph_EV_EXPOSE`
  - `Ph_EV_INFO`
  - `Ph_EV_KEY`
  - `Ph_EV_PTR_MOTION_BUTTON`
  - `Ph_EV_PTR_MOTION_NOBUTTON`
PhEvent_t

- Ph_EV_RAW
- Ph_EV_SERVICE
- Ph_EV_SYSTEM
- Ph_EV_TIMER
- Ph_EV_WM

These types are described below. The event type determines how the data associated with the event is interpreted.

**subtype**
Further information about the event. For the possible values of subtype, see the description of each event type.

**processing_flags**
Flags used or set in processing the event:

- Ph_BACK_EVENT — the event has gone down the widget family hierarchy and is now on its way back up.
- Ph_CONSUMED — the event has been consumed. (When a widget has processed an event and prevents another widget from interacting with the event, the first widget is said to have consumed the event.)
- Ph_DIRECTED_FOCUS — the event has caused focus to change.
- Ph_FAKE_EVENT — set this bit if the event is a fake one created by your application.
- Ph_FOCUS_BRANCH — focus is changing, and the current widget is on the focus path, but isn’t the destination.
- Ph_TYPE_SPECIFIC — a mask for bits that are specific to the type of event.
- Ph_USER_RSRVD_BITS — a mask for bits that you can use for your own purposes.
**PhEvent_t**

*emitter*  
A `PhEventRegion_t` structure identifying the region that emitted the event.

Your application can emit an event from a region — even one it doesn’t own — by setting *emitter* to the ID of that region. You can use this approach when to target the device region by setting the `Ph_EVENT_INCLUSIVE` flag.

*collector*  
A `PhEventRegion_t` structure identifying the region that collected the event. When your process has many regions open, *collector* lets you distinguish which of its regions was involved.

*input_group*  
The number of the input group. A value of 0 means there’s no input group.

*flags*  
Event-modifier flags. You can OR the following values into flags:

- **Ph_EVENT_ABSOLUTE**
  
  Forces the rectangle set associated with the event to be relative to the root region’s origin. By default, the coordinates of the rectangle set are relative to the origin of the emitting region.

- **Ph_EVENT_DIRECT**
  
  Emits the event directly from *emitter* to *collector*.

- **Ph_EVENT_INCLUSIVE**
  
  Emits the event first to the emitting region and then through the event space. Using this flag, an application can guarantee that the emitter sees the event (assuming the emitter is sensitive to that event type).

- **Ph_EMIT_TOWARD**
  
  Emits the event toward the user. By default, events are emitted away from the user.
PhEvent_t

**timestamp**
When the event was emitted, in milliseconds. The Photon Manager generates this member.

**translation**
A PhPoint_t structure that specifies the translation between the emitting region’s origin and the collecting region’s origin. An application uses this member to convert coordinates that are relative to the emitter’s region to coordinates that are relative to the collector’s region.

For example, let’s say the graphics driver wants to render Ph_EV_DRAW events. When these events reach the driver, they contain coordinates relative to the region that emitted them. To render these events within its own region, the graphics driver uses translation to convert the coordinates.

**num_rects**
The number of rectangles associated with the event.
To extract the list of rectangles, see PhGetRects().

**data_len**
The length of the data associated with the event.
Since event data is optional, you can set data_len to 0 when there’s no data. To extract the data from an event, see PhGetData().

Ph_EV_BOUNDARY

Emitted when the pointer crosses region boundaries. The subtype member of the PhEvent_t structure indicates one of the following boundary conditions:

- Ph_EV_PTR_ENTER*
- Ph_EV_PTR_LEAVE*

Emitted when the region the cursor points at changes. Both the previous and current regions must have the Ph_FORCE_BOUNDARY flag set, since Photon only considers regions with this bit set to be pointed at by the cursor. The cursor will always point at something, since the root region has this bit set. When the region pointed to changes, any regions
between the previous and new regions (that is, regions in the common ancestor tree) regardless of whether they have the \texttt{Ph\_FORCE\_BOUNDARY} flag set also receive boundary events. Since these events are emitted directly to the region, they are not affected by opacity to boundary events. The event subtypes are:

- \texttt{Ph\_EV\_PTR\_ENTER\_FROM\_PARENT} — emitted to a child region when the pointer enters it from a parent region. Formerly \texttt{Ph\_EV\_PTR\_ENTER}, which is deprecated.
- \texttt{Ph\_EV\_PTR\_ENTER\_FROM\_CHILD} — emitted to a parent region when the pointer enters it from a child region.
- \texttt{Ph\_EV\_PTR\_LEAVE\_TO\_PARENT} — emitted to a child region when the pointer leaves it to a parent region. Formerly \texttt{Ph\_EV\_PTR\_LEAVE}, which is deprecated.
- \texttt{Ph\_EV\_PTR\_LEAVE\_TO\_CHILD} — emitted to a parent region when the pointer leaves it to a parent region.

\texttt{Ph\_EV\_PTR\_STEADY}

Emitted when the pointer remains motionless for 1.25 seconds. Another \texttt{Ph\_EV\_PTR\_STEADY} won’t be emitted until the user moves the pointer and then lets it remain motionless again. This event is propagated through the Photon space, starting from the device region, and therefore is affected by opacity to boundary events.

\texttt{Ph\_EV\_PTR\_UNSTEADY}

Emitted when the pointer is moved after a \texttt{Ph\_EV\_PTR\_STEADY} is emitted. Another \texttt{Ph\_EV\_PTR\_UNSTEADY} won’t be emitted until the user allows the pointer to remain motionless and then moves it again. This event is propagated through the Photon space, starting from the device region, and therefore is affected by opacity to boundary events.
PhEvent_t

**Ph_EV_BUT_PRESS**

Emitted when the user presses a button on a pointing device. This event’s rectangle set consists of a point source that indicates the current pointer focus. The event data is a `PhPointerEvent_t` structure.

**Ph_EV_BUT_RELEASE**

Emitted when the user releases a pointing-device button. This event’s rectangle set consists of a point source that indicates the current pointer focus. The event data is a `PhPointerEvent_t` structure. However, in this case, the `buttons` member indicates the buttons that were released.

This event type has the following subtypes:

- **Ph_EV_RELEASE_REAL**
  
  Emitted at the current position of the pointer (that is, where the user actually released the button).

- **Ph_EV_RELEASE_PHANTOM**
  
  Emitted where the user pressed the button.

- **Ph_EV_RELEASE_ENDCLICK**
  
  Emitted when multiclicks are no longer possible, i.e. when the user moves the mouse or stops clicking for a while.

- **Ph_EV_RELEASE_OUTBOUND**
  
  Emitted when the user starts dragging, i.e. when the mouse is moved a few pixels but the button hasn’t been released.

**Ph_EV_BUT_REPEAT**

Emitted when the user presses an auto-repeating button on a pointing device. This event is emitted each time the button repeats. Its rectangle set consists of a point source that indicates where the button was pressed. The event data is a `PhPointerEvent_t` structure.
These events are emitted during a drag-and-drop operation.

Ph_EV_DNDROP events with these subtypes are emitted to the source of the operation:

Ph_EV_DND_INIT
The operation has started successfully.

Ph_EV_DND_CANCEL
The operation was canceled (for example, if the drop occurred when not over a drop zone, or the destination terminated the operation before receiving the drop or before it finished fetching requestable data).

If the operation is canceled in this way, the library cleans up the data structures automatically.

Ph_EV_DND_COMPLETE
The drag-and-drop event is enqueued at the destination (the destination hasn’t seen it yet).

Ph_EV_DND_DELIVERED
The destination has dequeued the drag-and-drop event.

These subtypes of a drag-and-drop event are emitted to the destination of the operation:

Ph_EV_DND_ENTER
The pointer has moved into the widget’s region but no drop has occurred. This is the reason_subtype the first time that the drag-and-drop callback is called.

Ph_EV_DND_MOTION
The pointer is moving inside the widget’s region. This type of event is emitted only if the Pt_DND_SELECT_MOTION bit is set in the select_flags member of the PtDndFetch_t structure for a piece of selected data.
**PhEvent_t**

Ph_EV_DND_REPEAT

This type of event is emitted periodically when Ph_EV_DND_MOTION events have been requested. The destination might want to track Ph_EV_DND_REPEAT events in, for example, a list widget, to select more than one item to be replaced by dropped data.

Ph_EV_DND_DROP

The user has dropped the data.

Ph_EV_DND_LEAVE

The pointer has moved out of the widget’s region, but the user didn’t drop the data.

Events with these subtypes are emitted internally to the Photon server to accept or deny a drop:

- Ph_EV_DND_ACK
- Ph_EV_DND_NAK

Ph_EV_DRAG

Used by an application to initiate drag events, to determine their completion, and to indicate intermediate drag-motion events.

The event data is a PhDragEvent_t structure.

The Ph_EV_DRAG event can have any of the following subtypes:

Ph_EV_DRAG_BOUNDARY

Emitted when rect hits a boundary. The flags member of the PhDragEvent_t structure specifies which boundary.

Ph_EV_DRAG_COMPLETE

When the user completes the drag operation, the device region emits a Ph_EV_DRAG event with this subtype toward the root region so that the initiating application collects the event. This event is direct.
PhEvent_t

Ph_EV_DRAG_INIT
To initiate a drag operation, an application must emit a Ph_EV_DRAG event with this subtype to the device region. The Photon Manager takes care of the user’s interaction with the screen pointer and the drag outline.

The PhInitDrag() function, which emits Ph_EV_DRAG_INIT, provides a convenient way to initiate drag operations. You can cancel drag operations with PhCancelDrag()

Ph_EV_DRAG_KEY_EVENT
Emit the event with a PhKeyEvent_t structure.

Ph_EV_DRAG_MOTION_EVENT
Emit the event with a PhPointerEvent_t structure.

Ph_EV_DRAG_MOVE
Indicates intermediate drag motion. The Photon Manager emits this drag-event subtype if the Ph_DRAG_TRACK flag is set in the flags member of the PhDragEvent_t structure when the drag operation is initiated.

Ph_EV_DRAG_START
Emitted when the server begins the drag operation.

Ph_EV_DRAW
Emitted by the Pg functions when applications perform draw operations. The event travels toward the user and is collected by the graphics driver.

The event data is a PhDrawEvent_t structure that contains at least the following members:

unsigned short cmd_buffer_size
Size of the draw buffer, in bytes.
PhEvent_t

unsigned long id

An ID number that’s unique for each application in this Photon space. The Pg functions set this number, which is used to optimize drawing operations.

Ph_EV_EXPOSE

Emitted by the Photon Manager on behalf of a region being moved, resized, or removed from the event space. The event travels away from the user and appears to originate from the removed region.

Since any regions now exposed see the expose event, an application can determine which of its regions have been uncovered. It can then redraw any portion of the regions that become visible by passing the rectangle set to PgSetClipping(). This event’s rectangle set describes those areas that are now exposed. This event has no associated data.

The Ph_EV_EXPOSE event can have any of the following subtypes:

Ph_NORMAL_EXPOSE

Emitted when a region is moved, resized, or removed from the event space. This is the most common type of expose.

Ph_CAPTURE_EXPOSE

Emitted by an application (typically a printer driver) that wishes to receive an encapsulated draw event starting with:

```c
PgFFlush (Ph_START_DRAW);
```

and ending with:

```c
PgFFlush (Ph_DONE_DRAW);
```

when the applications that received the expose have completed their updates.

This type of event indicates that the expose wasn’t caused by a region change. You can use this event type to collect data for the purpose of producing some form of hardcopy.
PhEvent_t

Ph_EVENT_GRAPHIC_EXPOSE

Emitted by a graphics driver. This subtype indicates that no region was moved, removed, or resized to generate the expose event.

Ph_EVENT_INFO

All regions must always be transparent to Ph_EVENT_INFO events. They are emitted by applications or service providers to disseminate information or respond to requests. The currently defined subtypes are:

Ph_EVENT_INVALIDATE_SYSINFO

Emitted by Photon as regions are moved, created, or destroyed. The application must ask Photon for updated system information should a need for this information arise. This is handled automatically by the widget library. The event data is NULL.

Ph_EVENT_FEP

Emitted primarily by FEP service providers to inform applications of their presence or impending absence. The data portion of the event is a PhFEPInfo_t structure that contains at least the following members:

long type

The valid types are:

- Ph_FEP_REGISTER — a FEP has been launched (all applications can see the event), or is responding to a Ph_FEP_BROADCAST service message (seen only by the application requesting the broadcast).
- Ph_FEP_DEREGISTER — a FEP is shutting down.
PhEvent_t

**long subtype** The language type of the FEP. The valid subtypes are:
  - Ph_FEP_JAPANESE
  - Ph_FEP_CHINESE
  - Ph_FEP_KOREAN

**long len** Not currently used.

**char data [1]** Not currently used.

**Ph_OFFSCREEN_INVALID**

Emitted when an offscreen context is invalidated by the graphics driver for any reason. Applications planning on using offscreen contexts should be sensitive to this event and reinitialize their off screen contexts accordingly. The data portion of this event is a single `long` describing why the offscreen areas have been invalidated. The defined types are:

**Pg_VIDEO_MODE SWITCHED**

The graphics driver has changed video modes.

**Pg_ENTERED_DIRECT**

An application has entered direct mode.

**Pg_EXITED_DIRECT**

An application has left direct mode.

**Pg_DRIVER_STARTED**

The video driver has just started execution.

**Ph_EV_KEY**

Emitted when a key state changes (for example, the user presses or releases a key). This event’s rectangle set consists of a point source that indicates the current focus. The event data is a `PhKeyEvent_t` structure.

The `processing_flags` member of the `PhEvent_t` structure for this event type also include:
**PhEvent_t**

Ph\_NOT\_CUAKEY

Force **PtContainer** *not* to use the key for traversal (CUA).

Ph\_NOT\_HOTKEY

Force **PtContainer** *not* to treat the key as a hotkey.

**Ph\_EV\_PTR\_MOTION\_BUTTON**

Emitted when the user moves the pointing device while pressing a button. This event’s rectangle set consists of a point source that indicates the current pointer focus. The event data is a **PhPointerEvent_t** structure. The *buttons* member indicates which buttons the user is pressing.

**Ph\_EV\_PTR\_MOTION\_NOBUTTON**

Emitted when the user moves the pointing device *without* pressing a button. This event’s rectangle set consists of a point source that indicates the current pointer focus. The event data is a **PhPointerEvent_t** structure.

Large numbers of Ph\_EV\_PTR\_MOTION\_NOBUTTON events can slow down your system. To avoid this, you should make your applications sensitive to Ph\_EV\_PTR\_MOTION\_BUTTON whenever possible, rather than to Ph\_EV\_PTR\_MOTION\_NOBUTTON.

**Ph\_EV\_RAW**

These are raw, unfocused events that the Photon server handles.

**Ph\_EV\_SERVICE**

These events may be emitted by applications requesting services or providing information to services, other applications that provide some kind of service in a Photon system. The currently defined subtypes are:
PhEvent_t

Ph_EV_REMOTE_WM

Handled by relay-type services such as phrelay (see the QNX Neutrino Utilities Reference).
Normally only emitted by a Window Manager to synchronize a remote Window Manager’s state. The event data is a PhRemoteWMEvent_t structure that contains at least the following members:

short type  Valid values of type are
             REMOTE_WM_WINDOW or
             REMOTE_WM_TITLE.
short len  Not used.

If type is REMOTE_WM_WINDOW, the window member is also defined. The window member has at least the following members:

ushort_t xpos  New absolute x coordinate of the window.
ushort_t ypos  New absolute y coordinate of the window.
ushort_t height  New height dimension of the window.
ushort_t width  New width dimension of the window.
short flags  Valid flag bits are:
             REMOTE_FLAG_FIXED
             Window shouldn’t be resized by the Window Manager; the application resizes it.
             REMOTE_FLAG_INITIAL
             New window.
**PhEvent_t**

REMOTE_FLAG_IS_ORIGIN

Use xpos, ypos as the new origin.

REMOTE_FLAG_NO_DIM

The dim variable shouldn’t be modified.

If type is REMOTE_WM_TITLE, the title member is defined as follows:

```c
char title[64] A title for the window.
```

**Ph_EV_FEP**

Handled by Front End Processor (FEP) service providers (e.g. Japanese input). The event data is a PhFEPService_t structure that contains at least the following members:

```c
long type The valid types are:
```

**Ph_FEP_BROADCAST**

Request a broadcast from FEP service. If a FEP is present, it responds with an Ph_FEP_REGISTER register event.

**Ph_FEP_RECT**

Give rectangle (for pre-edit window and cursor) to FEP services. The pre-edit rectangle is defined by the event rectangle. The cursor rectangle is defined by this
**PhEvent_t**

structure’s rectangle member.

Ph_FEP_NORECT
Invalidate rectangle in FEP service.

Ph_FEP_ACTIVATE
Request activation of FEP filter.

Ph_FEP_DEACTIVATE
Request deactivation of FEP filter.

long len
Not used.

PhRect_t rect
The cursor rectangle relative to the event rectangle.

long num_rids
The number of regions that are parents of this region (the region owned by the currently focused widget). An array of num_rids RIDs should be appended to the event data. The first RID in this list should be the RID of the focused widget; otherwise num_rids should be set to 0.

**Ph_EV_SYSTEM**

Ph_EV_SYSTEM events are emitted when Photon or a service wants to inform applications of changes in the system. The event data is a PhSystemEvent_t union. The valid member is dictated by the subtype of the system event.
If the event subtype is Ph_SYSTEM_REGION_CHANGE, the valid union member is RegionChange, which contains at least the following members:

**PhRid_t rid**
ID of the region that changed.

**PhPoint_t origin**
A PhPoint_t structure that specifies the origin of the region, relative to its parent’s region.

**PhRect_t rect**
Its rectangle, relative to its origin.

**ulong_t flags**
The region’s flags.

**ulong_t fields**
A set of bits indicating which fields of the PhRegion_t structure were modified:
- 0xFFFFFFFF — the region was opened.
- 0x00000000 — the region was closed.
- Other values — fields with a 1 bit were changed.

For more information, see:
- PhRegion_t
- PhRegionOpen()
- PhRegionChange()
- <PhT.h>

**unsigned short input_group**
Nonzero if the region being changed belongs to an input group.

Ph_EV_TIMER

Emitted by an application directly to the Device region to request a reciprocal event after a specific amount of time has elapsed (arm a timer). This is usually done via PtTimerArm() or PhTimerArm().

It is also emitted by Photon when an armed timer expires. In both cases, the event data is a PhTimerEvent_t structure that contains the following members:
**PhEvent_t**

```c
unsigned msec
unsigned zero
PhEventRegion_t region
```

`region.rid` is the RID, and `region.handle` is a pointer to the widget specified as the handle in a `PtTimerArm()` call.

When this event is received by the widget library, it delivers the event directly to the widget designated by `region.handle`. It’s best to avoid setting `region.handle` to anything other than a valid widget pointer.

**Ph_EV_WM**

Both the Window Manager and applications can emit this event. The Window Manager emits this event when an application has asked to be notified. An application can emit this event to communicate to the Window Manager regarding windows.

Ph_EV_WM can have the following subtype:

**Ph_EV_WM_EVENT**

The rectangle set of the event has no useful value. The event data is a `PhWindowEvent_t` structure.

**Classification:**

Photon

**See also:**

`PhCancelDrag()`, `PhDragEvent_t`, `PhEventRegion_t`, `PhGetData()`, `PhGetRects()`, `PhInitDrag()`, `PhKeyEvent_t`, `PhPoint_t`, `PhPointerEvent_t`, `PhRect_t`, `PhWindowEvent_t`, `PtDndFetch_t`

Events chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
int PhEventArm( void );
```

Library:

ph

Description:

This function arms the current Photon channel so that the channel will send a Neutrino pulse to the application when an event becomes available. If an event is already available, the pulse is sent immediately.

You must call this function before calling `PhEventRead()` for the first time.

Returns:

- 0  Successful completion.
- -1  An error occurred.

Examples:

See `PhEventRead()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PhEventArm()**

See also:

*PhAttach(), PhEventRead()*

“Collecting events” in the Events chapter of the Photon *Programmer’s Guide*
PhEventEmit()

Emit an event

Synopsis:

```c
int PhEventEmit( PhEvent_t const *event,
                 PhRect_t const *rects,
                 void const *data );
```

Library:

ph

Description:

This function emits the event described by the given PhEvent_t structure. PhEmit() does the same things as PhEventEmit(), but provides a cleaner API.

The rects argument points to an array of PhRect_t structures that define the rectangles associated with the event. If event->num_rects isn’t 0, then rects must point to an array of event->num_rects valid rectangles.

The data argument points to variable-length event-specific data. If event->data_len isn’t 0, then data must point to a buffer of at least event->data_len bytes.

If you set the collector ID (event->collector.rid) to zero, the event is enqueued to every appropriately sensitive region that intersects with the event. If you set collector.rid to a region ID, only that region notices the event.

The Photon library fills in the collector and translation fields in the PhEvent_t structure after a copy of the event is enqueued to an application.

Returns:

0 Success

-1 An error occurred, or no further events are pending. Check the value of errno:

  - If errno is ENOMSG, Photon had no messages enqueued to your application at the time you emitted the event.
PhEventEmit()

- If *errno* isn’t ENOMSG, an error occurred.

These return codes are useful for applications that spend most of their time emitting events and want to retrieve an event only if there’s one pending for them.

**Examples:**

The following example emits an expose event from the device region. Because the event covers the entire event space, any visible part of the event space is refreshed:

```c
#include <stdio.h>
#include <time.h>
#include <Ph.h>

int main( int argc, char *argv[] )
{
    PhEvent_t event = { 0 };
    PhRect_t rect;

    if( NULL == PhAttach( NULL, NULL ) ) {
        fprintf( stderr,
                "Couldn't attach Photon channel.\n"");
        exit( EXIT_FAILURE );
    }
    event.type = Ph_EV_EXPOSE;
    event.subtype = 0;
    event.flags = 0;
    event.num_rects = 1;
    event.data_len = 0;
    event.emitter.rid = Ph_DEV_RID;
    rect.ul.x = rect.ul.y = SHRT_MIN;
    rect.lr.x = rect.lr.y = SHRT_MAX;
    PhEventEmit( &event, &rect, NULL );

    return EXIT_SUCCESS;
}
```

**Classification:**

Photon
**PhEventEmit()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

*PhEmit(*), PhEmitmx(*), PhEvent_t, PhEventEmitmx(*), PhEventNext(*), PhEventPeek(*), PhEventRead(*), PhRect_t, PtSendEventToWidget(*)

**PhEventEmitmx()**

Emit an event when the event-specific data isn’t contiguous in memory

**Synopsis:**

```c
int PhEventEmitmx( PhEvent_t const *event,
                    PhRect_t const *rects,
                    int mparts,
                    iov_t *mx );
```

**Library:**

ph

**Description:**

This function provides an alternative to PhEventEmit(). You’ll find it useful when the event-specific data isn’t contiguous in memory.

PhEmitmx() does the same things as PhEventEmitmx() and provides a cleaner API.

The mx argument points to an array of iov_t entries, and mparts contains the number of mx entries pointed to by mx. You should leave the first three entries of mx blank; these are filled in by the PhEventEmitmx() call. You’re free to use the remaining entries to build a description of the data to be attached to the event. If event->data_len isn’t 0, then the event data must be at least event->data_len bytes long.

The rects argument points to an array of PhRect_t structures that define the rectangles associated with the event. If event->num_rects isn’t 0, then rects must point to an array of event->num_rects valid rectangles.

**Returns:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred, or no further events are pending. Check the value of errno:</td>
</tr>
<tr>
<td></td>
<td>• If errno is EMSG, Photon had no messages enqueued to your application at the time you emitted the event.</td>
</tr>
<tr>
<td></td>
<td>• If errno isn’t EMSG, an error occurred.</td>
</tr>
</tbody>
</table>
These return codes are useful for applications that spend most of their time emitting events and want to retrieve an event only if there’s one pending for them.

**Examples:**

The following example emits a pointer press event. (A call to `PhEventEmit()` is just as efficient and slightly more convenient.)

```c
#include <stdio.h>
#include <time.h>
#include <Ph.h>

int main( int argc, char *argv[] )
{
    PhEvent_t event = { 0 };
    PhRect_t rect;
    PhPointerEvent_t ptr_event;
    iov_t mx[4];

    if( NULL == PhAttach( NULL, NULL ) ) {
        fprintf( stderr,
            "Could not attach a Photon channel.\n"");
        exit( EXIT_FAILURE );
    }
    event.type = Ph_EV_BUT_PRESS;
    event.subtype = 0;
    event.emitter.rid = Ph_DEV_RID;
    event.flags = 0;
    event.num_rects = 1;
    event.data_len = sizeof( ptr_event );
    rect.ul.x = rect.lr.x = 100;
    rect.ul.y = rect.lr.y = 200;
    ptr_event.flags = 0;
    ptr_event.buttons = Ph_BUTTON_SELECT;
    SETIOV( &mx[3], &ptr_event, sizeof( ptr_event ) );
    PhEventEmitmx( &event, &rect, 4, mx );

    return EXIT_SUCCESS;
}
```
PhEventEmitmx()

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgFlush(), PhEmit(), PhEmitmx(), PhEvent_t, PhEventEmit(), PhPointerType_t, PhRect_t


MsgSendv(), SETIOV() in the QNX Neutrino Library Reference
PhEventNext()

Provide synchronous event notification

Synopsis:

```c
int PhEventNext( void *buffer,
                 unsigned size );
```

Library:

ph

Description:

This function provides a completely synchronous event-notification mechanism. It causes the application to become REPLY-blocked on the currently attached Photon channel until an event occurs.

For asynchronous event notification, see PhEventRead() and PhEventArm().

If the application’s event queue contains an event when this call is made, Photon replies immediately with that event.

If your application uses widgets, don’t try to write your own event-handling loop; use PtMainLoop() or PtProcessEvent() instead.

Returns:

Ph_EVENT_MSG

Successful completion.

Ph_RESIZE_MSG

The Ph_DYNAMIC_BUFFER flag was set in PhAttach(), and there’s a pending Photon event that’s larger than the client’s event buffer. This event that indicates how large the client’s buffer needs to be to receive the entire event message.

-1 An error occurred.
**PhEventNext()**

**Examples:**

```c
#define EVENT_SIZE sizeof( PhEvent_t ) + 1000

main( int argc, char *argv[] )
{
    PhEvent_t *event;

    if( initialize() == -1 )
        exit( EXIT_FAILURE );

    if( NULL == ( event = malloc( EVENT_SIZE ) ) )
        exit( EXIT_FAILURE );

    while( 1 ) {
        switch( PhEventNext( event, EVENT_SIZE ) ) {
            case Ph_EVENT_MSG:
                PtEventHandler( event );
                break;
            case -1:
                perror( "PhEventNext failed" );
                break;
        }
    }
}
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhAttach()`, `PhEvent_t`, `PhEventPeek()`, `PhEventRead()`, `PhGetMsgSize()`, `PtEventHandler()`

“Collecting events” in the Events chapter of the Photon *Programmer’s Guide*
PhEventPeek()  
Check to see if an event is pending

Synopsis:

```c
int PhEventPeek( void *buffer,
                 unsigned size );
```

Library:

ph

Description:

This function lets you check if an event is pending on the current Photon channel:

- When there’s an event pending, Photon replies immediately with that event, and this function returns Ph_EVENT_MSG.
- If no message is available, this function returns 0.

Since this function is nonblocking, you should find it useful for applications that need to run continuously and still interact with Photon.

For asynchronous event notification, see PhEventRead() and PhEventArm().

If your application uses widgets, don’t try to write your own event-handling loop; use PtMainLoop() or PtProcessEvent() instead.

Returns:

Ph_EVENT_MSG

Successful completion.

Ph_RESIZE_MSG

The Ph_DYNAMIC_BUFFER flag was set in PhAttach(), and there’s a pending Photon event that’s larger than the client’s event buffer. This event that indicates how large the client’s buffer needs to be to receive the entire event message.

0

No message was available.
PhEventPeek()  

Examples:

```c
#define EVENT_SIZE sizeof( PhEvent_t ) + 1000

main( int argc, char *argv[] )
{
    int go = 1, count = 0;
    PhEvent_t *event;

    if( initialize() == -1 )
        exit( EXIT_FAILURE );

    if( NULL == ( event = malloc( EVENT_SIZE ) ) )
        exit( EXIT_FAILURE );

    while( go ) {
        if(( ++count & 15) == 0) {
            PgFlush();
            switch( PhEventPeek( event, EVENT_SIZE ) ) {
                case Ph_EVENT_MSG:
                    PtEventHandler( event );
                    break;
                case -1:
                    perror( "PhEventPeek failed" );
                    break;
            }
        }
        iterate_graphics_process();
    }
    exit( 0 );
}
```

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PhEventPeek()

See also:

PhAttach(), PhEvent_t, PhEventNext(), PhEventRead(), PtEventHandler()

“Collecting events” in the Events chapter of the Photon Programmer’s Guide
**PhEventRead()**  
*Provide asynchronous event notification*

**Synopsis:**

```c
int PhEventRead ( int rcvid,  
                 void *buffer,  
                 unsigned size );
```

**Arguments:**

- `rcvid` The receive ID returned by `MsgReceive()` (see the QNX Neutrino Library Reference).
- `buffer` A pointer to the message received. If the message is a pulse telling you that there’s a Photon event waiting, `PhEventRead()` stores the event in this buffer. This buffer must be large enough to hold a Photon event and its associated rectangles and data.
- `size` The size of the buffer, in bytes.

**Library:**

`ph`

**Description:**

This function provides an asynchronous event-notification mechanism. You’ll find it useful for applications that need to interact with Photon but also need to collect Neutrino pulses or messages from other processes.

For synchronous event notification, see `PhEventNext()` and `PhEventPeek()`.

The widget library calls `PhEventRead()` internally; if you call `PhEventRead()` in an application that uses widgets, you might get unexpected results. Use `PtMainLoop()` or `PtProcessEvent()` instead.

Typically, you call this function with the `rcvid` returned by `MsgReceive()` to see if the message received was a pulse sent by Photon.
If the message received is the application’s event pulse, then:

- If the buffer is large enough for the event, `PhEventRead()` retrieves the event, stores it in the location pointed to by `buffer`, and rearms Photon. `PhEventRead()` returns `Ph_EVENT_MSG`.

- If the buffer is too small for the event, but you set `Ph_DYNAMIC_BUFFER` when you called `PhAttach()`, `PhEventRead()` stores a resize event in the buffer and returns `Ph_RESIZE_MSG`; see `PhGetMsgSize()`.

- If the buffer is too small for the event, and you didn’t set `Ph_DYNAMIC_BUFFER` when you called `PhAttach()`, `PhEventRead()` sets `errno` to `EMSGSIZE` and returns -1.

If the message isn’t the application’s event pulse, `PhEventRead()` returns 0.

Photon may close a region (for example, via the window manager) before you read the pending event. As a result, `PhEventRead()` may indicate that no event is pending even though you were notified otherwise. In this case, `PhEventRead()` returns -1 and sets `errno` to `ENOMSG`.

You must call `PhAttach()` and arm the event pulse by calling `PhEventArm()` before you call `PhEventRead()` for the first time.

**Returns:**

- **Ph_EVENT_MSG**
  
  Successful completion.

- **Ph_RESIZE_MSG**

  The `Ph_DYNAMIC_BUFFER` flag was set in `PhAttach()`, and there’s a pending Photon event that’s larger than the client’s event buffer. This event indicates how large the client’s buffer needs to be to receive the entire event message.

- **0**

  A non-Photon message was available.
An error occurred.

Examples:

This code fragment shows how you can use `PhEventRead()` with `PhGetMsgSize()` to maintain a dynamic event buffer. You need to define `my_msg_struct`, `initialize()`, and `process_app_msg()`:

```c
#include <stdlib.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <Pt.h>

int initialize( void );
void process_app_msg( void * );

int main( int argc, char *argv[] )
{
    int rcvid, chid;
    void *msg;
    unsigned msg_size;

    if( initialize() == -1 )
        exit( EXIT_FAILURE );

    msg_size = sizeof (my_msg_struct);
    if ( !(msg = malloc( msg_size )))
    {
        errno = ENOMEM;
        return( EXIT_FAILURE );
    }

    PhEventArm();
    chid = PhChannelAttach( 0, -1, NULL );

    while( 1 )
    {
        rcvid = MsgReceive( chid, msg, msg_size, NULL );

        switch( PhEventRead( rcvid, msg, msg_size) )
        {
        case Ph_EVENT_MSG:
            PtEventHandler( (PhEvent_t *)msg );
            break;

        case Ph_RESIZE_MSG:
            msg_size = PhGetMsgSize( (PhEvent_t *)msg );
            if ( !( msg = realloc( msg, msg_size )))
            {
                errno = ENOMEM;
            }
        
```
PhEventRead()

```c
    return( EXIT_FAILURE );
}
break;

case 0:
    process_app_msg( msg );
    break;

case -1:
    perror( "PhEventRead failed" );
    break;
}
}
return( EXIT_SUCCESS );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PhAttach(), PhEvent_t, PhEventArm(), PhEventNext(), PhEventPeek(), PhGetMsgSize(), PtEventHandler()

“Collecting events” in the Events chapter of the Photon Programmer’s Guide

MsgReceive() in the QNX Neutrino Library Reference
**PhEventRegion_t**

*Data structure describing the emitter and collector of an event*

**Synopsis:**

```c
typedef struct Ph_event_region {
    PhRid_t    rid;
    long       handle;
} PhEventRegion_t;
```

**Description:**

The `PhEventRegion_t` structure describes the emitter and the collector of events. It contains at least the following members:

- **PhRid_t rid**
  
  The ID of a region. This member lets an application determine which of its regions emitted or collected an event.

  The following constants are defined in `<photon/PhT.h>`:

  - `Ph_DEV_RID` — the ID of the device region.
  - `Ph_ROOT_RID` — the ID of the root region.

- **long handle**

  The user-definable handle that the application specifies when it opens the region. Applications can use `handle` to quickly pass a small amount of information along with events. For example, the widget (Pt*()) functions use `handle` internally.

  If the region described by a `PhEventRegion_t` structure isn’t owned by the application that collected the event, then the Photon Manager sets `handle` to 0.

**Classification:**

Photon
PhEventRegion_t

See also:

PhEvent_t, PhRegion_t

Events chapter of the Photon *Programmer’s Guide.*
**PhFindTransportType()**

*Find a transport type in the transport registry*

**Synopsis:**

```c
PhTransportRegEntry_t * PhFindTransportType( char *packing_type );
```

**Library:**

ph

**Description:**

`PhFindTransportType()` finds the transport type matching `packing_type` within the transport registry.

**Returns:**

A pointer to a `PhTransportRegEntry_t` structure that describes the requested `packing_type`, which can be used in functions (such as `PhPackEntry()`) that require a `PhTransportRegEntry_t` as a parameter, or NULL if no entry for `packing_type` was found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhFreeTransportType()`, `PhMallocUnpack()`, `PhPackEntry()`, `PhRegisterTransportType()`, `PhTransportRegEntry_t`, `PhTransportType()`, `PhUnpack()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
PhFreeTiles()
Return a list of tiles to the internal tile pool

Synopsis:

```c
void PhFreeTiles( PhTile_t * tile );
```

Library:

ph

Description:

This function returns the given list of tiles to the internal tile pool.

Photon maintains an internal pool of tiles because they’re frequently used, and using a pool reduces the amount of time spent allocating and freeing the tiles. Use PhGetTile() to get a tile from the pool, and this function to return a list of tiles to the pool. Don’t free() a PhTile_t structure.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhAddMergeTiles(), PhClipTilings(), PhCoalesceTiles(), PhCopyTiles(), PhDeTranslateTiles(), PhGetTile(), PhIntersectTilings(), PhMergeTiles(), PhRectsToTiles(), PhSortTiles(), PhTile_t, PhTilesToRects(), PhTranslateTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide

Chapter 9 • Ph—Photon 901
**PhFreeTransportType()**

Free data associated with a transport registry entry

**Synopsis:**

```c
int PhFreeTransportType( void *data,                
                         char *packing_type );
```

**Library:**

`ph`

**Description:**

This function assumes that the data pointed to by `data` is an allocated structure that’s described by a transport registry entry for `packing_type` — for more information, see `PhTransportRegEntry_t`.

All data associated with `data` is freed using the transport registry entry for `packing_type`.

**Returns:**

- 0 Success.
- -1 A type matching `packing_type` wasn’t found.

**Examples:**

```
PhFreeTransportType( ptr, "PhImage" );
PhFreeTransportType( ptr, "files" );
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

902 Chapter 9 • Ph—Photon
PhFreeTransportType()  

See also:  

PhFindTransportType(), PhMallocUnpack(), PhPackEntry(),  
PhRegisterTransportType(), PhTransportRegEntry_t,  
PhTransportType(), PhUnpack()  

Drag and Drop chapter of the Photon Programmer’s Guide
**PhGetAllTransportHdtrs()**

*Extract all the headers from a buffer of packed transport data*

**Synopsis:**

```c
PhTransportHdr_t * PhGetAllTransportHdtrs(
    char *buffer,
    int unsigned buffer_size);
```

**Library:**

`ph`

**Description:**

`PhGetAllTransportHdtrs()` extracts all the headers from the given buffer of packed transport data and puts them into a linked list. The `buffer_size` argument specifies the size of the buffer, in bytes.

**Returns:**

A pointer to the first entry in the linked list of transport headers.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhGetNextTransportHdr()`, `PhGetTransportHdr()`, `PhLocateTransHdr()`, `PhReleaseTransportHdtrs()`, `PhUnlinkTransportHdr()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PhGetConnectId()**

*Get the connection ID of the calling process*

**Synopsis:**

```c
PhConnectId_t PhGetConnectId( void );
```

**Library:**

`ph`

**Description:**

This function returns the connection ID of the calling process.

**Returns:**

The connection ID of the calling process, or -1 on error.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhGetConnectInfo()`
PhGetConnectInfo()
Get information about a Photon channel

Synopsis:

PhConnectId_t PhGetConnectInfo(
    PhConnectId_t coid,
    PhConnectInfo_t *buf);

Library:

ph

Description:

This function fills *buf with information about the specified Photon channel. If coid is zero, information about the calling process is returned. If it isn’t zero but doesn’t match any existing channel, then the next monotonically greater channel ID is used. If coid is greater than any existing channel ID, -1 is returned and errno is set to ESRCH.

The PhConnectInfo_t structure includes at least the following members:

- **unsigned long flags** — a combination of flags that describes the channel:
  - Ph_PROC_VIRTUAL — the process is the Photon server
  - Ph_PROC_BLOCKED — the process is "Reply-blocked" on a server (for example, pwm)
  - Ph_PROC_HELD — the process is held on someone else’s queue

- **PhChannelParms_t parms** — the channel’s parameters; see PhAttach()

- **PhConnectId_t block** — if Ph_PROC_BLOCKED is set, this is the ID of the server. Otherwise, if Ph_PROC_HELD is set, it’s the ID of one of the apps whose event queue this process is trying to overflow. (It’s possible for both these flags to be set if the application is multithreaded.)

- **unsigned num_q_entries** — the number of events in the queue
PhGetConnectInfo()

- `unsigned buf_len` — the size of the application’s event buffer
- `PhConnectId_t id` — the connector ID
- `int nid` — the node descriptor
- `pid_t pid` — the process ID
- `unsigned long chev_sense` — the application’s sensitivity to channel events

**Returns:**

The channel ID (the same as `buf->id`), or -1 if the call fails.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhGetConnectId()
PhGetData()
Get data for an event

Synopsis:

```c
void *PhGetData( PhEvent_t const *event );
```

Library:

ph

Description:

This function returns a pointer to an event-specific data structure. For a complete description of the data structures returned with each event type, see the description of the `PhEvent_t` structure.

You can determine the size of the data from `event->data_len`.

Returns:

A pointer to an event-specific data structure.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhEvent_t, PhGetRects()

Events chapter of the Photon Programmer’s Guide
**PhGetMsgSize()**

*Get message size*

**Synopsis:**

```c
unsigned PhGetMsgSize( PhEvent_t const *event_buf );
```

**Library:**

`ph`

**Description:**

This function returns the size of buffer necessary to contain the event that couldn’t fit into the current buffer. You typically use this function after `PhEventRead()`, `PhEventPeek()`, or `PhEventNext()` has returned a value of `Ph_RESIZE_MSG`.

These functions can return `Ph_RESIZE_MSG` only if the application set the `Ph_DYNAMIC_BUFFER` flag when it attached the Photon channels; see `PhAttach()`.

**Returns:**

The size of the buffer required to accommodate the entire event.

**Examples:**

See `PhEventRead()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 9 • Ph—Photon  909
PhGetMsgSize()

See also:

PhAttach(), PhEvent_t, PhEventNext(), PhEventPeek(), PhEventRead()
**PhGetNextInlineData()**

Get the data for the next entry in a linked list of transport data

**Synopsis:**

```c
void * PhGetNextInlineData(
    PhTransportCtrl_t *ctrl,
    PhTransportLink_t *current,
    PhTransportLink_t **new_link);
```

**Library:**

ph

**Description:**

`PhGetNextInlineData()` gets the inline data for the next entry in the linked list belonging to the control structure pointed to by `ctrl`. The `current` argument is a pointer to the `PhTransportLink_t` structure for the current entry in the list, or NULL to start at the beginning of the list. The memory pointed to by `new_link` is set to the address of the next entry, or NULL if the end of the list has been reached.

**Returns:**

A pointer to the data stored in the next entry of the list, or NULL if the end of the list has been reached.

**Classification:**

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No
**PhGetNextInlineData()**

See also:

PhCreateTransportCtrl(), PhReleaseTransportCtrl(),
PhTransportCtrl_t, PhTransportLink_t

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PhGetNextTransportHdr()**

*Get the next header from a buffer of packed transport data*

**Synopsis:**

```c
PhTransportHdr_t * PhGetNextTransportHdr(
    PhTransportHdr_t *last_hdr,
    PhTransportHdr_t *trans_hdr);
```

**Library:**

`ph`

**Description:**

This function gets the next header out of a buffer of packed transport data. This header contains information needed to unpack the data that follows the header.

The `last_hdr` argument points to the last transport header that was unpacked, or to the beginning of the buffer. The information for the next header found is stored in the structure pointed to by `trans_hdr`.

**Returns:**

The same pointer as `trans_hdr`, or NULL if there isn’t another header in the buffer.

**Errors:**

- **EINVAL**  
  The structure pointed to by `last_hdr` has a NULL buffer pointer.

- **ENOENT**  
  The size of the buffer in the structure pointed to by `trans_hdr` is 0, or there’s no entry for the header in the transport registry.

**Classification:**

Photon
**PhGetNextTransportHdr()**

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

- `PhGetAllTransportHdrs()`, `PhGetTransportHdr()`,
- `PhLocateTransHdr()`, `PhReleaseTransportHdrs()`,
- `PhUnlinkTransportHdr()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
PhGetRects()
Get an event’s rectangle set

Synopsis:

PhRect_t *PhGetRects( PhEvent_t const *event );

Library:

ph

Description:

This function gets the rectangle set associated with the specified event. The number of rectangles in the event’s rectangle set is given by event->num_rects.

In the case of an expose event, this function returns a pointer to the list of rectangles that need to be repaired. In the case of a pointer event, only one rectangle is associated with the event — the one associated with the current position of the pointer.

For more information on the meaning of the rectangle set for different event types, see the event types described for the PhEvent_t structure.

Returns:

A pointer to the rectangles associated with the event.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PhGetRects()

See also:

PhEvent_t, PhGetData(), PhRect_t

Events chapter of the Photon *Programmer’s Guide*
**PhGetTile()**

*Retrieve a tile from the internal tile pool*

**Synopsis:**

```c
PhTile_t *PhGetTile( void );
```

**Library:**

`ph`

**Description:**

Photon maintains an internal pool of tiles because tiles are frequently used, and using the pool reduces the amount of time allocating and freeing the tiles. This function retrieves a tile from the internal tile pool. The tile isn’t initialized.

Don't `free()` the tile; instead, use `PhFreeTiles()` to return the tile to the pool.

**Returns:**

A pointer to the new tile, or NULL if an error occurred.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`, `PhCopyTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`,
PhGetTile()

PhIntersectTilings(), PhMergeTiles(), PhRectsToTiles(),
PhSortTiles(), PhTile_t, PhTilesToRects(), PhTranslateTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of
the Photon Programmer’s Guide
**PhGetTransportHdr()**
*Extract the header from a buffer of packed transport data*

**Synopsis:**

```c
PhTransportHdr_t *PhGetTransportHdr(
    PhTransportHdr_t *hdr,
    char *buffer);
```

**Library:**

ph

**Description:**

This function extracts the header from the given buffer of packed transport data, storing the header in the structure pointed to by `hdr`. `PhGetTransportHdr()` is used by the recipient of the transport operation to get instructions for unpacking the data that follows the header in the buffer.

**Returns:**

The same pointer as `hdr`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhGetAllTransportHdrs()`, `PhGetNextTransportHdr()`, `PhLocateTransHdr()`, `PhReleaseTransportHdrs()`, `PhUnlinkTransportHdr()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PhGetTransportVectors()**

Build an I/O vector of data to be transported

**Synopsis:**

```c
iov_t * PhGetTransportVectors(
    PhTransportCtrl_t *trans_ctrl,
    int num_hdr_vectors,
    int unsigned *num_vectors,
    int unsigned *size );
```

**Library:**

`ph`

**Description:**

This function builds an array of I/O vectors from the linked list of packed data belonging to the transport-control structure pointed to by `trans_ctrl`.

The `num_hdr_vectors` argument specifies the number of vectors to reserve at the beginning of the array. These entries can be used for headers suitable for sending to the destination of the transport operation.

The variable pointed to by `num_vectors` is set to the number of entries in the array, including the reserved ones. The variable pointed to by `size` is set to the size of the array, excluding the size of the reserved entries.

**Returns:**

A pointer to the array of I/O vectors, or NULL if there wasn’t enough memory to allocate it.

**Classification:**

Photon

**Safety**

Interrupt handler No

continued...
PhGetTransportVectors()

Safety

<table>
<thead>
<tr>
<th>Signal handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhCreateTransportCtrl(), PhGetNextInlineData(), PhPackEntry(), PhPackType(), PhTransportCtrl_t, PhTransportLink_t, PhTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
PhImage_t
Data and characteristics of an image

Synopsis:

typedef struct PhImage
{
    int               type;
    ulong_t           image_tag;
    int               bpl;
    PhDim_t           size;
    ulong_t           palette_tag;
    int               colors;
    PgAlpha_t         *alpha;
    PgColor_t         transparent;
    char              format;
    char              flags;
    char              ghost_bpl;
    char              spare1;
    char              *ghost_bitmap;
    int               mask_bpl;
    char              *mask_bm;
    PgColor_t         *palette;
    char              *image;
} PhImage_t;

Description:

The PhImage_t structure describes the data and characteristics of an image. When you give an image to a PtLabel subclass widget, this is the structure you must provide. To get a pointer to this structure, use PhCreateImage() or PxLoadImage(). To free the allocated members of this structure, call PhReleaseImage().

The structure contains at least the following members:

int type The graphic type; see “Image types,” below.

ulong_t image_tag

The image-data tag, a cyclic redundancy check (CRC) that’s used extensively by phrelay (see the QNX Neutrino Utilities Reference) to cache images.
This tag is filled in automatically for images created in PhAB or by `PxLoadImage()`. If you’re creating an image in some other way, you should fill in the tag by calling `PtCRC()`, passing it the image pixel data and the size of that data.

**int bpl**
The number of bytes in each line of the image.

**PhDim_t size**
A `PhDim_t` structure that defines the size of the image.

**ulong_t palette_tag**
The palette-data tag.

**int colors**
The number of colors in the image.

**PgAlpha_t *alpha**
The image alpha map that’s used if the source alpha map option is enabled.

**PgColor_t transparent**
The color to mask out when drawing.

**char format**
Not used.

**char flags**
The image flags. The valid bits are:

- `Ph_RELEASE_IMAGE`—free the image data.
- `Ph_RELEASE_PALETTE`—free the palette data.
- `Ph_RELEASE_TRANSPARENCY_MASK`—free the transparency mask bitmap.
- `Ph_RELEASE_GHOST_BITMAP`—free the bitmap used for ghosting.
- `Ph_RELEASE_IMAGE_ALL`—free all the above.
- `Ph_USE_TRANSPARENCY` — make the image transparent by using the color specified by the `transparent` member as the key for a chroma operation.
**PhImage_t**

A widget automatically frees the memory pointed to by the `PhImage_t` members if these bits are set. Calling `PhReleaseImage()` with an image frees any resources that have the corresponding bit set in the image flags.

```c
char ghost_bpl
```

The number of bytes per line for the ghosting bitmap.

```c
char *ghost_bitmap
```

A pointer to the transparency mask for ghosting an image. The leftmost pixel corresponds to the top bit of the first byte in the mask.

```c
int mask_bpl
```

The number of bytes per line for the transparency mask.

```c
char *mask_bm
```

A pointer to the transparency mask. The leftmost pixel corresponds to the top bit of the first byte in the mask.

```c
PgColor_t *palette
```

The image palette.

```c
char *image
```

The image pixel data.

**Image types**

The types of image are:

```c
Pg_BITMAP_BACKFILL
```

A bitonal image; the two colors are specified in the color palette.

```c
Pg_BITMAP_TRANSPARENT
```

A monochrome image with transparent regions. The bits in the image data that are set to 1 are drawn using color palette entry 1; zeros are treated as transparent, so they're not drawn.
Pg_IMAGE_DIRECT_1555

This format has, for each pixel, one bit of alpha, and 5 bits each of red, green, and blue.

Pg_IMAGE_DIRECT_4444

This format has 4 bits of alpha, and 4 bits each of red, green, and blue.

Pg_IMAGE_DIRECT_444

This format requires 2 bytes per pixel. It matches the high-speed color lookup tables used by palette-based graphics drivers and provides the fastest method of drawing direct-color images with palette-based graphics drivers. Here’s the bit order:

xxxx.RRRR.GGGG.BBBB

Pg_IMAGEDIRECT_555

This format packs each pixel into 2 bytes. Although it allows only 32 levels of each color, this format provides reasonable image reproduction with less data. Here’s the bit order:

xRRR.RRGG.GGGB.BBBB

Pg_IMAGE_DIRECT_565

This format packs each pixel into 2 bytes and matches the display format of most 16-bit, direct-color, graphics drivers. Here’s the bit order:

RRRR.RGGR.GGGB.BBBB

Pg_IMAGE_DIRECT_8888

This format is an array of 4-byte color entries. The least significant byte is the blue component, after that there is the green component, following that there is the red component and the most significant byte is reserved.
This format packs each pixel into 3 bytes. Using this format, you can represent a full 24 bit color image. Here’s the bit order:

```
RRRR.RRRR.GGGG.GGGG.BBBB.BBBB
```

This format uses 1 byte per pixel. The colors are algorithmically generated as a gradient between the color set by `PgSetFillColor()` and the color set by `PgSetTextColor()`. A pixel value of 0 produces the fill color, a pixel value of 255 produces the text color, and a pixel value of 128 produces an even blend of the two.

This format packs 2 pixels per byte, allowing up to 16 levels. The first pixel is in the upper half of the byte and the second pixel is in the lower half. The colors are algorithmically generated as a gradient between the color set by `PgSetFillColor()` and the color set by `PgSetTextColor()`. A pixel value of 0 produces the fill color, a pixel value of 15 produces the text color, and a pixel value of 8 produces an even blend of the two.

This format packs 1 pixel per byte, allowing up to 256 colors. This format indexes directly into the current palette; see `PgSetPalette()`. If no palette is set, the function chooses colors from the global palette; this may cause colors to look different on each system.

This format packs 2 pixels per byte, allowing up to 16 colors. The first pixel is in the upper half of the byte, the second is in the lower half. These pixel values index directly into the current palette. If no palette is set, the function chooses colors from the global palette.
PhImage_t

For convenience, you can AND Pg_IMAGE_CLASS_MASK with the image type to determine the image’s class:

Pg_IMAGE_CLASS_PALETTE
   The image requires a palette defined by PgSetPalette().

Pg_IMAGE_CLASS_GRADIENT
   The image requires first and last colors defined by PgSetFillColor() and PgSetTextColor().

Pg_IMAGE_CLASS_DIRECT
   Each pixel defines its red, blue, and green components.

The Pg_BITMAP_BACKFILL and Pg_BITMAP_TRANSPARENT types don’t fit into the “image class” scheme. If you’re checking the class, check the type against these two constants first. If it doesn’t match either of them, check the class as described above.

Classification:
Photon

See also:
ApGetImageRes(), PgColor_t, PgDrawPhImage*(), PgDrawPhImageRect*(), PgDrawRepPhImage*(), PhCreateImage(), PhDim_t, PhMakeGhostBitmap(), PhMakeTransBitmap(), PhMakeTransparent(), PhReleaseImage(), PmMemCreateMC(), PmMemFlush(), PxLoadImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhInitDrag()
Initiate a drag operation

Synopsis:

```c
int PhInitDrag(
    PhRid_t rid,
    unsigned flags,
    const PhRect_t *rect,
    const PhRect_t *boundary,
    unsigned int input_group,
    const PhDim_t *min,
    const PhDim_t *max,
    const PhDim_t *step,
    const PhPoint_t *ptrpos,
    const PhCursorDescription_t *cursor );
```

Arguments:

- **rid**: A `PhRid_t` that specifies the region that initiates the drag.
- **flags**: Determines how the drag operation behaves. Defined values:
  - `Ph_TRACK_LEFT`: Left edge of drag rectangle tracks pointer.
  - `Ph_TRACK_RIGHT`: Right edge of drag rectangle tracks pointer.
  - `Ph_TRACK_TOP`: Top edge of drag rectangle tracks pointer.
  - `Ph_TRACK_BOTTOM`: Bottom edge of drag rectangle tracks pointer.
  - `Ph_TRACK_DRAG`: All edges of drag rectangle track pointer.
  - `Ph_DRAGTRACK`: Emit drag events to track the drag, but don’t rubber-band.
  - `Ph_DRAG_KEY_MOTION`: Emit `Ph_EV_KEY`, `Ph_EV_PTR_MOTION_BUTTON` and
Ph_InitDrag()

Ph_EV_PTR_MOTION_NOBUTTON events with the Ph_EV_DRAG subtype during the drag. These events are normally suppressed during a drag.

**Ph_DRAG_NOBUTTON**

Start the drag even if no buttons are held down.

**rect**

A pointer to a PhRect_t structure that defines the rectangle to be dragged (or “rubber-banded”) on screen. The boundary represents a rectangular constraint area beyond which the edges of rect may not extend. The coordinates in rect are relative to the origin of the region that initiates the drag, specified by rid.

**boundary**

A pointer to a PhRect_t structure that defines a rectangular constraint area for rect. The edges of the rectangle may not exceed this area. The coordinates in boundary are relative to the origin of the region that initiates the drag, specified by rid.

**input_group**

You should set the input_group argument to the input-group value supplied with the event in cbinfo (see example).

**min**

A pointer to a PhDim_t structure that defines the minimum size for the drag rectangle returned in the drag events. (The application receives the events in absolute coordinates.)

**max**

A pointer to a PhDim_t structure that defines the maximum size for the drag rectangle returned in the drag events. (The application receives the events in absolute coordinates.)

**step**

A pointer to a PhDim_t structure that defines the drag granularity. This structure’s width (w) and
PhInitDrag()

height \((h)\) members indicate the distance a dragged object moves between drag events.

\textit{ptrpos}\ A pointer to a \texttt{PhPoint_t} structure that defines the initial cursor position for the drag. Applications should take it from the event that makes them decide to start a drag. If the cursor moves from that position by the time your \texttt{PhInitDrag()} reaches Photon, your drag is updated accordingly.

In other words:

- Photon “virtually” moves the cursor back to the location indicated by \textit{ptrpos}, then starts the drag, and then “virtually” drags the cursor back to where it really is.

Or:

- Photon makes the drag behave as if it started from where you thought the cursor was rather than from where Photon thought it was a few moments later.

\textit{cursor}\ A pointer to a \texttt{PhCursorDescription_t} structure. This is the header member of either a \texttt{PhCharacterCursorDescription_t} or \texttt{PhBitmapCursorDescription_t} that defines how the cursor should look while dragging. This member may be NULL.

\textbf{Library:}\n
\texttt{ph}

\textbf{Description:}\n
This function starts a drag. Normally, when the drag has completed, the application collects a Ph\_EV\_DRAG event that describes the results of the operation. But if the application closes the region that has initiated the drag operation, the operation completes without returning a Ph\_EV\_DRAG event.
**PhInitDrag()**

Any attempt to initiate a drag operation while another is in progress in the same input group fails.

You can cancel a drag operation with *PhCancelDrag()*.

**Returns:**

- **0** Successful completion.
- **-1** An error occurred.

**Examples:**

```c
drag_lower_left( PhRect_t *rect, PhRect_t *boundary,
                 PtCallbackInfo_t *cbinfo )
{
    PhEvent_t *event = cbinfo->event;
    PhPointerEvent_t *ptrev = PhGetData( event );
    static const PhCharacterCursorDescription_t
        cursor = { { Ph_CURSOR_DRAG_BL, sizeof(cursor) }, Pg_RED };

    PhInitDrag( my_region, Ph_TRACK_LEFT | Ph_TRACK_BOTTOM,
                rect, boundary, event->input_group,
                NULL, NULL, NULL, &ptrev->pos, &cursor.hdr );
}

raw_callback( PtWidget_t *widget, void *data,
              PtCallbackInfo_t *cbinfo)
{
    PhRect_t *rect;
    PhDragEvent_t *drag;
    ...

    switch( cbinfo->event->type )
    {
        case Ph_EV_DRAG:
            drag = (PhDragEvent_t *)PhGetData( cbinfo->event );
            rect = &drag->rect;
            // drag rectangle in ABSOLUTE coordinates.
            PhTranslateRect( rect,
                             &cbinfo->event->translation );
            // rect is now relative to the region the drag
            // was initiated on.
            ...
    }
    ...
}
```
**PhInitDrag()**

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PhBitmapCursorDescription_t`
- `PhCharacterCursorDescription_t`
- `PhCancelDrag()
- `PhDragEvent_t`
- `PhEvent_t`
- `PhGetData()
- `PhDim_t`
- `PhPoint_t`
- `PhRect_t`
- `PhTranslateRect()`

“Dragging” in the Events chapter of the Photon *Programmer’s Guide*
**PhInputGroup()**

*Determine the input group associated with an event*

**Synopsis:**

```c
int PhInputGroup( PhEvent_t *event );
```

**Library:**

`ph`

**Description:**

This function determines the input group associated with the given event. If `event` is NULL or its input group is 0, `PhInputGroup()` returns the default input group.

**Returns:**

The input group.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhEvent_t`
**PhIntersectTilings()**

*Determine the intersection of two lists of tiles*

**Synopsis:**

```c
PhTile_t *PhIntersectTilings(
    PhTile_t const * const tile1,
    PhTile_t const * const tile2,
    unsigned short *num_intersect_tiles );
```

**Library:**

`ph`

**Description:**

This function creates a new list of tiles that’s the intersection of the lists pointed to by `tile1` and `tile2`. The original lists aren’t modified.

**Returns:**

A pointer to the new list, or NULL if there’s no intersection.

Don’t `free()` the list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`, `PhCopyTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`, `PhGetTile()`,
"Using damage tiles" in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhKeyEvent_t
Data structure describing a key event

Synopsis:
See below.

Description:
This structure describes a key event. It includes at least these members:

unsigned long key_mods
Some keys (e.g. Shift or Num Lock) modify other keys. When a modifier key is pressed or released, it’s evaluated through a table and the key_mods field is updated accordingly. This evaluation is done before the key event is sent.

The key_mods is a combination of the following bits:

- Pk_KM_Shift
- Pk_KM_Ctrl
- Pk_KM_Alt
- Pk_KM_AltGr
- Pk_KM_Shl3
- Pk_KM_Mod6
- Pk_KM_Mod7
- Pk_KM_Mod8
- Pk_KM_Shift_Lock
- Pk_KM_Ctrl_Lock
- Pk_KM_Alt_Lock
- Pk_KM_AltGr_Lock
- Pk_KM_Shl3_Lock
- Pk_KM_Mod6_Lock
- Pk_KM_Mod7_Lock
- Pk_KM_Mod8_Lock
- Pk_KM_Caps_Lock
PhKeyEvent_t

- Pk_KM_Num_Lock
- Pk_KM_Scroll_Lock

If the Shift key is pressed, the Shift modifier is on; if it's released, the Shift modifier is off. Because some keys occur twice on the keyboard, a key release doesn’t guarantee that the corresponding modifier is off — the matching key may still be pressed.

unsigned long key_flags

Flags that indicate the status of the key:
- Pk_KF_Key_Down — the key has been pressed.
- Pk_KF_Key_Repeat — the key is repeating.
- Pk_KF_Scan_Valid — the key_scan member is valid.
- Pk_KF_Sym_Valid — the key_sym member is valid; this bit is set only on a key press, not a release.
- Pk_KF_Cap_Valid — the key_cap member is valid.
- Pk_KF_Compose — a compose sequence is in progress.

unsigned long key_cap

The unique scan code produced by the key, without any modifiers. This member is valid only if Pk_KF_Cap_Valid is set in the key_flags.

unsigned long key_sym

The value of the key with modifiers applied to it. This member is valid only if Pk_KF_Sym_Valid is set in the key_flags.

This field holds the value that’s used for text entry; it can also be used in a switch statement to determine a key’s function.

unsigned short key_scan

The hardware-dependent scan code for the key. This member is valid only if Pk_KF_Scan_Valid is set in the key_flags.

PhPoint_t pos

A PhPoint_t structure that specifies the current mouse-pointer position.
**PhKeyEvent_t**

```c
unsigned short button_state
```

The current state of the pointing-device buttons (i.e. which buttons are currently pressed):

- `Ph_BUTTON_SELECT`
- `Ph_BUTTON_MENU`
- `Ph_BUTTON_ADJUST`

All flags and key symbols are defined in `<photon/PkKeyDef.h>`.

Before using the `key_cap`, `key_scan`, or `key_sym` members, check the `key_flags` to make sure they’re valid. The `key_cap` identifies the key that caused the event, while `key_sym` defines the character (or function key code) that the event carries, if any.

The keyboard is divided into groups, as dictated by ISO 9995. When a key in the text group is pressed and the Ctrl or Alt modifier is on, the keyboard driver doesn’t generate a `key_sym`. If the key is in any other key group, the driver generates a `key_sym`.

For any key press event, there’s a corresponding release event. For example, if you press the A key, `key_cap` is set to a in both the press and release (and any repeats), but only the press and repeats have a valid `key_sym`. Its value may be a, A, or perhaps an accented character or some symbol, depending on whether or not this keystroke completed a compose sequence.

If you’re looking for printable characters (i.e. textual input as opposed to “raw” key presses and releases), look at `key_sym` (after throwing away any events that don’t have the `Pk_KF_Sym_Valid` flag) and ignore the modifiers. Also, ignore symbols in the 0xF0XX range; those are nonprintable control characters, such as Home, PageUp, and function keys.

If you’re looking for cursor keys or function keys and don’t care about the difference between the two PageDown keys or about interpreting the NumLock flag yourself, also look at `key_sym` (again, after ignoring events that don’t have a valid symbol). Look at the modifiers if you want to recognize combinations such as Shift-Home or Ctrl-PageUp.
**PhKeyEvent_t**

In the rare cases where you need to distinguish between the two PageDown keys (Pk_Pg_Down and Pk_KP_3), look at `key_cap` (discarding any events that don’t have the Pk_KF_Cap_Valid flag). But beware: if an event contains the symbol 3, it’s probably wiser to assume that the person meant the number 3 rather than PageDown. Since key releases normally contain a valid cap, you’ll also need to look at the Pk_KF_Key_Down flag to distinguish between presses and releases (and possibly the Pk_KF_Key_Repeat flag if you want to distinguish between presses and repeats).

If you need to detect keystrokes such as Ctrl-A or Alt-B, you have no choice; those normally don’t carry a symbol, and you need to look at the `key_cap` and the other flags. Actually, if an Alt-B keystroke does carry a symbol, it’s probably safer to assume that it wasn’t meant to be also recognized as an Alt-B — there might be keyboard mappings that map Alt-B to some special symbol that has nothing to do with Alt-B.

**Classification:**

Photon

**See also:**

PhEvent_t, PhPoint_t, PhPointerEvent_t

Events chapter of the Photon *Programmer’s Guide*
**PhKeyToMb()**

*Get the UTF-8 value of a key event*

**Synopsis:**

```c
int PhKeyToMb( char *buffer,
               PhKeyEvent_t const *keyevent );
```

**Library:**

`ph`

**Description:**

This function stores, in `buffer`, a valid UTF-8 char array for the given key event, if one exists.

**Returns:**

The number of bytes in the UTF-8 code, or -1 if there’s no valid UTF-8 code for the given event.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhGetData()`, `PhKeyEvent_t`, `PhTo8859_1()`

`<photon/PkKeyDef.h>`
PhLibVersion()
Get the version number of the Photon libraries

Synopsis:

unsigned PhLibVersion( void );

Library:

ph

Description:

This function returns the version of the Photon library that the calling
application is using.

You can use Ph_LIB_VERSION (defined in <PhT.h>) to determine
the Photon library version when you’re compiling.

Returns:

The version number of the Photon libraries, expressed as:

major version * 100 + minor version

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

“Photon libraries” in the Introduction chapter of the Photon
Programmer’s Guide
PhLinkTransportData()
Add transport data to a linked list

Synopsis:

PhTransportLink_t *
PhLinkTransportData(
    PhTransportLink_t **first_link,
    PhTransportLink_t **last_link,
    char const * const data,
    int unsigned const size,
    iov_t *iovs,
    int unsigned const niovs );

Library:

ph

Description:

This function allocates a PhTransportLink_t structure and fills it in using the data, size, iovs, and niovs arguments. It then adds the new structure to the end of the linked list whose head is pointed to by first_link and whose tail is pointed to by last_link.

This function is used to build the linked list of data in a PhTransportCtrl_t structure.

Returns:

A pointer to the PhTransportLink_t structure, or NULL if there wasn’t enough memory.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

942  Chapter 9  Ph—Photon
PhLinkTransportData()

See also:

PhCreateTransportCtrl(), PhTransportCtrl_t,
PhTransportFindLink(), PhTransportLink_t, PhTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
PhLocateTransHdr()

Look for specific data in a linked list of transport headers

Synopsis:

```c
int PhLocateTransHdr( PhTransportHdr_t **hdr_list,
                       PhTransLoc_t *desired_data,
                       PhTransportHdr_t *found );
```

Library:

ph

Description:

`PhLocateTransHdr()` searches the list of transport headers pointed to by `*hdr_list` for the specific data of the type and description specified by `desired_data`.

If the function finds the data, it sets `found` to point to the header. It also removes any headers for data in the same group as the data found (i.e. the other forms or choices of the data).

Returns:

0 The data wasn’t found.

1 The data was found.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
PhLocateTransHdr()

See also:

PhGetAllTransportHdrs(), PhGetNextTransportHdr(), PhGetTransportHdr(), PhReleaseTransportHdrs(), PhUnlinkTransportHdr()

Drag and Drop chapter of the Photon Programmer’s Guide
**PhMakeGhostBitmap()**

Create a ghost bitmap for an image

**Synopsis:**

```c
int PhMakeGhostBitmap( PhImage_t *image );
```

**Library:**

`ph`

**Description:**

This function takes a `PhImage_t` pointer and creates a ghost bitmap for the image. The ghost bitmap is stored in the image’s data structure.

The `image` argument must point to a valid `PhImage_t` structure. It can point to a regular or transparent image.

The ghost image is used when either Pt_GHOST or Pg_GHOST is passed as a flag to `PgDrawPhImage()` or `PgDrawPhImageMX()`.

**Returns:**

- 0  The image was successfully created.

- -1  The image wasn’t created. The `image` parameter may have been NULL, or the allocation of the bitmap may have failed.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

`PgDrawPhImage*()`, `PgDrawPhImageRect*()`,  
`PgDrawRepPhImage*()`, `PhCreateImage()`, `PhImage_t`  

“Images” in the Raw Drawing and Animation chapter of the Photon

*Programmer’s Guide*
**PhMakeTransBitmap()**

Create a transparency mask for an image

**Synopsis:**

```c
int PhMakeTransBitmap( PhImage_t *image,
                        PgColor_t trans_color );
```

**Library:**

ph

**Description:**

This function creates a transparent bitmap or transparency mask for the given image, provided there isn’t one already.

PhMakeTransparent() is similar to PhMakeTransBitmap(), but PhMakeTransparent() uses chroma when possible; chroma is accelerated by most hardware, whereas transparency bitmaps are always implemented in software.

The meaning of the `trans_color` argument depends on the type of the image:

- **Pg_IMAGE_PALETTE_NIBBLE**
  - The `trans_color` argument is the color in the image’s palette to be made transparent. If more than one entry in the palette contains this color, the first one found is used. You can pass an index into the palette as `trans_color` by ORing it with `Pg_INDEX_COLOR`. For example:
    ```c
    if ( PhMakeTransBitmap( my_image,
                             n | Pg_INDEX_COLOR ) == 0 )
    {
        ...
    }
    ```

- **Pg_IMAGE_PALETTE_BYTE**
  - The `trans_color` argument is the grey index (0-255 for BYTE, 0-15 for NIBBLE) to be made transparent.
PhMakeTransBitmap()

Pg_IMAGE_DIRECT_1555
Pg_IMAGE_DIRECT_4444
Pg_IMAGE_DIRECT_8888
Pg_IMAGE_DIRECT_888

The \textit{trans}\_\textit{color} argument is the color to be made transparent, expressed as a \texttt{PgColor\_t}.

Pg_IMAGE_DIRECT_555
Pg_IMAGE_DIRECT_565
Pg_IMAGE_DIRECT_444

The \textit{trans}\_\textit{color} argument is interpreted as a \texttt{short} packed with the color information in the appropriate format (see \texttt{PhImage\_t}).

The resulting bitmap is stored in the \textit{mask}\_\textit{bm} member of the \texttt{PhImage\_t} structure. This function sets the image’s \texttt{Ph\_RELEASE\_TRANSPARENCY\_MASK} flag.

To draw the image using the transparency mask, use \texttt{PgDrawPhImage()} or \texttt{PgDrawPhImagemx()}.

\textbf{Returns:}

\begin{itemize}
  \item 0  \hspace{1em} \text{Success.}
  \item -1  \hspace{1em} \text{An error occurred.}
\end{itemize}

\textbf{Examples:}

\begin{verbatim}
/*
 * This is code for a PhAB application that demonstrates
 * how to make a transparency mask for an image. This
 * also shows how to take that image and to put it into
 * a label widget and to draw it into a PtRaw's canvas.
 */

/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

Chapter 9 • Ph—Photon 949

PhMakeTransBitmap()
/* Toolkit headers */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Local headers */
#include "abimport.h"
#include "proto.h"

ApDBase_t *database;
PhImage_t trans_image;

/* Setup function for the base window */
int
base_window_setup( PtWidget_t *link_instance,
   ApInfo_t *apinfo,
   PtCallbackInfo_t *cbinfo )
{
   PhImage_t *imgptr;

   // Get the original image from an image-type label widget that we’ve put in a PhAB picture module - don’t close the database since we’ll still be using its image and palette data
   database = ApOpenDBase( ABM_our_picture_module );
   imgptr = ApGetImageRes( database, "image_label" );

   // Copy it so that we don’t change the original // PhImage_t; we’ll still be using the same image // and palette data though
   memcpy( &trans_image, imgptr, sizeof(PhImage_t) );

   // all white pixels will be transparent
   PhMakeTransBitmap( &trans_image, Pg_WHITE );

   // Put the image that contains the transparency mask // into another image-type label
   PtSetResource( ABW_destination_label,
      Pt_ARG_LABEL_IMAGE, &trans_image, 0 );

   /* eliminate ‘unreferenced’ warnings */
   link_instance = link_instance, apinfo = apinfo;
   cbinfo = cbinfo;
PhMakeTransBitmap()

    return( Pt_CONTINUE );
}

/*
 * Draw function (Pt_ARG_RAW_DRAW_F) for a PtRaw widget
 */
void
raw_draw_f( PtWidget_t *widget, PhTile_t *damage )
{
    PhPoint_t    pos = {0, 0};
    PhRect_t     rect;

damage = damage;
PtSuperClassDraw( PtBasic, widget, damage );

    // Find our canvas
PtCalcCanvas( widget, &rect );

    // Set translation so that drawing is relative to
    // the PtRaw widget, not its parent.
PgSetTranslation( &rect.ul, Pg_RELATIVE );

    // Clip to our basic canvas (it’s only polite).
PtClipAdd( widget, &rect );

    // Do our drawing...
PgDrawPhImagemx( &pos, &trans_image, 0 );

    // Remove our translation and clipping
rect.ul.x *= -1; // subtract what we added above
rect.ul.y *= -1;
PgSetTranslation( &rect.ul, Pg_RELATIVE );
PtClipRemove();
}

Classification:

Photon

Safety

Interrupt handler   No
Signal handler      No

continued...
**PhMakeTransBitmap()**

**Safety**

| Thread | No |

See also:

*PgColor_t, PgDrawPhImage*, *PgDrawPhImageRect*,

*PgDrawRepPhImage*, *PhCreateImage*, *PhImage_t,

*PhMakeTransparent*

“Images” in the Raw Drawing and Animation chapter of the Photon

*Programmer’s Guide*
PhMakeTransparent()

Make a given color transparent in an image, using chroma if possible

Synopsis:

```c
int PhMakeTransparent( PhImage_t *image, PgColor_t trans_color );
```

Library:

ph

Description:

PhMakeTransparent() makes the given trans_color transparent in the given image. This function is similar to PhMakeTransBitmap(), but PhMakeTransparent() uses chroma when possible; chroma is accelerated by most hardware, whereas transparency bitmaps are always implemented in software.

If the image is palette-based, and the specified color appears more than once in the palette, both become transparent if chroma is used. In this case, PhMakeTransparent() calls PhMakeTransBitmap() to create a transparency mask.

The trans_color argument is the RGB color in the image’s palette to be made transparent. You can pass an index into the palette as trans_color by ORing it with Pg_INDEX_COLOR. For example:

```c
if ( PhMakeTransparent( my_image, 
    n | Pg_INDEX_COLOR ) == 0 )
{
    ... 
}
```

To draw the image, use PgDrawPhImage() or PgDrawPhImage() mac().

Returns:

- 0  Success.
- -1  An error occurred.
**Examples:**

```c
/*
 * This is code for a PhAB application that demonstrates
 * how to make an image transparent. It also shows how
 * to take that image, put it into a label widget, and
 * draw it on a PtRaw's canvas.
 */

/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Toolkit headers */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Local headers */
#include "abimport.h"
#include "proto.h"

ApDBase_t *database;
PhImage_t trans_image;

/*
 * Setup function for the base window
 */
int base_window_setup( PtWidget_t *link_instance,
                      ApInfo_t *apinfo,
                      PtCallbackInfo_t *cbinfo )
{
    PhImage_t *imgptr;

    /* Get the original image from an image-type
     * label widget that we've put in a PhAB picture
     * module - don't close the database since we'll
     * still be using its image and palette data. */
    database = ApOpenDBase( ABM_our_picture_module );
    imgptr = ApGetImageRes( database, "image_label" );

    /* Copy it so that we don't change the original
     * PhImage_t; we'll still be using the same image
     * and palette data though. */
    memcpy( &trans_image, imgptr, sizeof(PhImage_t) );
```

954  Chapter 9 • Ph—Photon
PhMakeTransparent()

/* Make all the white pixels transparent. */
PhMakeTransparent( &trans_image, Pg_WHITE );

/* Put the transparent image into another image-type label. */
PtSetResource( ABW_destination_label,
                Pt_ARG_LABEL_IMAGE, &trans_image, 0 );

/* eliminate 'unreferenced' warnings */
link_instance = link_instance, apinfo = apinfo;
            cbinfo = cbinfo;

            return( Pt_CONTINUE );
}

/* Draw function (Pt_ARG_RAW_DRAW_F) for a PtRaw widget */
void
raw_draw_f( PtWidget_t *widget, PhTile_t *damage )
{
    PhPoint_t    pos = {0, 0};
    PhRect_t     rect;

    damage = damage;

    PtSuperClassDraw( PtBasic, widget, damage );

    /* Find our canvas. */
    PtCalcCanvas( widget, &rect );

    /* Set translation so that drawing is relative to the PtRaw widget, not its parent. */
    PgSetTranslation( &rect.ul, Pg_RELATIVE );

    /* Clip to our basic canvas (it’s only polite). */
    PtClipAdd( widget, &rect );

    /* Do our drawing. */
    PgDrawPhImagemx( &pos, &trans_image, 0 );

    /* Remove our translation and clipping by subtracting what we added above. */
    rect.ul.x *= -1;
    rect.ul.y *= -1;
    PgSetTranslation( &rect.ul, Pg_RELATIVE );
    PtClipRemove();
}
PhMakeTransparent()

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**
- PgColor_t
- PgDrawPhImage*( ), PgDrawPhImageRect*( ), PgDrawRepPhImage*( ), PhCreateImage(), PhImage_t
- PhMakeTransBitmap()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhMallocUnpack()

Unpack transport data, using a custom memory-allocation function

Synopsis:

```c
char * PhMallocUnpack( PhTransportHdr_t *hdr,
                    void **ret_struct,
                    TransportMalloc_t *ymalloc,
                    void *ymalloc_cb_data );
```

Library:

ph

Description:

This function unpacks data packed using one of the `PhTransport*` or `PtTransport*` functions. Any memory required for the extraction is allocated via calls to the provided `ymalloc` function. If `ymalloc` is NULL, a default allocator is used.

The `hdr` parameter describes the data to be unpacked and is itself extracted from the packed data stream. See `PhGetTransportHdr()`.

The pointer referenced by `ret_struct` is set to point to the newly unpacked data. The actual data representation of the data extracted is described in the `hdr` that’s passed as the first parameter to this function.

The `TransportMalloc_t` type (used for the prototype of the `ymalloc` function) is:

```c
typedef void *TransportMalloc_t(
    PhTransportRegEntry_t *trans_entry,
    void *cbdata,
    int fixup_index,
    size_t size );
```

For each piece of memory that must be allocated while unpacking, the provided `ymalloc` function is called with the following arguments:

- the transport registry entry for the data being extracted — see `PhTransportRegEntry_t`
- the provided `ymalloc_cb_data`
**PhMallocUnpack()**

- the index for the data into the *fixup* array in the transport registry entry
- the number of bytes that must be allocated.

This makes it very easy to unpack specific pieces of data into shared memory blocks, etc.

As the data is unpacked, it’s automatically endian-corrected.

To free data unpacked using this function, call `PhFreeTransportType()`.

**Returns:**

A pointer to the byte in the data stream following the data just extracted.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhAllocPackType()`, `PhFindTransportType()`, `PhFreeTransportType()`, `PhPackEntry()`, `PhRegisterTransportType()`, `PhTransportRegEntry_t`, `PhTransportType()`, `PhUnpack()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
PhTile_t * PhMergeTiles( PhTile_t *tiles );
```

Library:

`ph`

Description:

This function removes all overlap from the list of tiles pointed to by `tiles`.

Returns:

The same pointer as `tiles`.

Don’t `free()` the list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`, `PhCopyTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`, `PhGetTile()`, `PhIntersectTilings()`, `PhRectsToTiles()`, `PhSortTiles()`, `PhTile_t`, `PhTilesToRects()`, `PhTranslateTiles()`
PhMergeTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhMoveCursorAbs()

Move cursor to absolute position

Synopsis:

```c
void PhMoveCursorAbs( int input_group,
                      int x,
                      int y );
```

Library:

ph

Description:

This function moves the cursor for `input_group` to the absolute coordinates specified by `x` and `y`.

To determine the current input group, call `PhInputGroup()`, passing to it the current event, if any.

This function contravenes the standards put forth by the OSF/Motif Style Guide. Most users will be disconcerted if your program changes the pointer position. See section 2.2.3.2 of the OSF/Motif Style Guide (ISBN 0-13-640491-X).

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PhMoveCursorRel()`, `PhQueryCursor()`
**PhMoveCursorRel()**

*Move cursor to relative position*

**Synopsis:**

```c
void PhMoveCursorRel( int input_group,
                       int x,
                       int y );
```

**Library:**

*ph*

**Description:**

This function moves the cursor for `input_group` to the coordinates specified by `x` and `y` relative to the current cursor position.

To determine the current input group, call `PhInputGroup()`, passing to it the current event, if any.

This function contravenes the standards put forth by the *OSF/Motif Style Guide*. Most users will be disconcerted if your program changes the pointer position. See section 2.2.3.2 of the *OSF/Motif Style Guide* (ISBN 0-13-640491-X).

**Classification:**

*Photon*

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhMoveCursorAbs(), PhQueryCursor()*
PhPackEntry()

Pack transport data, given a transport registry entry

Synopsis:

```c
char * PhPackEntry(
    char * buffer,
    PhTransportRegEntry_t *regent,
    char const * const type,
    char const * const desc,
    int unsigned const grouping_num,
    int unsigned const handle,
    int unsigned const request_transport,
    int unsigned const inlined_transport,
    char const * const data,
    int unsigned const size,
    int unsigned const *tot_size,
    iov_t *iovs,
    int unsigned *niov,
    int unsigned *iovsize
);
```

Library:

ph

Description:

This function takes the data referenced by `data` and packs it into a stream buffer pointed to by `buffer`.

You're not likely to call this function directly; call `PhTransportType()` instead.

The arguments to `PhPackEntry()` are:

- **buffer**  
  A pointer to the buffer in which to pack the data:
  
  - If `buffer` is NULL, the function simply calculates the size of the buffer required to hold the data. In this case, `tot_size`, `niov`, and `iovsize` should all be nonNULL.
  
  - If `buffer` isn’t NULL, the function packs the data into the buffer, advancing the current position in the buffer.
**PhPackEntry()**

regent  A pointer to the transport registry entry for the type of data to be packed. For more information, see `PhTransportRegEntry_t`.

If you don’t have a pointer to the registry entry for the data to be packed, call `PhPackType()` instead of `PhPackEntry()`.

type  A descriptive type name, such as `image`, `text`, `filename`, or `files`. This is simply added to the header for the packed data.

desc  The specifics of what’s in the data. The extractor uses a regular-expression match against the description to determine if the data should be unpacked or discarded. This is simply added to the header for the packed data.

grouping_num  When used with Photon’s drag and drop mechanism, the `grouping_num` is used to indicate which stream is just a different representation of other data also packed into the same `PhTransportCtrl_t` structure. Only one of each `grouping_num` should be unpacked by the reader/destination.

This value is simply added to the header for the packed data.

handle  A number that you can use to identify a transaction. This is simply added to the header for the packed data.

request_transport  The available transport types that can be specified when requesting data from the source. This can be any of:

- `Ph.TRANSPORT_INLINE` — the data being transported is in memory and can be unpacked immediately.
- `Ph.TRANSPORT_FILEREF` — the data being transported is in the temporary file(s) named in the inlined data.
PhPackEntry()

- Ph_TRANSPORT_SHMEM — the data being transported is in the temporary shared object(s) named in the inlined data.
- Ph_TRANSPORT_STREAM — the data being transported will be inlined a small piece at a time.
- Ph_TRANSPORT_NAMED_STREAM — the data being transported will be inlined a small piece at a time. The streamed data is named so multiple streams of data can be transferred serially.
- Ph_TRANSPORT_FILE_STREAM — the contents of files are streamed using extended named streams. This is like the named stream but with extra information with each data block, including file information and so on. The requester of data must choose one of the available request transport types when requesting delivery of additional data.

**inlined_transport**

The transport type used for the inlined data. This can be one of:

- Ph_TRANSPORT_INLINE — the data being transported is in memory and can be unpacked immediately.
- Ph_TRANSPORT_FILEREF — the data being transported is in the temporary file(s) named in the inlined data.
- Ph_TRANSPORT_SHMEM — the data being transported is in the temporary shared object(s) named in the inlined data.

**data**

A pointer to the data to be packed.

**size**

The size, in bytes, of the data to be packed. This size is used only for raw data.
**PhPackEntry()**

- **tot_size**  
  A pointer to a variable in which to store the total size of the buffer. This should be non-NULL if you’re calculating the size required for the buffer (i.e. buffer is NULL).

- **iovs**  
  A pointer to an array of I/O vectors.

- **niovs**  
  A pointer to a variable in which to store the number of I/O vectors required. This should be non-NULL if you’re calculating the size required for the buffer (i.e. buffer is NULL).

- **iovsize**  
  A pointer to a variable in which to store the size of the I/O vectors required. This should be non-NULL if you’re calculating the size required for the buffer (i.e. buffer is NULL).

**Returns:**

The current position in the buffer after packing the data, or NULL if no data was packed.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhAllocPackType(), PhFindTransportType(), PhMallocUnpack(), PhPackType(), PhRegisterTransportType(), PhTransportCtrl_t, PhTransportRegEntry_t, PhTransportType(), PhUnpack()

Drag and Drop chapter of the Photon Programmer’s Guide
PhPackType()

Pack transport data, given the type of data

Synopsis:

```
char * PhPackType( char *buffer,
                   char const * const type,
                   char const * const desc,
                   int unsigned const grouping_num,
                   int unsigned const handle,
                   int unsigned const request_transport,
                   int unsigned const inlined_transport,
                   char const * const packing_type,
                   void const * const data,
                   int unsigned const size,
                   int unsigned const *tot_size,
                   iov_t *iovs,
                   int unsigned const *nios,
                   int unsigned const *iovsize );
```

Library:

ph

Description:

This function takes the data referenced by data and packs it into a stream buffer pointed to by buffer.

You’re not likely to call this function directly; call PhTransportType() instead.

The arguments to PhPackType() are:

- **buffer**: A pointer to the buffer in which to pack the data:
  - If buffer is NULL, the function simply calculates the size of the buffer required to hold the data. In this case, tot_size, nios, and iovsize should all be nonNULL.
  - If buffer isn’t NULL, the function packs the data into the buffer, advancing the current position in the buffer.
**PhPackType()**

**type**  
A descriptive type name, such as `image`, `text`, `filename`, or `files`. This is simply added to the header for the packed data.

**desc**  
The specifics of what’s in the data. The extractor uses a regular-expression match against the description to determine if the data should be unpacked or discarded. This is simply added to the header for the packed data.

**grouping_num**  
When used with Photon’s drag and drop mechanism, the `grouping_num` is used to indicate which stream is just a different representation of other data also packed into the same `PhTransportCtrl_t` structure. Only one of each `grouping_num` should be unpacked by the reader/destination.

This value is simply added to the header for the packed data.

**handle**  
A number that you can use to identify a transaction. This is simply added to the header for the packed data.

**request_transport**  
The available transport types that can be specified when requesting data from the source. This can be any of:

- **Ph_TRANSPORT_INLINE** — the data being transported is in memory and can be unpacked immediately.

- **Ph_TRANSPORT_FILEREF** — the data being transported is in the temporary file(s) named in the inlined data.

- **Ph_TRANSPORT_SHMEM** — the data being transported is in the temporary shared object(s) named in the inlined data.
PhPackType()

- Ph_TRANSPORT_STREAM — the data being transported will be inlined a small piece at a time.
- Ph_TRANSPORT_NAMED_STREAM — the data being transported will be inlined a small piece at a time. The streamed data is named so multiple streams of data can be transferred serially.
- Ph_TRANSPORT_FILE_STREAM — the contents of files are streamed using extended named streams. This is like the named stream but with extra information with each data block, including file information and so on. The requester of data must choose one of the available request transport types when requesting delivery of additional data.

**inlined_transport**

The transport type used for the inlined data. This can be one of:

- Ph_TRANSPORT_INLINE — the data being transported is in memory and can be unpacked immediately.
- Ph_TRANSPORT_FILEREF — the data being transported is in the temporary file(s) named in the inlined data.
- Ph_TRANSPORT_SHMEM — the data being transported is in the temporary shared object(s) named in the inlined data.

**packing_type**

The name of the entry in the transport registry to be used to pack the data. For more information, see `PhTransportRegEntry_t`. If you already have a pointer to the registry entry, you can call `PhPackEntry()` instead of `PhPackType()`.

**data**

A pointer to the data to be packed.
PhPackType()

- **size**: The size, in bytes, of the data to be packed. This size is used only for raw data.
- **tot_size**: A pointer to a variable in which to store the total size of the buffer. This should be non-NULL if you’re calculating the size required for the buffer (i.e. buffer is NULL).
- **iovs**: A pointer to an array of I/O vectors.
- **niovs**: A pointer to a variable in which to store the number of I/O vectors required. This should be non-NULL if you’re calculating the size required for the buffer (i.e. buffer is NULL).
- **iovsize**: A pointer to a variable in which to store the size of the I/O vectors required. This should be non-NULL if you’re calculating the size required for the buffer (i.e. buffer is NULL).

**Returns:**

A pointer to the buffer, or NULL if no data was packed.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PhPackType()

See also:

PhAllocPackType(), PhFindTransportType(), PhMallocUnpack(), PhPackEntry(), PhRegisterTransportType(), PhTransportCtrl_t, PhTransportRegEntry_t, PhTransportType(), PhUnpack()

Drag and Drop chapter of the Photon Programmer's Guide
PhPoint_t
Coordinates of a single point

Synopsis:

```c
typedef struct Ph_point {
    short x, y;
} PhPoint_t;
```

Description:

The PhPoint_t structure describes the coordinates of a single point. It contains at least the following members:

- `x` X-axis coordinate.
- `y` Y-axis coordinate.

Classification:

Photon

See also:

PhArea_t, PhDim_t, PhRect_t

“Geometry data types” in the Working with Code chapter of the Photon Programmer’s Guide
**PhPointerEvent_t**

Data associated with a pointer event

**Synopsis:**

```c
typedef struct Ph_ev_ptr_data {
    PhPoint_t pos;
    unsigned short buttons;
    unsigned short button_state;
    unsigned char click_count;
    unsigned char flags;
    short z;
    unsigned long keyMods;
    unsigned long zero;
} PhPointerEvent_t;
```

**Description:**

The `PhPointerEvent_t` structure holds the data associated with all pointer events:

- `Ph_EV_BUT_PRESS`
- `Ph_EV_BUT_RELEASE`
- `Ph_EV_BUT_REPEAT`
- `Ph_EV_PTR_MOTION_BUTTON`
- `Ph_EV_PTR_MOTION_NOBUTTON`

For more information, see `PhEvent_t`.

The members of `PhPointerEvent_t` include:

- **pos**
  The untranslated, absolute position of the current pointer focus. As a rule, you should use the event’s rectangle set to determine coordinate positions. However, for situations that demand absolute coordinates (for example, calibrating a touchscreen), you can use `pos`.

- **buttons**
  Indicates which buttons the user pressed or released, depending on the event. This is a combination of:
PhPointerEvent_t

- Ph_BUTTON_SELECT—normally the left button. Because a pointing device might provide only this button, you should design most applications such that the user has the option to use this button to perform any task.

- Ph_BUTTON_MENU—normally the right button. It can be used to invoke menus when they’re available.

- Ph_BUTTON_ADJUST—normally the middle button on a three-button pointer. Its use is currently unspecified.

button_state The current state of all the buttons (i.e. which buttons are pressed). This is a combination of the same bits as buttons.

click_count The number of clicks (for example, a value of 2 indicates a double-click). See the Ph_EV_RELEASE_ENDCLICK subtype for Ph_EV_BUT_RELEASE.

flags Indicates that the z field is valid.

z Can be used with touchscreens to indicate touch pressure.

key_mods The modifier keys that are currently pressed. This is a combination of:

- Pk_KM_Shift
- Pk_KM_Ctrl
- Pk_KM_Alt
- Pk_KM_AltGr
- Pk_KM_Sh13
- Pk_KM_Mod6
- Pk_KM_Mod7
- Pk_KM_Mod8
PhPointerEvent_t

- Pk_KM_Shift_Lock
- Pk_KM_Ctrl_Lock
- Pk_KM_Alt_Lock
- Pk_KM_AltGr_Lock
- Pk_KM_Shl3_Lock
- Pk_KM_Mod6_Lock
- Pk_KM_Mod7_Lock
- Pk_KM_Mod8_Lock
- Pk_KM_Caps_Lock
- Pk_KM_Num_Lock
- Pk_KM_Scroll_Lock

If the Shift key is pressed, the Shift modifier is on; if it’s released, the Shift modifier is off. Because some keys occur twice on the keyboard, a key release doesn’t guarantee that the corresponding modifier is off — the matching key may still be pressed.

Classification:
Photon

See also:
PhEvent_t, PhKeyEvent_t

Events chapter of the Photon *Programmer’s Guide*
**PhQueryCursor()**

*Collect cursor information*

**Synopsis:**

```c
int PhQueryCursor( unsigned short ig, 
                 PhCursorInfo_t *buf );
```

**Arguments:**

- `ig` The input group which contains the cursor you want to query. Set to 0 to query any input group.
- `buf` A pointer to a `PhCursorInfo_t` structure that the function puts cursor information into.

**Library:**

`ph`

**Description:**

This function collects information about the cursor for input group `ig` and places this information in the provided structure `buf`.

To determine the current input group, call `PhInputGroup()`, passing to it the current event, if any.

The `PhCursorInfo_t` structure is defined in `<photon/PhT.h>` and contains at least:

- `PhPoint_t pos` Last position, in absolute coordinates.
- `PhRid_t region` Region that currently contains the cursor.
- `PhRid_t ig_region` Region representing the input group.
- `PgColor_t color` Cursor color.
PhQueryCursor()

**PhPoint_t** last_press

Location of last Ph_EV_BUT_PRESS event.

**unsigned long** msec

msec of last press.

**PhPoint_t** steady

Last steady base point.

**PhRid_t** dragger

Region currently dragging.

**PhRect_t** drag_boundary

A PhRect_t structure that defines the area to which to restrict dragging.

**PhRid_t** phantom_rid

Region ID to deliver phantom to.

**unsigned short** type

Cursor type (from cursor font).

**unsigned short** ig

Input group number.

**unsigned short** button_state

Flags that indicate which pointer buttons changed their state:

- Ph_BUTTON_SELECT
- Ph_BUTTON_MENU
- Ph_BUTTON_ADJUST

**unsigned char** click_count

The number of button clicks.

**unsigned long** key_mods

Flags indicating which modifier keys are currently held down.
**PhQueryCursor()**

**Returns:**

0  Successful completion.
-1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PgColor_t, PhMoveCursorAbs(), PhMoveCursorRel(), PhPoint_t, PhRect_t
**PhQueryRids()**

*Get a list of regions*

**Synopsis:**

```c
int PhQueryRids( unsigned flags,
                 PhRid_t rid,
                 unsigned input_group,
                 unsigned type,
                 unsigned sense,
                 PhRid_t emitter,
                 const PhRect_t *rect,
                 PhRid_t rids[],
                 int num );
```

**Library:**

*ph*

**Description:**

This function builds a list of up to `num` regions in the `rids` array. The other parameters specify which regions are to be included in the list:

- **flags**
  The possible flag bits are:
  - Ph_RIDQUERY_IG_POINTER — use `input_group`’s pointer position as `rect`.
  - Ph_RIDQUERY_TOWARD — act as if the event were emitted towards the user (away from the root region).

- **rid**
  Consider regions that intersect with the region with this ID. Set `rid` to 0 to consider all regions.

- **input_group**
  Consider regions that belong to this input group (0 means any).
  To determine the current input group, call `PhInputGroup()`, passing to it the current event, if any.

- **type**
  Consider regions of these types:
  - Ph_WINDOW_REGION
**PhQueryRids()**

- Ph_WND_MGR_REGION
- Ph_GRAFX_REGION
- Ph_PTR_REGION
- Ph_KBD_REGION
- Ph_PRINT_REGION
- Ph_INPUTGROUP_REGION
- Ph_AUXPTR_REGION
- Ph_FORCE_FRONT
- Ph_FORCE_BOUNDARY

Set *type* to 0 (i.e. all bits off) to consider all types of regions.

*sense* Consider regions that are sensitive or opaque to these event types. The possible bits are:

- Ph_EV_BOUNDARY
- Ph_EV_BUTPRESS
- Ph_EV_BUT_RELEASE
- Ph_EV_BUT_REPEAT
- Ph_EV_DNDROP
- Ph_EV_DRAG
- Ph_EV_DRAW
- Ph_EV_EXPOSE
- Ph_EV_INFO
- Ph_EV_KEY
- Ph_EV_PTR_MOTION_BUTTON
- Ph_EV_PTR_MOTION_NOBUTTON
- Ph_EV_RAW
- Ph_EV_SERVICE
- Ph_EV_SYSTEM
- Ph_EV_TIMER
PhQueryRids()

- Ph_EV_WM
  Set sense to 0 (i.e. all bits off) to consider regions regardless of their sensitivity.

emitter The region to begin the query from.

rect A pointer to a PhRect_t structure that specifies the area (relative to emitter’s origin) that other regions must intersect to be considered in the query. This can be NULL.

Returns:

The number of regions found, or -1 if an error occurred.

If this function returns -1, check errno. If it’s ENOMSG, there are no pending Photon events; this isn’t really an error.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhEvent_t, PhRect_t
PhQuerySystemInfo()
Query the system for information about a given region

Synopsis:

```c
PhSysInfo_t * PhQuerySystemInfo(
    PhRid_t rid,
    PhRect_t const *rect,
    PhSysInfo_t *sysinfo
);
```

Library:

ph

Description:

This function queries the system for information for the region with the given rid:

- system bandwidth
- graphics drivers and their capabilities
- pointing devices
- keyboard devices

The information is stored in the PhSysInfo_t structure pointed to by sysinfo. Photon reports information about itself and system regions that intersect the rectangular area specified by the PhRect_t structure pointed to by rect. If rect is NULL, the area is the extent of the given region.

Returns:

A pointer to the PhSysInfo_t structure passed to the function, or NULL if an error occurred.

Classification:

Photon
**PhQuerySystemInfo()**

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhRect_t, PhSysInfo_t, PtQuerySystemInfo()*

**PhReattach()**

*Change the current Photon channel*

**Synopsis:**

```c
struct _Ph_ctrl *PhReattach( struct _Ph_ctrl *Ph );
```

**Library:**

`ph`

**Description:**

This function lets you change the current Photon channel if more than one channel has been opened. The function flushes the draw buffer before changing the channel.

The `Ph` argument points to a Photon control structure returned by a previous call to `PhAttach()`.

If `Ph` is NULL, this function simply returns a pointer to the current Photon control structure.

**Returns:**

Returns a pointer to the previous Photon channel.

**Examples:**

`See PhAttach()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

984 Chapter 9 • Ph—Photon
See also:

PhAttach(), PhDetach()
**PhRect_t**

*Coordinates of a rectangle*

**Synopsis:**

```c
typedef struct Ph_rect {
    PhPoint_t ul, lr;
} PhRect_t;
```

**Description:**

The `PhRect_t` structure describes the coordinates of a rectangle. It contains at least the following members:

- `ul`  Upper-left corner.
- `lr`  Lower-right corner.

**Classification:**

Photon

**See also:**

- `PhArea_t`, `PhAreaToRect()`, `PhDeTranslateRect()`, `PhDim_t`, `PhPoint_t` `PhRectIntersect()`, `PhRectToArea()`, `PhRectUnion()`, `PhTranslateRect()`
- “Geometry data types” in the Working with Code chapter of the Photon *Programmer’s Guide*
PhRectIntersect()

Find the intersection of two rectangles

Synopsis:

```c
int PhRectIntersect( PhRect_t *rect1,
                     PhRect_t const *rect2 );
```

Library:

`ph`

Description:

This function finds the intersection of two rectangles. If rectangles `rect1` and `rect2` intersect, this function sets `rect1` to that intersection.

Returns:

- ≠ 0 There was an intersection.
- 0 There was no intersection.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PhRect_t, PhRectUnion()`
**PhRectsToTiles()**

Create a list of tiles from an array of rectangles

**Synopsis:**

```c
PhTile_t * PhRectsToTiles( PhRect_t *rects, int num_rects );
```

**Library:**

`ph`

**Description:**

This function creates a list of tiles from the array of rectangles pointed to by `rects`, with `num_rects` entries.

**Returns:**

A pointer to the list of tiles.

---

Don’t `free()` the list of tiles; instead, use `PhFreeTiles()` to return the tiles to the internal pool.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`,
`PhCopyTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`, `PhGetTile()`,
`PhIntersectTilings()`, `PhMergeTiles()`, `PhRect_t`, `PhSortTiles()`,
`PhTile_t`, `PhTilesToRects()`, `PhTranslateTiles()`
“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
PhRectToArea()
Convert a rectangle into an area

Synopsis:

```c
void PhRectToArea( PhRect_t const *rect,
                   PhArea_t *area);
```

Library:

ph

Description:

This function converts a rectangle (i.e. upper-left and lower-right points) into an area (i.e. a position and dimensions).

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhArea_t, PhAreaToRect(), PhRect_t
PhRectUnion()

Determine a bounding box for two rectangles

Synopsis:

```c
int PhRectUnion( PhRect_t *rect1,
                 PhRect_t const *rect2 );
```

Library:

ph

Description:

This function changes the rectangle pointed to by `rect1` to a rectangle that encompasses `rect2` using `rect1` as a starting point. The result is a bounding box for the two original rectangles.

Returns:

0  The resulting rectangle is inverted (only possible if inverted rectangles are provided as parameters i.e. `ul > lr`).
1  The resulting rectangle is a regular rectangle (i.e. `ul < lr`).

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhRect_t`, `PhRectIntersect()`
PhRegion_t

Data structure that describes a region

Synopsis:

See below.

Description:

The PhRegion_t structure describes a region. It contains at least the following members:

PgColor_t cursor_color

The cursor color for this region.

unsigned short cursor_type

Sets the cursor type for this region. See <PhCursor.h>. A cursor_type of Ph_CURSOR_INHERIT indicates this region inherits the cursor from the parent region.

If you OR cursor_type with Ph_CURSOR_NO_INHERIT, then children of this region won’t inherit its cursor type. The children inherit the cursor from their first ancestor that doesn’t have the Ph_CURSOR_NO_INHERIT flag set.

PhRid_t rid

This region’s ID. The Photon Manager assigns this when the region is opened.

long handle

A user-definable handle that is included as part of the event structure. Applications can use handle to quickly pass a small amount of information along with events. For example, the widget toolkit functions (Pt*) use handle to point to a widget in memory so that they can quickly find the appropriate callback.

PhConnectId_t owner

Indicates the owner of this region. See PhGetConnectInfo().
**PhRegion_t**

**unsigned long flags**

Controls certain aspects of this region and also indicates the region’s type. Of the following flags, `Ph_FORCE_BOUNDARY` and `Ph_FORCE_FRONT` affect how the region behaves. The others simply indicate the region type.

These *type* flags are set by the API functions for the convenience of applications that wish to identify a region’s purpose. For example, an application can use these flags when querying the Photon Manager for a list of regions that have a specific type.

You can OR the following into *flags*:

- **Ph_FORCE_BOUNDARY**
  Determines whether the cursor points at this region. Photon considers the cursor to point at the foremost region with this bit set. This affects the cursor color and type, and whether the region causes some types of `Ph_EV_BOUNDARY` events.

- **Ph_FORCE_FRONT**
  Forces the Photon Manager to place this region in front of any of its brothers that don’t have this flag set, and behind any brothers that do have this flag set.

- **Ph_GRAFX_REGION**
  Indicates that this region belongs to a graphics driver.

- **Ph_INPUTGROUP_REGION**
  Indicates that this region is an input group.

- **Ph_KBD_REGION**
  Indicates that this region belongs to an input/keyboard driver.
**PhRegion_t**

**Ph_PTR_REGION**
Indicates that this region belongs to an
pointer/mouse driver.

**Ph_WINDOW_REGION**
Indicates that this region is a window.

**Ph_WND_MGR_REGION**
Indicates that the Window Manager owns this
region.

**unsigned long events_sense**
Determines which event types this region is
sensitive to. If an event is one of these types and it
passes through the region, the event is enqueued to
the application.

**unsigned long events_opaque**
Determines which event types this region is opaque
to. If an event is one of these types and it passes
through the region, any portion of the event that
intersects with the region is clipped out.

**PhPoint_t origin**
A *PhPoint_t* structure that determines the region’s
origin relative to its parent’s origin. All coordinates
returned in events and elsewhere in this structure are
relative to *origin*.

In almost all cases, the coordinates used by the API
functions are relative to the origin of the caller’s
region. However, when emitting events, the
application may use absolute coordinates by setting
the Ph_EVENT_ABSOLUTE flag.

**PhRid_t parent**
Indicates the region’s parent.
PhRegion_t

PhRid_t child  Indicates the frontmost child region (that is, closest to the user). If no child regions exist, the Photon Manager sets child to -1.

PhRid_t bro_in_front  Indicates the brother region that’s located immediately in front. If there’s no brother in front, the Photon Manager sets bro_in_front to -1.

PhRid_t bro_behind  Indicates the brother region that’s located immediately behind. If there’s no brother behind, the Photon Manager sets bro_behind to -1.

unsigned short data_len  The length of the data portion of this region. The data is stored in a single block of memory; to get a copy of it, call PhRegionQuery().

The region data may consist of blocks of data of different types, each preceded by a PhRegionDataHdr_t structure. There’s at most one block of a given type. To find a specific type of data in the region’s data, call PhRegionDataFindType().

If you need to change the region data, modify your copy or create a new block (starting with a PhRegionDataHdr_t structure) and pass it to PhRegionChange().

unsigned short input_group  Indicates the number of the input group. A value of 0 means this region isn’t an input group.

unsigned short num_rects  Indicates the number of rectangles the region has. Currently, always set to 1.
PhRegion_t

Classification:
Photon

See also:
PgColor_t, PgGetRegion(), PgSetRegion(), PhPoint_t,
PhRegionChange(), PhRegionClose(), PhRegionDataFindType(),
PhRegionDataHdr_t, PhRegionInfo() PhRegionOpen(),
PhRegionQuery()

Regions chapter of the Photon Programmer’s Guide
PhRegionChange()

Change the definition of a region

Synopsis:

```c
int PhRegionChange( unsigned long fields,
                    unsigned long flags,
                    PhRegion_t const *info,
                    PhRect_t const *rect,
                    void const *data );
```

Library:

ph

Description:

This function changes the definition of the region specified by `info->rid`. The `fields` argument describes which fields in the `info` structure are to be changed — for more information, see `PhRegionOpen()`.

The `rect` argument points to a `PhRect_t` structure that defines the rectangle associated with the region, and `data` points to data associated with the region. If you don’t specify the region’s rectangle and data in the `fields` argument, you can set `rect` and `data` to NULL.

The data consists of one or more `PhRegionDataHdr_t` structures, each followed immediately by the appropriate type of data. This data is merged into any existing region data, replacing the blocks of the same types as given in `data`.

The `flags` argument controls whether or not an expose event will be emitted to this region, when necessary. You can OR the following into `flags`:

- **Ph_EXPOSE_REGION**
  
  If part of the region becomes exposed, send a Ph_EV_EXPOSE event to the region.

- **Ph_EXPOSE_FAMILY**
  
  If part of the region becomes exposed, send a Ph_EV_EXPOSE event to the region’s descendants.
PhRegionChange()

Returns:

A nonnegative value
Successful completion.

-1 An error occurred.

Examples:

```c
#include <Pt.h>

typedef struct my_data_things {
    PhRegionDataHdr_t hdr;
    char my_string[20];
}MyData_t;

static void display_data( ushort_t type,
    PhRegion_t *region, PhRegionDataHdr_t *data)
{
    PhRegionDataHdr_t *data_hdr;
    if ((data_hdr = PhRegionDataFindType(region,
        data, type))){
        MyData_t *regdata = (void *)data_hdr;
        printf("data len: %d, content: %s\n",
            data_hdr->len, regdata->my_string);
    } else
        printf("No region data matching type found\n");
}

int main() {
    PtWidget_t *win;
    PhRegion_t region;
    char data[1000];
    MyData_t region_data;
    PhRid_t rid;

    /* Initialize the photon widget library */
    PtInit(NULL);

    /* Create a nondescript window */
    win = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0,
        NULL);

    /* Realize it so it has a region */
    PtRealizeWidget(win);
    rid = PtWidgetRid(win);

    /* Populate the region_data struct */
    region_data.hdr.type = Ph_RDATA_USER;
```
PhRegionChange()

region_data.hdr.len = sizeof(region_data);
strcpy(region_data.my_string, "This is some data");

region.data_len = region_data.hdr.len;
region.rid = rid;
PhRegionQuery(rid, &region, NULL,
              (PhRegionDataHdr_t *)&data,
              sizeof(data));

/* Add the data to the region */
region.data_len = region_data.hdr.len;
region.rid = rid;
PhRegionChange(Ph_REGION_DATA, 0, &region, NULL,
              &region_data);

/* Retrieve the data from the region */
PhRegionQuery(rid, &region, NULL,
              (PhRegionDataHdr_t *)&data,
              sizeof(data));
display_data(Ph_RDATA_USER, &region,
              (PhRegionDataHdr_t *)&data);

/* Remove the data from the region */
region.data.hdr.len = 0;
PhRegionChange(Ph_REGION_DATA, 0, &region, NULL,
              &region_data);

/* Retrieve the data from the region */
PhRegionQuery(rid, &region, NULL,
              (PhRegionDataHdr_t *)&data,
              sizeof(data));
display_data(Ph_RDATA_USER, &region,
              (PhRegionDataHdr_t *)&data);
return EXIT_SUCCESS;

Classification:
Photon

Safety

Interrupt handler  No
Signal handler     No

continued...
**PhRegionChange()**

Safety

| Thread | No |

See also:

*PhRect_t, PhRegion_t, PhRegionClose(), PhRegionDataHdr_t, PhRegionOpen(), PhRegionQuery()*

Regions chapter of the Photon *Programmer’s Guide*
PhRegionClose()  
Remove a region

Synopsis:

```c
int PhRegionClose( PhRid_t rid );
```

Library:

`ph`

Description:

This function removes the specified region from the current Photon Manager. If the specified region has child regions, they’re removed as well.

Returns:

- A nonnegative value
  - Successful completion.
- -1 An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PhRegion_t`, `PhRegionChange()`, `PhRegionOpen()`

Regions chapter of the Photon Programmer’s Guide
PhRegionDataFindType()
Find a data type within a region’s data

Synopsis:

```c
PhRegionDataHdr_t *
PhRegionDataFindType(
    PhRegion_t const *region,
    PhRegionDataHdr_t const *data,
    short type);
```

Library:

ph

Description:

This function finds the specified type of data within the provided region’s data block. For a list of types, see the description of PhRegionDataHdr_t.

Returns:

A pointer to a PhRegionDataHdr_t structure that matches the specified data type. If no data entries within the region’s data block match the specified type, the function returns NULL.

Examples:

See PhRegionChange().

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PhRegionDataFindType()

See also:

PhRegion_t, PhRegionDataHdr_t, PhRegionChange(), PhRegionQuery()

Regions chapter of the Photon Programmer’s Guide
PhRegionDataHdr_t

Data that’s attached to a region

Synopsis:

typedef struct Ph_region_data_hdr {
    unsigned short len;
    unsigned short type;
} PhRegionDataHdr_t;

Description:

The PhRegionDataHdr_t structure describes data that’s attached to a region. It includes at least the following members:

- **len**  
The length of the data, in bytes. The data immediately follows the PhRegionDataHdr_t structure in the region’s block of data.

- **type**  
The type of data, which indicates the data structure used:

<table>
<thead>
<tr>
<th>Type</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ph_RDATA_WINDOW</td>
<td>PhWindowInfo_t</td>
</tr>
<tr>
<td>Ph_RDATA_CURSOR</td>
<td>PhCursorDef_t</td>
</tr>
<tr>
<td>Ph_RDATA_IG</td>
<td>PhIgRegionData_t</td>
</tr>
<tr>
<td>Ph_RDATA_GFXINFO</td>
<td>PhGrafxRegionData_t</td>
</tr>
<tr>
<td>Ph_RDATA_KBDINFO</td>
<td>PhKbdRegionData_t</td>
</tr>
<tr>
<td>Ph_RDATA_PTRINFO</td>
<td>PhPtrRegionData_t</td>
</tr>
<tr>
<td>Ph_RDATA_WMCONFIG</td>
<td>WmConfig_t</td>
</tr>
<tr>
<td>Ph_RDATA_GFXDETAIL</td>
<td>PhGrafxDetail_t</td>
</tr>
<tr>
<td>Ph_RDATA_INPMGRINFO</td>
<td>Internal use only</td>
</tr>
<tr>
<td>Ph_RDATA_CLIPBOARD</td>
<td>PhClipboardHdr</td>
</tr>
<tr>
<td>Ph_RDATA_USER</td>
<td>User-defined</td>
</tr>
</tbody>
</table>
Classification:

Photon

See also:

PhRegion_t, PhRegionDataFindType()

Regions chapter of the Photon *Programmer's Guide*
**PhRegionInfo()**  
Retrieve information about a region

**Synopsis:**
```c
int PhRegionInfo( PhRid_t rid,  
                 PhRegion_t *region,  
                 PhRect_t rects[],  
                 unsigned nrect,  
                 void *data,  
                 unsigned data_len );
```

**Arguments:**
- `rid` The region to query.
- `region` A `PhRegion_t` structure that the function fills in with region information.
- `rects` An array of `PhRect_t` structures that the function fills in with region rectangle information.
- `nrect` The length of the array pointed to by `rects`. See the note below.
- `data` If the queried region has data attached to it, up to `data_len` bytes is copied into this member. This data may consist of smaller blocks of data of different types, each preceded by a `PhRegionDataHdr_t` structure. To find a specific type of data in the region’s data, call `PhRegionDataFindType()`.
- `data_len` The maximum number of region data bytes to copy into `data`.

**Library:**
```c
ph
```

**Description:**
This function returns information about the region identified by `rid`. On completion, the `PhRegion_t` structure pointed to by `region` and
the PhRect_t structures pointed to by rects contain a description of the region.

This function is similar to PhRegionQuery(), but it supports multiple rectangles. In a future release, it may be possible to create regions with multiple rectangles. If you want your code to be prepared to handle such regions correctly, you may consider using PhRegionInfo() instead of PhRegionQuery().

The number of rectangles in the region is returned in region->num_rects. If that number is greater than nrect, and nrect is greater than zero, then rects[0] is set to the region’s bounding box and the rest of rects remain unchanged.

**Returns:**

0 Success.

-1 An error occurred.

**Examples:**

The following example gets information about the device region:

```c
#define NRECTS 10
PhRegion_t region; PhRect_t rects[ NRECTS ];

if( !PhRegionInfo( Ph_DEV_RID, &region, rects, NRECTS, NULL, 0 ) ) {
    printf( "Sensitive to: %.8x Opaque to: %.8x\n", region.events_sense, region.events_opaque );
    if ( region.num_rects > NRECTS )
        printf( "%d rectangles, bounding box: {(%d,%d),(%d,%d)}\n", region.num_rects,
                region.origin.x + rect[0].ul.x,
                region.origin.y + rect[0].ul.y,
                region.origin.x + rect[0].lr.x,
                region.origin.y + rect[0].lr.y );
    else {
        int i;
        printf( "%d rectangles:\n", region.num_rects );
        for ( i=0; i<region.num_rects; ++i )
            printf( " {(%d,%d),(%d,%d)}\n", region.origin.x + rect[i].ul.x,
                    region.origin.y + rect[i].ul.y,
                    region.origin.x + rect[i].lr.x,
                    region.origin.y + rect[i].lr.y );
```
PhRegionInfo()

```c
    region.origin.x + rect[i].ul.x,
    region.origin.y + rect[i].ul.y,
    region.origin.x + rect[i].lr.x,
    region.origin.y + rect[i].lr.y );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t, PhRegion_t, PhRegionChange(), PhRegionOpen()

Regions chapter of the Photon Programmer’s Guide
Synopsis:

```c
PhRid_t PhRegionOpen( unsigned fields,
                      PhRegion_t const *info,
                      PhRect_t const *rect,
                      void const *data );
```

Library:

`ph`

Description:

This function opens a new region. The `fields` argument describes which members specified by `info` will be used. If some fields aren’t specified, the function sets the corresponding members of the new region to their defaults.

The `info` argument points to a `PhRegion_t` structure that defines a template region used when opening the new region. You must set the `parent` member of `info`; Photon fills in the other family members.

The `rect` argument points to a `PhRect_t` structure that defined the rectangle associated with the region, and `data` points to the data associated with the region.

<table>
<thead>
<tr>
<th><code>fields</code> bit</th>
<th>Argument</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ph_REGION_OWNER</td>
<td><code>info-&gt;owner</code></td>
<td>(you)</td>
</tr>
<tr>
<td>Ph_REGION_HANDLE</td>
<td><code>info-&gt;handle</code></td>
<td>0</td>
</tr>
<tr>
<td>Ph_REGION_FLAGS</td>
<td><code>info-&gt;flags</code></td>
<td>0</td>
</tr>
<tr>
<td>Ph_REGION_EV_OPAQUE</td>
<td><code>info-&gt;events_opaque</code></td>
<td>0</td>
</tr>
<tr>
<td>Ph_REGION_EV_SENSE</td>
<td><code>info-&gt;events_sense</code></td>
<td>0</td>
</tr>
<tr>
<td>Ph_REGION_ORIGIN</td>
<td><code>info-&gt;origin</code></td>
<td><code>{0, 0}</code></td>
</tr>
<tr>
<td>Ph_REGION_PARENT</td>
<td><code>info-&gt;parent</code></td>
<td>Ph_ROOT_RID</td>
</tr>
</tbody>
</table>

continued…
**PhRegionOpen()**

<table>
<thead>
<tr>
<th>fields bit</th>
<th>Argument</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ph_REGION_BEHIND</td>
<td>info-&gt;bro_behind</td>
<td>See PhRegion_t</td>
</tr>
<tr>
<td>Ph_REGION_IN_FRONT</td>
<td>info-&gt;bro_in_front</td>
<td>See PhRegion_t</td>
</tr>
<tr>
<td>Ph_REGION_RECT</td>
<td>rect</td>
<td>{{0, 0}, {0, 0}}</td>
</tr>
<tr>
<td>Ph_REGION_DATA</td>
<td>info-&gt;data_len, data</td>
<td>{0, 0}</td>
</tr>
<tr>
<td>Ph_REGION_CURSOR</td>
<td>info-&gt;cursor_type, cursor_color</td>
<td>0 to inherit</td>
</tr>
</tbody>
</table>

**Returns:**

A positive region ID, or -1 if an error occurred.

**Examples:**

The following example opens a region out of the root region. The new region will sense any mouse motion events that pass through it, and draw a rectangle at the current position of the pointer. If the user clicks in the region, the program will terminate. If a window manager is running, this region will be in front of the window manager. The window manager won’t be aware of the region.

```c
#include <stdio.h>
#include <Ph.h>

PhRid_t open_region( void )
{
    PhRegion_t region;
    PhRect_t rect;
    PhRid_t rid;

    memset(&region, 0, sizeof(region));
    /* Wish to have pointer motion */
    /* events enqueued to us. */
    region.events_sense = Ph_EV_PTR_MOTION | Ph_EV_BUT_RELEASE;

    /* Wish to be opaque to all pointer-type */
    /* events and be visually opaque. */
    region.events_opaque = Ph_EV_PTR_ALL | Ph_EV_DRAW |
```

Chapter 9 • Ph—Photon
PhRegionOpen()

Ph_EV_EXPOSE;
/* Origin at (100,100) relative to root */
region.origin.x = region.origin.y = 100;
region.parent = Ph_ROOT_RID;
/* Open region from (absolute) (100,100)
to (300,300) */
rect.ul.x = rect.ul.y = 0;
rect.lr.x = rect.lr.y = 200;

rid = PhRegionOpen( Ph_REGION_PARENT |
Ph_REGION_EV_SENSE |
Ph_REGION_EV_OPAQUE |
Ph_REGION_ORIGIN |
Ph_REGION_RECT,
&region, &rect, NULL );

/* If open was successful, black out the region */
if( rid != -1 ) {
    PgSetRegion( rid );
    PgSetFillColor( Pg_BLACK );
    PgDrawRect( &rect, Pg_DRAW_FILL );
    PgFlush();
}
return( rid );

void draw_at_cursor( PhPoint_t *pos )
{
    PhRect_t rect;
    static int count = 0;
    rect.ul.x = pos->x - 10;
    rect.ul.y = pos->y - 10;
    rect.lr.x = pos->x + 10;
    rect.lr.y = pos->y + 10;
    switch( ++ count % 3 ) {
        case 0:
            PgSetFillColor( Pg_RED );
            break;
        case 1:
            PgSetFillColor( Pg_GREEN );
            break;
        default:
            PgSetFillColor( Pg_BLUE );
    }
    PgDrawRect( &rect, Pg_DRAW_FILL );
    PgFlush();
}

int main( int argc, char *argv[] )
PhRegionOpen()

{
    PhEvent_t *event;
    int go = 1;

    if( NULL == PhAttach( NULL, NULL ) ) {
        fprintf( stderr, "Couldn’t attach a "
                    "Photon channel.\n" );
        exit( EXIT_FAILURE );
    }

    if( -1 == open_region() ) {
        fprintf( stderr, "Couldn’t open region.\n" );
        exit( EXIT_FAILURE );
    }

    event = (PhEvent_t *)malloc( sizeof( PhEvent_t ) + 1000 );
    if( event == NULL ) {
        fprintf( stderr, "Couldn’t allocate event buffer.\n" );
        exit( EXIT_FAILURE );
    }

    while( go ) {
        if( PhEventNext( event, sizeof( PhEvent_t ) + 1000 ) == Ph_EVENT_MSG ) {
            if( (event->type & Ph_EV_PTR_MOTION) != 0 )
                draw_at_cursor( (PhPoint_t *)PhGetRects( event ) );
            else
                go = 0;
        } else
            fprintf( stderr, "Error.\n" );
    }

    return 0;
}

Classification:

Photon

Safety

Interrupt handler  No
Signal handler      No

continued...
PhRegionOpen()

Safety

| Thread | No |

See also:

PgGetRegion(), PgSetRegion(), PhAttach(), PhRect_t, PhRegion_t, PhRegionChange(), PhRegionClose()

Regions chapter of the Photon Programmer’s Guide
**PhRegionQuery()**  
_Retrieve information about a region_

**Synopsis:**

```c
int PhRegionQuery( PhRid_t rid,  
                  PhRegion_t *region,  
                  PhRect_t *rect,  
                  void *data,  
                  unsigned data_len );
```

**Library:**

`ph`

**Description:**

This function returns information about the region identified by `rid`. On completion, the `PhRegion_t` structure pointed to by `region` and the `PhRect_t` structure pointed to by `rect` contain a description of the region.

If the region has data attached to it, then the data, up to `data_len` bytes, is copied into `data`. This data may consist of smaller blocks of data of different types, each preceded by a `PhRegionDataHdr_t` structure. To find a specific type of data in the region’s data, call `PhRegionDataFindType()`.

This function is similar to `PhRegionInfo()`, but it doesn’t support multiple rectangles. In a future release, it may be possible to create regions with multiple rectangles. If you want your code to be prepared to handle such regions correctly, you should consider using `PhRegionInfo()` instead of `PhRegionQuery()`.

When multiple rectangles are supported, this function will return just the bounding box of the region rather than an accurate shape.

**Returns:**

- **0** Success.
- **-1** An error occurred.
**PhRegionQuery()**

### Examples:

The following example gets information about the device region:

```c
PhRegion_t region; PhRect_t rect;

if( !PhRegionQuery( Ph_DEV_RID, &region,
                   &rect, NULL, 0 ) ) {
    printf( "Sensitive to: %.8x Opaque to: %.8x\n",
            region.events_sense, region.eventsOpaque );
    printf( "Located at: {(%d,%d),(%d,%d)}\n",
            region.origin.x + rect.ul.x,
            region.origin.y + rect.ul.y,
            region.origin.x + rect.lr.x,
            region.origin.y + rect.lr.y );
}
```

### Classification:

**Photon**

**Safety**

<table>
<thead>
<tr>
<th></th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

PhRect_t, PhRegion_t, PhRegionChange(), PhRegionInfo(), PhRegionOpen()

Regions chapter of the Photon *Programmer’s Guide*
**PhRegisterTransportType()**

*Add a new transport type to the transport registry*

**Synopsis:**

```c
int PhRegisterTransportType(
    PhTransportRegEntry_t *ref);
```

**Library:**

ph

**Description:**

This function adds a the transport type definition pointed to by *ref* to the transport registry. For details on defining your own transport types, see “Registering new transport types” in the Drag and Drop chapter of the Photon *Programmer’s Guide*.

**Returns:**

- 0 The new type was added successfully.
- -1 The new type couldn’t be added because there wasn’t enough memory, the type was already defined, or *ref* was NULL.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhFindTransportType(), PhFreeTransportType(), PhMallocUnpack(), PhPackEntry(), PhTransportRegEntry_t, PhTransportType(), PhUnpack()*

1016 Chapter 9 • Ph—Photon
PhRegisterTransportType()

“Registering new transport types” in the Drag and Drop chapter of the Photon Programmer’s Guide
PhReleaseImage()  
Release the allocated members of an image

Synopsis:

```c
void PhReleaseImage( PhImage_t *image );
```

Library:

ph

Description:

This function releases the allocated members of the PhImage_t pointed to by `image`, based on the value of `image->flags`. The valid flags are:

- Ph_RELEASE_IMAGE
- Ph_RELEASE_PALETTE
- Ph_RELEASE_TRANSPARENCY_MASK
- Ph_RELEASE_GHOST_BITMAP
- Ph_RELEASE_IMAGE_ALL—free all the above.

This function doesn’t release the image structure itself.

Don’t use PhReleaseImage() on an image acquired using PgReadScreen().

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1018  Chapter 9  Ph—Photon
PhReleaseImage()

See also:

ApGetImageRes(), PgDrawPhImage*, PgDrawPhImageRect*, PgDrawRepPhImage*, PhCreateImage(), PhImage_t, PhMakeGhostBitmap(), PhMakeTransBitmap(), PmMemCreateMC(), PmMemFlush(), PxLoadImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PhReleaseTransportCtrl()**

*Free a PhTransportCtrl_t structure*

**Synopsis:**

```c
void PhReleaseTransportCtrl(
    PhTransportCtrl_t *ctrl );
```

**Library:**

`ph`

**Description:**

This function releases the transport control structure pointed to by `ctrl`, as well as any inline data pointed to by the structure.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhCreateTransportCtrl()`, `PhTransportCtrl_t`, `PtCreateTransportCtrl()`, `PtReleaseTransportCtrl()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PhReleaseTransportHdrs()**

*Free a linked list of headers for packed transport data*

**Synopsis:**

```c
void PhReleaseTransportHdrs(
    PhTransportHdr_t *hdrs);
```

**Library:**

`ph`

**Description:**

This function frees the linked list of transport headers pointed to by `hdrs`.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhGetAllTransportHdrs(), PhGetNextTransportHdr(), PhGetTransportHdr(), PhLocateTransHdr(), PhUnlinkTransportHdr()*

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PhSortTiles()**  
Sort a list of tiles

**Synopsis:**

PhTile_t * PhSortTiles( PhTile_t * tiles );

**Library:**

ph

**Description:**

This function sorts the given list of tiles by the y coordinate then the x coordinate. Sorting a list of tiles usually results in a smaller merged list.

Don’t free() the list of tiles; instead, use PhFreeTiles() to return the tiles to the internal pool.

**Returns:**

The same pointer as tiles.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhAddMergeTiles(), PhClipTilings(), PhCoalesceTiles(),  
PhCopyTiles(), PhDeTranslateTiles(), PhFreeTiles(), PhGetTile(),  
PhIntersectTilings(), PhMergeTiles(), PhRectsToTiles(), PhTile_t,  
PhTilesToRects(), PhTranslateTiles()
PhSortTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhSysInfo_t
Data structure for system information

Synopsis:

typedef struct Ph_sys_info {
    PhGeneralSysInfo_t gen;
    PhGrafxInfo_t gfx;
    PhKbdInfo_t kbd;
    PhPtrInfo_t ptr;
    PhIgInfo_t ig;
} PhSysInfo_t;

Description:

The PhSysInfo_t structure contains system information, and is filled in by PhQuerySystemInfo() and PtQuerySystemInfo(). This structure includes at least:

PhGeneralSysInfo_t gen
    General information. Always examine this.

PhGrafxInfo_t gfx
    Information for graphics regions.

PhKbdInfo_t kbd
    Information for keyboard regions.

PhPtrInfo_t ptr
    Information for graphics regions.

PhIgInfo_t ig
    Information for input-group regions.

Always examine the general information gen first, to see which of the other structures contain data.

The fields in the PhSysInfo_t structure each have a valid_fields field that you should check before using the data.

For example, before referring to gfx in the PhSysInfo_t structure, you should check that it’s valid:
PhSysInfo_t

if (sysinfo.gen.valid_fields & Ph_GEN_INFO_NUM_GFX)
{
    /* It’s valid. */
    ...
}

The gen member is a PhGeneralSysInfo_t structure that contains at least:

**ulong_t valid_fields**
Indicates which of the other fields are valid, and can be one or more of the following:
- Ph_GEN_INFO_BANDWIDTH
- Ph_GEN_INFO_CAPABILITIES
- Ph_GEN_INFO_NUM_GFX
- Ph_GEN_INFO_NUM_KBD
- Ph_GEN_INFO_NUM_PTR
- Ph_GEN_INFO_NUM_IG

**ushort_t version**
The version of the Photon server.

**ulong_t bandwidth**
The estimated bandwidth between your process and the Photon server. It can be one of:
- Ph_BAUD_MIN — the minimum possible bandwidth
- Ph_BAUD_SLOW — a slow connection
- Ph_BAUD_NETWORK — a network speed connection
- Ph_BAUD_CONSOLE — a fast connection
- Ph_BAUD_MAX — the fastest possible connection

**ulong_t capabilities**
Not used.
**PhSysInfo_t**

**ushort_t num_gfx**

The number of graphics regions.

**ushort_t num_kbd**

The number of keyboard regions.

**ushort_t num_ptr**

The number of pointer regions.

**ushort_t num_ig**

The number of input-group regions.

**gfx**

The `gfx` member is a `PhGrafxInfo_t` structure that contains at least:

**ulong_t valid_fields**

Indicates which of the other fields are valid, as described below.

**ulong_t reserved1, reserved2[3]**

Reserved for QNX internal use.

**ulong_t bandwidth**

Connection speed (PhRelay only).

**ulong_t capabilities**

Indicates the lowest common denominator graphics rendering capabilities of all graphics drivers in the system. This bitfield can be one or more of the following:

- **Ph_GCAP_BLIT** — the driver supports blitting.
- **Ph_GCAP_DIRECT** — the driver is in direct mode.
- **Ph_GCAP_DIRECTCOLOR** — the current video mode is a direct color mode.
- **Ph_GCAP_DRAW_ALPHA** — the driver supports alpha blending.
PhSysInfo_t

- Ph_GCAP_DRAW_CHROMA — the driver supports chroma keying.
- Ph_GCAP_DRAW_GRADIENTS — the driver supports gradient drawing.
- Ph_GCAP_DRAW_OFFSCREEN — offscreen memory contexts are supported.
- Ph_GCAP_DRAW_TERN_ROPS — the driver supports ternary raster operations.
- Ph_GCAP_FRAME_READ — PgReadScreen() is supported.
- Ph_GCAP_FutureSupported — reserved.
- Ph_GCAP_LOCALHW — the driver region is created by io-graphics running on the current node.
- Ph_GCAP_MASKED_BLIT — the driver supports planemasked blitting.
- Ph_GCAP_NONINTERLACED — the display framebuffer is not interlaced.
- Ph_GCAP_OVERLAY — the driver supports video overlay.
- Ph_GCAP_PALETTE — the driver supports a hardware palette.
- Ph_GCAP_PHINDOWS — the driver region is created by phindows.
- Ph_GCAP_RELAY — the driver region is created by phrelay.
- Ph_GCAP_SHMEM — the driver supports draw data sent in shared memory.
- Ph_GCAP_TEXT_AREA — PgDrawTextArea() is supported.
- Ph_GCAP_VIDEO_READABLE — the display framebuffer can be read linearly.

uchar_t color_bits

The current video mode’s bits per pixel.

ulong_t possibilities

Indicates the best possible graphics rendering capabilities of all graphics drivers in the system. The flags are the same as for capabilities.
The other fields in the PhSysInfo_t structure are similar. For details on these structures, see the <photon/PhT.h> header file.

Classification:
Photon

See also:
PhQuerySystemInfo(), PtQuerySystemInfo()
“System information” in the Regions chapter of the Programmer’s Guide.
**PhTile t**

_A list of rectangles_

**Synopsis:**

```c
typedef struct Ph_tile {
    PhRect_t rect;
    struct Ph_tile *next;
} PhTile_t;
```

**Description:**

The `PhTile_t` structure is used to build linked lists of rectangles. It includes at least the following members:

- `rect` A `PhRect_t` structure that defines the rectangle.
- `next` A pointer to the next tile in the list.

Photon maintains an internal pool of tiles because they’re frequently used, and using a pool reduces the amount of time spent allocating and freeing the tiles. Use `PhGetTile()` to get a tile from the pool, and `PhFreeTiles()` to return a list of tiles to the pool.

**Classification:**

Photon

**See also:**

- `PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`, `PhCopyTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`, `PhGetTile()`, `PhIntersectTilings()`, `PhMergeTiles()`, `PhRectsToTiles()`, `PhSortTiles()`, `PhTilesToRects()`, `PhTranslateTiles()`
- `PhRect_t`

“Geometry data types” in the Working with Code chapter, and “Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PhTilesBoundingRect()

Calculate the bounding box from a list of tiles

Synopsis:

```c
void PhTilesBoundingRect( const PhTile_t *tiles,
                        PhRect_t *r );
```

Arguments:

- `tiles` A list of `PhTile_t` structures you want to calculate a bounding box for. This list can’t be empty.
- `r` A pointer to a `PhRect_t` structure in which the function stores the bounding box.

Library:

- `ph`

Description:

This function takes an array of `PhTile_t` structures and calculates the bounding box for all the tiles in the list. The bounding box is put in the `r` argument.

The list of tiles can’t be empty.

Classification:

- Photon

Safety

- Interrupt handler: No
- Signal handler: No
- Thread: No
**PhTilesBoundingRect()**

See also:

- `PhAddMergeTiles()`, `PhClipTilings()`, `PhCoalesceTiles()`,
- `PhCopyTiles()`, `PhDeTranslateTiles()`, `PhFreeTiles()`, `PhGetTile()`,
- `PhIntersectTilings()`, `PhMergeTiles()`, `PhRect_t`, `PhRectsToTiles()`,
- `PhSortTiles()`, `PhTile_t`, `PhTilesToRects()`, `PhTranslateTiles()`

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
PhTilesToRects()
Create an array of rectangles from a list of tiles

Synopsis:

PhRect_t * PhTilesToRects( PhTile_t * tiles,
                          int *num_rects );

Library:

ph

Description:

This function allocates an array of num_rects PhRect_t structures
and fills it with the rectangles described by the list of tiles pointed to
by tiles.

Don’t free() the list of tiles; instead, use PhFreeTiles() to return the
tiles to the internal pool.

Returns:

A pointer to the array of rectangles.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhAddMergeTiles(), PhClipTilings(), PhCoalesceTiles(),
PhCopyTiles(), PhDeTranslateTiles(), PhFreeTiles(), PhGetTile(),
PhTilesToRects()

PhIntersectTilings(), PhMergeTiles(), PhRect_t, PhRectsToTiles(), PhSortTiles(), PhTile_t, PhTranslateTiles()

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PhTimerArm()**

*Arm a timer event*

**Synopsis:**

```c
int PhTimerArm ( PhRid_t rid,
                long handle,
                unsigned msec );
```

**Library:**

ph

**Description:**

The `PhTimerArm()` arms a timer event on the region specified by `rid` to be triggered after `msec` milliseconds. The `handle` argument is returned in the timer event. See PhT.h for information about the timer event structure.

Don’t use `PhTimerArm()` in an application that uses widgets — use `PtTimerArm()` or `RtTimerCreate()` instead.

**Returns:**

A nonnegative value

  Successful completion.

-1 An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1034 Chapter 9 • Ph—Photon
See also:

`PtTimerArm()`, `RtTimerCreate()`, `RtTimerDelete()`, `RtTimerGetTime()`, `RtTimerSetTime()`

“Timers” in the Working with Code chapter of the Photon Programmer’s Guide
**PhTo8859_1()**

Get the ISO8859-1 value of a key event

**Synopsis:**

```c
int PhTo8859_1( PhKeyEvent_t const *keyevent );
```

**Library:**

`ph`

**Description:**

This function returns a valid ISO8859-1 code, if one exists, for the key event described by the `PhKeyEvent_t` structure pointed to by `keyevent`.

**Returns:**

The ISO8859-1 code, or -1 if there’s no valid code for the given event.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhGetData()`, `PhKeyEvent_t`, `PhKeyToMb()`

`<photon/PkKeyDef.h>`
PhTranslateRect()
Translate a rectangle (add offset)

Synopsis:

PhRect_t *PhTranslateRect( PhRect_t *rect,
            PhPoint_t const *delta );

Library:

ph

Description:

This convenience function adds \texttt{delta->x} to \texttt{rect->ul.x} and \texttt{rect->lr.x},
and adds \texttt{delta->y} to \texttt{rect->ul.y} and \texttt{rect->lr.y}.

You’ll find this function handy for translating events, extents, or
 canvases so that they become relative to various points.

Returns:

The pointer passed in \texttt{rect}.

Examples:

See \texttt{PhInitDrag()}.  

Classification:

Photon

\begin{tabular}{|c|c|}
\hline
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\hline
\end{tabular}
PhTranslateRect()

See also:

PhDeTranslateRect(), PhPoint_t, PhRect_t

“Geometry data types” in the Working with Code chapter of the Photon Programmer’s Guide
PhTranslateTiles()
Add x and y offsets to the vertices of a list of tiles

Synopsis:

PhTile_t * PhTranslateTiles(
   PhTile_t *tile,
   PhPoint_t const *point_add);

Library:

ph

Description:

This function adds the coordinates of point_add to the vertices of each tile in the list pointed to by tile.

Don’t free() the list of tiles; instead, use PhFreeTiles() to return the tiles to the internal pool.

Returns:

The same pointer as tile.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhAddMergeTiles(), PhClipTilings(), PhCoalesceTiles(), PhCopyTiles(), PhDeTranslateTiles(), PhFreeTiles(), PhGetTile,
**PhTranslateTiles()**

PhIntersectTilings(), PhMergeTiles(), PhPoint_t, PhRectsToTiles(), PhSortTiles(), PhTile_t, PhTilesToRects()

“Using damage tiles” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
PhTransportCtrl_t
Control structure for the Photon transport mechanism

Synopsis:

typedef struct phdattransp {
    PhTransportLink_t *first_inline;
    PhTransportLink_t *last_inline;
    int unsigned n_inline;
    int unsigned rqdata_cnt;
} PhTransportCtrl_t ;

Description:

This structure is used when packing data for use with Photon’s transport mechanism. It includes these members:

first_inline A pointer to the beginning of a linked list of inlined data.
last_inline A pointer to the end of the linked list of inlined data.
n_inline The number of items in the linked list.
rqdata_cnt The number of requested data types.

Classification:

Photon

See also:

PhCreateTransportCtrl(), PhGetNextInlineData(),
PhGetTransportVectors(), PhPackEntry(), PhPackType(),
PhReleaseTransportCtrl(), PhTransportLink_t,
PhTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
**PhTransportFindLink()**

Search a linked list of transport data for some specific data

**Synopsis:**

```c
PhTransportLink_t *PhTransportFindLink(
    PhTransportLink_t *link_list,
    void *data );
```

**Library:**

`ph`

**Description:**

This function searches the list of transport data pointed to by `link_list` for the entry containing the given data.

**Returns:**

A pointer to the entry containing the data, or NULL if it couldn’t be found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhCreateTransportCtrl()`, `PhLinkTransportData()`, `PhTransportType()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PhTransportLink_t**  
*Entry in a linked list of transport data*

**Synopsis:**
```c
typedef struct __phlink PhTransportLink_t;
struct __phlink {
    void *data;
    int unsigned size;
    int niovs;
    iov_t *iovs;
    PhTransportLink_t *next;
};
```

**Description:**
This structure is used to build linked lists of transport data, such as the one found in `PhTransportCtrl_t` (see `PhCreateTransportCtrl()`). The members include:

- **data**  
  A pointer to the packed transport data; see below.

- **size**  
  The size of the data, in bytes.

- **niovs**  
  The number of entries in the `iovs` array?

- **iovs**  
  An array of buffers used to store the data; see below.

- **next**  
  A pointer to the next entry in the linked list.

If the data is copied into the structure, the `data` points to the copy. If the data isn’t copied, the `iovs` vectors point to the original data.

**Classification:**
Photon

**See also:**

- `PhCreateTransportCtrl()`, `PhGetNextInlineData()`,
- `PhLinkTransportData()`, `PhReleaseTransportCtrl()`,
- `PhTransportCtrl_t`, `PhTransportFindLink()`, `PhTransportType()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
PhTransportRegEntry_t
Data structure that describes data to be transported

Synopsis:
typedef struct ph_transport_reg_entry {
    char *type;
    int unsigned packing;
    int unsigned size;
    int unsigned num_fixups;
    PhTransportFixupRec_t const *fixups;
    int unsigned const *endians;
} PhTransportRegEntry_t;

Description:
The PhTransportRegEntry_t structure describes how data is to be packed for the Photon transport mechanism, taking into account endian-ness and references to memory outside of the data type.

This structure includes the following:

type The name of the type being registered.

packing The packing method to be used (Ph_PACK_RAW, Ph_PACK_STRING, or Ph_PACK_STRUCT).

size The size, in bytes, of the data type.

num_fixups The number of entries in the fixups arrays.

fixups A list of instructions for dealing with references to data outside the type being defined. For more information, see “Fixup manifests,” below.

endians A zero-terminated array of endian information for the members of the data type. All types or references to types correct the endian-ness of their members based on the endian array defined for the type. For more information, see “Endian information,” below.
**Fixup manifests**

The fixup manifests are:

<table>
<thead>
<tr>
<th>Member type</th>
<th>Fixup Manifest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>None</td>
</tr>
<tr>
<td>Scalar Array</td>
<td>None</td>
</tr>
<tr>
<td>Reference (string)</td>
<td>$Tr_{\text{STRING}}( \text{type, member})$</td>
</tr>
<tr>
<td>Reference (scalar array)</td>
<td>$Tr_{\text{REF ARRAY}}( \text{type, member, number_of_elements})$</td>
</tr>
<tr>
<td>Registered type</td>
<td>$Tr_{\text{TYPE}}( \text{type, member, type_name})$</td>
</tr>
<tr>
<td>Registered type array</td>
<td>$Tr_{\text{TYPE ARRAY}}( \text{type, member, type_name})$</td>
</tr>
<tr>
<td>Reference (registered type)</td>
<td>$Tr_{\text{REF TYPE}}( \text{type, member, type_name})$</td>
</tr>
<tr>
<td>Reference (registered type array)</td>
<td>$Tr_{\text{REF TYPE ARRAY}}( \text{type, member, num_elements, type_name})$</td>
</tr>
<tr>
<td>Reference (registered type reference array)</td>
<td>$Tr_{\text{REF TYPE REF ARRAY}}( \text{type, member, num_elements, type_name})$</td>
</tr>
</tbody>
</table>

**Endian information**

The classifications of endian-sensitive members are:

<table>
<thead>
<tr>
<th>Members of type</th>
<th>Example</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>int, long, or short (signed or unsigned)</td>
<td>unsigned int</td>
<td>$Tr_{\text{ENDIAN}}( \text{typedef_name, member})$</td>
</tr>
<tr>
<td></td>
<td>my_scalar</td>
<td>continued...</td>
</tr>
</tbody>
</table>
**PhTransportRegEntry_t**

<table>
<thead>
<tr>
<th>Members of type</th>
<th>Example</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays of short or int entries</td>
<td><code>short short_nums[10]</code></td>
<td><code>Tr_ENDIAN_ARRAY(typedef_name, member)</code></td>
</tr>
<tr>
<td>References to endian scalars</td>
<td><code>int *nums</code></td>
<td><code>Tr_ENDIAN_REF(typedef_name, member, num)</code></td>
</tr>
</tbody>
</table>

**Classification:**
Photon

**See also:**

- `PhFindTransportType()`, `PhFreeTransportType()`, `PhMallocUnpack()`, `PhPackEntry()`, `PhRegisterTransportType()`, `PhTransportType()`, `PhUnpack()`
- “Registering new transport types” in the Drag and Drop chapter of the Photon *Programmer’s Guide*
PhTransportType()

Pack data into a PhTransportCtrl_t structure

Synopsis:

PhTransportLink_t *
PhTransportType(
    PhTransportCtrl_t *ctrl,
    char const * const type,
    char const * const desc,
    int unsigned const grouping_num,
    int unsigned const handle,
    int unsigned inlined_transport,
    char *packing_type,
    void *vdata,
    int unsigned len,
    int unsigned const flags);

Library:

ph

Description:

This function takes the data referenced by vdata and packs it into a stream buffer within the provided PhTransportCtrl_t structure pointed to by ctrl.

The data is packed using the transport registry entry that matches packing_type. For more information, see PhTransportRegEntry_t.

The provided type, desc, grouping_num, and handle are added to the packed data’s header information.

The PhTransportCtrl_t structure pointed to by ctrl was created via a call to PhCreateTransportCtrl(). The len parameter is used only for packing raw data.

The inlined_transport argument indicates the transport type used for the inlined data, and can be one of:

Ph_TRANSPORT_INLINE

The data being transported is in memory and can be unpacked immediately.
PhTransportType()

Ph_TRANSPORT_FILEREF

The data being transported is in the temporary file(s) named in the inlined data.

Ph_TRANSPORT_SHMEM

The data being transported is in the temporary shared object(s) named in the inlined data.

When used with Photon’s drag and drop mechanism, the grouping_num is used to indicate which data is just a different representation of other data also packed into the same PhTransportCtrl_t. Only one of each grouping_num should be unpacked by the reader/destination.

The type should be a descriptive type name, such as image, text, filename, or files.

Each type has its most common and expected packing_type associated with it. For example:

<table>
<thead>
<tr>
<th>Type</th>
<th>Packing type</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>PhImage</td>
</tr>
<tr>
<td>text</td>
<td>string</td>
</tr>
<tr>
<td>files</td>
<td>files or PhTransFiles</td>
</tr>
</tbody>
</table>

Other packing types can be used, but there’s no guarantee that the reader/recipient of the data will be expecting the type of data packed. In this case, the data is ignored.

The desc should detail the specifics of what is in the data. The extractor uses a regular expression match against the description to determine if the data should be unpacked or discarded.

The flags parameter has the following valid bits:
PhTransportType()

Ph_DONT_COPY

Refer to the data to be transported instead of physically copying the data to the transport control’s stream buffers.

If this flag is used, any modifications to the data that occur between packing and actual transport of the data will be reflected in the data transported.

Returns:

A pointer to a PhTransportLink_t structure (within the transport control’s stream buffer list) that contains the packed data, or NULL if the call has failed (errno is set).

Errors:

EINVAL  One of the following occurred:
• One of the parameters was passed incorrectly.
• No transport control was provided.
• No data was provided via vdata.
• No inlined_transport was specified.
ENOENT  No transport registry entry was found for the provided packing_type.
ENOMEM  There wasn’t enough memory to pack the provided data.

Classification:

Photon

Safety
Interrupt handler  No

continued . . .
**PhTransportType()**

**Safety**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhAllocPackType()`, `PhFindTransportType()`, `PhFreeTransportType()`, `PhMallocUnpack()`, `PhPackEntry()`, `PhRegisterTransportType()`, `PhTransportCtrl_t`, `PhTransportLink_t`, `PhTransportRegEntry_t`, `PhUnpack()`, `PtCreateTransportCtrl()`, `PtDndFetch_t`, `PtDndSelect()`, `PtTransportType()`

Drag and Drop chapter of the Photon *Programmer’s Guide*
PhUnlinkTransportHdr()
Remove an entry from a linked list of transport headers

Synopsis:

PhTransportHdr_t * PhUnlinkTransportHdr(
    PhTransportHdr_t *hdr_list,
    PhTransportHdr_t *victim
);

Library:

ph

Description:

PhUnlinkTransportHdr() unlinks the transport header pointed to by victim from the list pointed to by hdr_list.

Returns:

A pointer to the beginning of the list after removing the victim.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PhGetAllTransportHdrs(), PhGetNextTransportHdr(), PhGetTransportHdr(), PhLocateTransHdr(), PhReleaseTransportHdrs()

Drag and Drop chapter of the Photon Programmer’s Guide
PhUnpack()
Unpack transport data

Synopsis:

```c
char * PhUnpack( PhTransportHdr_t *hdr,
                 void **ret_struct );
```

Library:

ph

Description:

This function unpacks data packed using one of the PhTransport* or PtTransport* functions. Any memory required for the extraction is allocated via malloc().

The hdr parameter describes the data to be unpacked and is itself extracted from the packed data stream. See PhGetTransportHdr().

The pointer referenced by ret_struct is set to point to the newly unpacked data. The actual data representation of the data extracted is described in the hdr that’s passed as the first parameter to this function.

As the data is unpacked, it’s automatically endian-corrected.

Use PhFreeTransportType() to free data unpacked using this function.

Returns:

A pointer to the byte in the data stream following the data just extracted.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
</tbody>
</table>

continued...
PhUnpack()

Safety

| Thread | No |

See also:

PhAllocPackType(), PhFindTransportType(), PhFreeTransportType(), PhMallocUnpack(), PhPackEntry(), PhRegisterTransportType(), PhTransportRegEntry_t, PhTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
PhWindowEvent_t
Data structure describing a window action

Synopsis:

typedef struct _Ph_window_event {
  ulong_t   event_f;
  ulong_t   state_f;
  PhRid_t   rid;
  PhPoint_t pos;
  PhDim_t   size;
  ushort_t  event_state;
  ushort_t  input_group;
  ulong_t   rsvd[4];
} PhWindowEvent_t;

Description:

This structure defines the data that’s associated with a Ph_EV_WM event of subtype Ph_EV_WM_EVENT. For more information, see PhEvent_t.

Using the PhWindowEvent_t structure, your application can determine what kind of window action just occurred, or can tell the window manager to perform a specific action on its behalf.

This structure contains at least the following members:

unsigned long event_f

The type of the window event. The flags you can set in this member are the same as those for Pt_ARG_WINDOW_MANAGED_FLAGS and Pt_ARG_WINDOW_NOTIFY_FLAGS resources of the PtWindow widget (described in the Photon Widget Reference):

- Ph_WM_CLOSE — the window is to be closed.
- Ph_WM_FOCUS — the window is to gain/lose focus.
- Ph_WM_MENU — the window menu is requested or dismissed.
- Ph_WM_TOFRONT — the window is to be moved to the front.
PhWindowEvent_t

- Ph_WM_TOBACK — the window is to be moved to the back.
- Ph_WM_CONSWITCH — the window is to switch consoles.
- Ph_WM_RESIZE — the window is to be resized.
- Ph_WM_MOVE — the window is to be moved.
- Ph_WM_HIDE — the window is to be hidden or unhidden.
- Ph_WM_MAX — the window is to be maximized.
- Ph_WM_BACKDROP — the window is to be made into a backdrop.
- Ph_WM_RESTORE — the window is to be restored.
- Ph_WM_HELP — the help button is pressed.
- Ph_WM_FFRONT — the window is to be made force-front or not force-front.

Some events can have two meanings; see the description of the event_state member.

unsigned long state_f

The current state of the window:
- Ph_WM_STATE_ISNORMAL — a normal window.
- Ph_WM_STATE_ISHIDDEN — window is hidden.
- Ph_WM_STATE_ISMAX — window is maximized.
- Ph_WM_STATE_ISICONIFIED — window is iconified.
- Ph_WM_STATE_INSTASKBAR — window is a taskbar.
**PhWindowEvent_t**

- Ph_WM_STATE_ISBACKDROP — window forms the workspace backdrop.

**PhRid_t rid**  
The ID of the affected region.

**short event_state**  
Indicates that the Window Manager has completed or been asked to complete the requested action. If the event is emitted to the Window Manager, the event is performed by the Window Manager. If an application collects the event, the Window Manager has completed the event.

The following operations have two event states; you can OR one of these states into `event_state`:

- **Ph_WM_FFRONT**—either  
  Ph_WM_EVSTATE_FFRONT or  
  Ph_WM_EVSTATE_FFRONT_DISABLE.
- **Ph_WM_FOCUS**—either  
  Ph_WM_EVSTATE_FOCUS or  
  Ph_WM_EVSTATE_FOCUSLOST.
- **Ph_WM_HIDE**—either  
  Ph_WM_EVSTATE_HIDE or  
  Ph_WM_EVSTATE_UNHIDE.
- **Ph_WM_MENU**—either  
  Ph_WM_EVSTATE_MENU or  
  Ph_WM_EVSTATE_MENU_FINISH.
- **All other toggle events**—either  
  Ph_WM_EVSTATE_PERFORM or  
  Ph_WM_EVSTATE_INVERSE.

**PhPoint_t pos**  
A PhPoint_t structure that defines the position of the window. This member is valid only for  
Ph_WM_BACKDROP, Ph_WM_MAX,  
Ph_WM_MOVE, Ph_WM_RESIZE, and  
Ph_WM_RESTORE events.
PhWindowEvent_t

**PhDim_t size**  A `PhDim_t` structure that indicates the width and height of the window. This member is only for `Ph_WM_BACKDROP`, `Ph_WM_MAX`, `Ph_WM_RESIZE`, and `Ph_WM_RESTORE` events.

**ushort_t input_group**  
The input group associated with the event.

**Classification:**  
Photon

**See also:**  
`PhDim_t`, `PhEvent_t`, `PhPoint_t`, `PtForwardWindowEvent()`, `PtForwardWindowTaskEvent()`

Window Management chapter of the Photon *Programmer’s Guide.*
**PhWindowQueryVisible()**

*Query a visible extent*

**Synopsis:**

```c
int PhWindowQueryVisible( unsigned flag,
                         PhRid_t rid,
                         unsigned input_group,
                         PhRect_t *rect );
```

**Library:**

`ph`

**Description:**

If `rid` is zero, this function calculates the visible extent based on the region type specified in `flag`.

If `rid` is nonzero, `PhWindowQueryVisible()` calculates the visible extent by finding every region intersecting with `rid` that matches the region type specified in `flag`.

The `input_group` argument indicates with which input group the visible extent must intersect. To determine the current input group, call `PhInputGroup()`, passing to it the current event, if any.

You must set at most one of the following bits in `flags`:

- `Ph_QUERY_GRAPHICS`
  - Return a graphics driver rectangle.

- `Ph_QUERY_INPUT_GROUP`
  - Return `input_group`'s rectangle.

- `Ph_QUERY_CONSOLE`
  - Return a console's rectangle. A console is defined as either an input group or a graphics driver, depending on the window manager's Multi-monitor placement option. In other words, this option is equivalent to either `Ph_QUERY_GRAPHICS` or `Ph_QUERY_INPUT_GROUP`, depending on your configuration.
PhWindowQueryVisible()

Ph_QUERY_WORKSPACE
   Return a console’s rectangle minus any reserved space around the edges.

Ph_QUERY_CONSOLE is the default value.
You can OR the following into flags:

Ph_QUERY_EXACT
   The visible extent that the function finds must match both input_group and rid; otherwise, rid is a hint.

Ph_QUERY_IG_POINTER
   Use the current location of input_group’s pointer (rid is ignored).

Ph_QUERY_IG_REGION
   Use input_group’s rectangle.

PhWindowQueryVisible() places the visible extent in the PhRect_t structure pointed to by rect.

Returns:

0   The rect argument is valid.
-1  The rect argument is invalid.

Examples:

Determine the absolute coordinates of the current console:

PhRect_t extent;

if( PhWindowQueryVisible( Ph_QUERY_CONSOLE, 0, input_group, &extent ) == 0 ) {
   printf( "Upper left: (%d,%d) Lower right: (%d,%d)\n",
      extent.ul.x, extent.ul.y,
      extent.lr.x, extent.lr.y );
}
**PhWindowQueryVisible()**

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

**PhRect_t**

Window Management chapter of the Photon *Programmer’s Guide*
Chapter 10

Pi—Images
The functions described in this chapter work with images.
**PiCropImage()**

_Crop an image to the specified boundary_

**Synopsis:**

```c
PhImage_t *PiCropImage( PhImage_t *image,  
PhRect_t const *bounds,  
int flags );
```

**Library:**

`ph`

**Description:**

This function crops a Photon image. It allocates space for the resulting image, leaving the original image untouched.

The `image` argument is a pointer to a `PhImage_t` structure that defines the image to be cropped, while the `bounds` argument points to a `PhRect_t` structure specifying the rectangle within the image to keep.

The `flags` argument dictates special behavior of the cropping process and can take on the following values:

- **Pi_FREE** Free the original image if the cropping succeeds.
- **Pi_SHMEM** Store the newly created image data in shared memory. This is useful if fast rendering of the image is required.

If you set the Pi_FREE flag, the function will free the old image by setting all its release flags and calling `PhReleaseImage()` on it.

**Returns:**

A pointer to the new image on success, or NULL if an error occurred.
Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhCreateImage(), PhImage_t, PhRect_t, PiDuplicateImage(), PiFlipImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PiDuplicateImage()**

*Duplicate an image*

**Synopsis:**

```c
PhImage_t *PiDuplicateImage( PhImage_t *image, int flags );
```

**Arguments:**

- `image` A pointer to the image you want to duplicate.
- `flags` Flags that indicate how the function should behave. They can be one or more of the following values:
  - `Pi_FREE` Free the original image if the duplication succeeds.
  - `Pi_SHMEM` Store the newly created image data in shared memory. This speeds up the rendering of the image if the graphics driver is local.

**Library:**

`ph`

**Description:**

This function creates a duplicate of the Photon image defined in the `PhImage_t` structure pointed to by `image`. `PiDuplicateImage()` does a “deep copy” of the `PhImage_t`, meaning it copies not only the structure itself but also any data associated with it such as the palette and image pixel data.

If you set the `Pi_FREE` flag, the function will free the old image by setting all its release flags and calling `PhReleaseImage()` on it.

**Returns:**

A pointer to the new image on success, or NULL if an error occurred.
PiDuplicateImage()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PhCreateImage(), PhImage_t, PiCropImage(), PiFlipImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PiFlipImage()
Flip all or part of an image

Synopsis:

```c
PhImage_t *PiFlipImage ( PhImage_t *image,
                         PhRect_t const *bounds,
                         int flags );
```

Library:

`ph`

Description:

This function flips a Photon image. It allocates space for the resulting image, leaving the original image untouched.

The `image` argument is a pointer to a `PhImage_t` structure that defines the image to be flipped, while the `bounds` argument points to a `PhRect_t` structure specifying the rectangle within the image to flip. Specifying NULL for `bounds` causes the entire image to be flipped.

The `flags` argument dictates special behavior of the flipping process, and can take on the following values:

- **Pi_HORIZONTAL**: Perform a horizontal flip (may be used in conjunction with Pi_VERTICAL).
- **Pi_VERTICAL**: Perform a vertical flip (may be used in conjunction with Pi_HORIZONTAL).
- **Pi_FREE**: Free the original image if the flip succeeds.
- **Pi_SHMEM**: Store the newly created image data in shared memory. This is useful if fast rendering of the image is required.
PiFlipImage()

If you set the Pi_FREE flag, the function will free the old image by setting all its release flags and calling PhReleaseImage() on it.

Returns:

A pointer to the new image on success, or NULL if an error occurred.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhCreateImage(), PhImage_t, PhRect_t, PiCropImage(), PiDuplicateImage()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PiGetPixel()
Retrieve the value of a pixel within an image

Synopsis:

```
int PiGetPixel( PhImage_t const *image,
                ushort_t x,
                ushort_t y,
                ulong_t *value );
```

Library:

ph

Description:

This function retrieves the value of a pixel within an image.

The `image` argument is a pointer to a `PhImage_t` structure that defines the image to be queried, while the `x` and `y` arguments specify the coordinates of the pixel to be examined.

The `value` argument points to an area of memory where the result of the query is stored. The meaning of this value depends on the type of the image, as follows:

- `Pg_IMAGE_DIRECT_1555`
- `Pg_IMAGE_DIRECT_444`
- `Pg_IMAGE_DIRECT_4444`
- `Pg_IMAGE_DIRECT_555`
- `Pg_IMAGE_DIRECT_565`
- `Pg_IMAGE_DIRECT_888`
- `Pg_IMAGE_DIRECT_8888`

A `PgColor_t` representing the pixel’s color.

- `Pg_IMAGE_PALETTE_BYTE`
- `Pg_IMAGE_PALETTE_NIBBLE`

An index into the image’s palette (0 - 255 for BYTE palettes, 0 - 15 for NIBBLE palettes).

- `Pg_IMAGE_GRADIENT_BYTE`
- `Pg_IMAGE_GRADIENT_NIBBLE`

A number representing the pixel’s intensity (0 - 255 for BYTE gradients, 0 - 15 for NIBBLE gradients).
PiGetPixel()

`Pg_BITMAP_BACKFILL`
`Pg_BITMAP_TRANSPARENT`

0 or 1 representing the pixel’s state.

**Returns:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred because the pixel was out of bounds or the image type wasn’t recognized.</td>
</tr>
</tbody>
</table>

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PgColor_t`, `PhImage_t`, `PiGetPixelFromData()`, `PiGetPixelRGB()`, `PiSetPixel()`, `PiSetPixelInData()`

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PiGetPixelFromData()**

*Retrieve a value from a run of pixels*

**Synopsis:**

```c
int PiGetPixelFromData( char const *data,
                        int type,
                        ushort_t pixel,
                        ulong_t *value );
```

**Library:**

ph

**Description:**

This function is similar to PiGetPixel(), except that PiGetPixelFromData() retrieves the value from a run of pixels pointed to by data, instead of from an image. The type argument indicates the format of the data and must correspond to one of the supported image types, as listed for PhImage_t. The pixel argument is the index of the pixel into the run of data, and value points to an area where you want the result to be stored.

**Returns:**

0 Success.

-1 An error occurred because the pixel was out of bounds or the image type wasn’t recognized.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PiGetPixelFromData()

See also:

PhImage_t, PiGetPixel(), PiGetPixelRGB(), PiSetPixel(), PiSetPixelInData()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PiGetPixelRGB()
Retrieve the RGB value of a pixel within an image

Synopsis:

```c
int PiGetPixelRGB( PhImage_t const *image,
                   ushort_t x,
                   ushort_t y,
                   PgColor_t *value );
```

Library:

ph

Description:

This function retrieves the RGB value of a pixel within an image.

The `image` argument is a pointer to a `PhImage_t` structure that defines the image to be queried, while the `x` and `y` arguments specify the coordinates of the pixel to be examined. The `value` argument points to a `PgColor_t` where the result of the query is stored.

Returns:

0 Success.

-1 An error occurred because the pixel was out of bounds or the image type wasn’t recognized.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

1074 Chapter 10 • Pi—Images
See also:

PiGetPixelRGB()

PgColor_t, PhImage_t, PiGetPixel(), PiGetPixelFromData(), PiSetPixel(), PiSetPixelInData()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PiSetPixel()

Alter the value of a pixel within an image

Synopsis:

```c
int PiSetPixel( PhImage_t *image,
        ushort_t x,
        ushort_t y,
        ulong_t value );
```

Library:

ph

Description:

This function alters the value of a pixel within an image.

The `image` argument is a pointer to a `PhImage_t` structure that defines the image to be modified, while the `x` and `y` arguments specify the coordinates of the pixel to change.

The `value` argument specifies the new value of the pixel, and its meaning depends on the image’s type. See the documentation for `PiGetPixel()` to see how this argument is interpreted.

Returns:

0   Success.

-1   An error occurred because the pixel was out of bounds or the image type wasn’t recognized.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

1076   Chapter 10 • Pi—Images
PiSetPixel()

See also:

PhImage_t, PiGetPixel(), PiGetPixelFromData(), PiGetPixelRGB(), PiSetPixelInData()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
PiSetPixelInData()
Set the value of a pixel in a run of pixels

Synopsis:

```c
int PiSetPixelInData( char *data,
                      int type,
                      ushort_t pixel,
                      ulong_t value );
```

Library:

```
ph
```

Description:

This function is similar to PiSetPixel(), except that PiSetPixelInData() sets the value of a pixel in a run of pixels pointed to by `data`, instead of in an image. The `type` argument indicates the format of the data and must correspond to one of the supported image types, as listed for `PhImage_t`. The `pixel` argument is the index of the pixel into the run of data, and `value` is the new value for the pixel.

Returns:

- 0 Success.
- -1 An error occurred because the pixel was out of bounds or the image type wasn’t recognized.

Classification:

```
Photon
```

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PiGetPixel(), PiGetPixelFromData(), PiGetPixelRGB(), PiSetPixel()

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Chapter 11
Pm—Memory
The functions described in this chapter support memory contexts.
PmMemCreateMC()

Create a memory context

Synopsis:

```c
#include <photon/PhRender.h>

PmMemoryContext_t * PmMemCreateMC(  
    PhImage_t * mc_image,  
    PhDim_t *dim,  
    PhPoint_t *translation );
```

Library:

`ph`

Description:

This function creates a memory context. A memory context is used to draw into a local memory image buffer. You must create a memory context before calling any other Photon Memory (Pm) functions. The memory context provides definition, control, and access to the memory image.

The parameters for this function are:

- `mc_image`  
  The resulting image’s type and dimensions. For more information, see below.

- `dim`  
  A `PhDim_t` structure that defines the source size of the draw stream. If the image dimension is different from the source dimension, any drawing done to the memory context will be scaled as necessary to fit the source dimension exactly into the image dimension.

- `translation`  
  A `PhPoint_t` structure that defines the amount by which the draw stream is translated when being rendered into the memory image buffer.

If the `image` member of the `PhImage_t` structure pointed to by `mc_image` (i.e. `mc_image->image`) is NULL, `PmMemCreateMC()` uses `calloc()` to allocate its own buffer. In this case, `PmMemReleaseMC()` frees the allocated image buffer.
**PmMemCreateMC()**

If `mc_image->image` isn’t NULL, `PmMemCreateMC()` uses it instead of allocating its own buffer. The size of the buffer depends on the type and dimensions specified for `mc_image`. In this case, `PmMemReleaseMC()` doesn’t free the buffer.

If you want the image to be in shared memory, allocate the shared space for the image data, instead of letting `PmMemCreateMC()` do it.

The `mc_image->type` member indicates the type of image that’s generated. The type must be one of:

- Pg_IMAGE_DIRECT_888
- Pg_IMAGE_DIRECT_555
- Pg_IMAGE_DIRECT_565
- Pg_IMAGE_DIRECT_8888
- Pg_IMAGE_PALETTE_BYTE

If the `type` member is `Pg_IMAGE_PALETTE_BYTE` or `Pg_IMAGE_PALETTE_NIBBLE`, the `palette` member is used to define the palette. If the `palette` member is NULL, the default palette is used.

The `image` member of the `PhImage_t` structure filled in by `PmMemFlush()` is a pointer to the `mc_image->image` buffer.

**Returns:**

A pointer to the new memory context, or NULL if there isn’t enough memory to allocate one.

**Examples:**

```c
/* pmmemtobutton.c

This demonstrates how to draw into an image. This example uses the PmMem*() functions to draw into a memory context. When finished drawing, the memory context is then dumped into an image. The image is then used as the image displayed on a button.
```
PmMemCreateMC()

To compile, you must link with the phrender library. For example:

```
qcc -w3 -opmmemtobutton -lphrender -lph pmmemtobutton.c
```

```c
#include <stdlib.h>
#include <mem.h>
#include <photon/PhRender.h>
#include <Pt.h>

void create_image( PhImage_t *image, PhDim_t *dim )
{
    PhPoint_t translation = { 0, 0 }, center, radii;
    PmMemoryContext_t *mc;
    mc = PmMemCreateMC( image, dim, &translation );
    PmMemStart( mc );
    // now all drawing goes into the memory context
    // draw whatever we want to appear in the image
    center.x = dim->w / 2;
    center.y = dim->h / 2;
    radii = center;
    PgSetFillColor( Pg_WHITE );
    PgSetStrokeColor( Pg_RED );
    PgDrawEllipse( &center, &radii, Pg_DRAW_FILL_STROKE );
    PgSetStrokeColor( Pg_GREEN );
    PgDrawILine( 0, 0, dim->w-1, dim->h-1 );
    PmMemFlush( mc, image ); // get the image
    PmMemStop( mc );
    // now all drawing goes to the default drawing context
    PmMemReleaseMC( mc );
}

int main( int argc, char *argv[] )
{
    PhArea_t area = { {80, 20}, {80, 40} };
    PhDim_t dim = { 240, 80 };
    PhImage_t image;
    PtArg_t args[3];
    PtWidget_t *button, *window;
    short bytes_per_pixel = 3;

    if (PtInit(NULL) == -1)


exit(EXIT_FAILURE);
PtSetArg( &args[0], Pt_ARG_WINDOW_TITLE,
"Memory Context Sample", 0 );
PtSetArg( &args[1], Pt_ARG_DIM, &dim, 0 );
if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT,
 2, args)) == NULL)
  PtExit(EXIT_FAILURE);
memset( &image, 0, sizeof(PhImage_t) );
image.type = Pg_IMAGE_DIRECT_888; // 3 bytes per pixel
  // with this type
  // If we want the image to be in shared memory, we must
  // allocate the shared space for the image data, instead
  // of letting PmMemCreateMC() do it.
image.size = dim;
image.image = PgShmemCreate(
  dim.w * dim.h * bytes_per_pixel,
  NULL );
create_image( &image, &area.size );
PtSetArg( &args[0], Pt_ARG_LABEL_TYPE, Pt_IMAGE, 0 );
PtSetArg( &args[1], Pt_ARG_AREA, &area, 0 );
PtSetArg( &args[2], Pt_ARG_LABEL_IMAGE, &image, 0 );
button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT,
  3, args );
PtRealizeWidget( window );
PtMainLoop();

  // Shared memory for the image is cleaned up by an
  // internal function that’s called when the program
  // exits.

  return (EXIT_SUCCESS);
PmMemCreateMC()

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PgShmemCreate(), PgShmemDestroy(), PhDim_t, PhImage_t, PhPoint_t, PmMemFlush(), PmMemReleaseMC()

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PmMemFlush()**

Flush a memory context to its bitmap

**Synopsis:**

```c
#include <photon/PhRender.h>

int PmMemFlush( PmMemoryContext_t *mc,
                PhImage_t *image );
```

**Library:**

```
ph
```

**Description:**

This function forces any unprocessed draw-stream commands to be processed. Then, the `PhImage_t` structure pointed to by `image` is filled in with information from the memory image. The image can then be rendered via any of the `PgDrawImage*()` or `PgDrawPhImage*()` functions.

**Returns:**

- 0 Success.
- -1 An invalid memory context was provided.

**Examples:**

See `PmMemCreateMC()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PhImage_t

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Synopsis:

```c
#include <photon/PhRender.h>

void PmMemReleaseMC( PmMemoryContext_t *mc );
```

Library:

```
ph
```

Description:

This function releases the draw context, and shuts down and frees any resources created by the provided memory context.

This function doesn’t release the image buffer if `PmMemCreateMC()` didn’t create it.

If the provided memory context is active at the time of this call, the default draw context automatically becomes the current draw context.

Examples:

See `PmMemCreateMC()`.

Classification:

```
Photon
```

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PmMemReleaseMC()

See also:

PmMemCreateMC()

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PmMemSetChunkSize()**

Set the increment for growing a memory context’s draw buffer

**Synopsis:**

```c
#include <photon/PhRender.h>

void PmMemSetChunkSize( PmMemoryContext_t *mc,
                          int size );
```

**Library:**

ph

**Description:**

This function sets the increment to be used when growing the memory context’s draw buffer.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PmMemCreateMC(), PmMemReleaseMC(), PmMemSetMaxBufSize(), PmMemSetType()

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PmMemSetMaxBufSize()**

*Set the maximum size of a memory context’s draw buffer*

**Synopsis:**

```c
#include <photon/PhRender.h>

void PmMemSetMaxBufSize( PmMemoryContext_t *mc,
                          int size );
```

**Library:**

*ph*

**Description:**

This function sets the maximum size that the memory context’s draw buffer will grow to. The larger the buffer, the less often a flush will be required, and the faster the application will be. The default size of the draw buffer is 4K.

**Classification:**

*Photon*

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PmMemCreateMC(), PmMemReleaseMC(), PmMemSetChunkSize(), PmMemSetType()*

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PmMemSetType()**

*Set the type of a memory context*

**Synopsis:**

```c
#include <photon/PhRender.h>

void PmMemSetType( PmMemoryContext_t *mc, int type );
```

**Library:**

`ph`

**Description:**

This function sets the type of the memory context pointed to by `mc`.

The valid types are:

- `Pm_PHS_CONTEXT`
  - Renders the draw buffer to image only when necessary (i.e. when the draw buffer is full, or `PmMemFlush()` is explicitly called). Otherwise, the only effect a flush has is to expand the draw buffer when necessary.

- `Pm_IMAGE_CONTEXT`
  - The draw stream is rendered to the image on every flush.

**Classification:**

 Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 11 • Pm—Memory  1095
PmMemSetType()

See also:

PmMemCreateMC(), PmMemReleaseMC(), PmMemSetChunkSize(),
PmMemSetMaxBufSize()

“Flickerless animation” in the Raw Drawing and Animation chapter
of the Photon Programmer’s Guide
Synopsis:

```c
#include <photon/PhRender.h>

PhDrawContext_t * PmMemStart(
    PmMemoryContext_t *mc);
```

Library:

```
ph
```

Description:

This function makes the provided memory context \textit{mc} active. That is, from this point until the memory context is deactivated, everything drawn becomes part of the memory image.

All subsequent Photon draw commands are routed through this memory context until:

- the memory context is made inactive by a call to \textit{PmMemStop()} or \textit{PmMemReleaseMC()}
- a different memory, print, or draw context is made active. In this case, the memory context is automatically deactivated as if \textit{PmMemStop()} had been called

Returns:

A pointer to the previously active draw context, or NULL if the provided context couldn’t be made active — see \textit{errno} for details.

Errors:

ENOMEM There wasn’t enough memory for the context’s work buffers.
**PmMemStart()**

**Examples:**
See *PmMemCreateMC()*.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**
*PmMemCreateMC(), PmMemReleaseMC(), PmMemStop()*

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon *Programmer’s Guide*
**PmMemStop()**

*Deactivate a memory context*

**Synopsis:**
```
#include <photon/PhRender.h>

PhDrawContext_t * PmMemStop(
    PmMemoryContext_t *mc
);
```

**Library:**
```
ph
```

**Description:**
This function deactivates the memory context `mc`, if it’s active, making the default draw context active. This means that draw commands are sent to Photon (i.e. draws will no longer be affecting the memory image).

**Returns:**
The `mc` argument if the memory context was successfully deactivated, or NULL if it wasn’t active at the time of this call.

**Examples:**
See *PmMemCreateMC()*.

**Classification:**
```
Photon
```

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PmMemCreateMC(), PmMemReleaseMC(), PmMemStart()

“Flickerless animation” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Chapter 12

Pp—Printing
This chapter describes the functions that Photon provides to support a wide range of printing needs.
**PpContinueJob()**

*Continue a suspended print job*

**Synopsis:**

```c
PhDrawContext_t *PpContinueJob(
    PpPrintContext_t *pc
);
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

**Library:**

`ph`

**Description:**

This function makes the provided print context `pc` active (i.e. from this point until the print context is deactivated, everything drawn is part of the printed output). The print context is initialized if this hasn’t already been done by a call to `PpStartJob()`.

All subsequent Photon draw commands are routed through this print context until:

- the print context is made inactive by a call to `PpSuspendJob()` or `PpEndJob()`
  Or:
- a different memory or print context is made active. In this case, the print context is automatically deactivated as if `PpSuspendJob()` had been called.

**Returns:**

A pointer to the previously active draw context, or NULL if the print context couldn’t be made active—see `errno` for the specific error.
Errors:

ESRCH No output target is specified in the print context and no printer definition could be found.

Examples:

To print the contents of a scroll area:

```c
int print_it( PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo )
{
    PhDim_t dim;

    pc = PpCreatePC();

    // Set the default printer for this app only.
    // This overrides the default printer set via
    // the "prsetup" utility
    PpSetPC( pc, Pp_PC_NAME, "R&D", 0 );

    // Pop up the standard print dialog and respond accordingly.
    switch( PtPrintSelection( ABW_base, &pos, "Select Printer", pc, 0 ) )
    {
        // The user has selected print or preview -- PpEndJob()
        // handles the difference.
        case Pt_PRINTSEL_PRINT:
        case Pt_PRINTSEL_PREVIEW:
            PtFlush(); // Ensure no draws are pending.
            PtWidgetDim( PtValidParent( ABW_my_scrollarea ), &dim );
            PpSetPC( pc, Pp_PC_SOURCE_SIZE, &dim, 0 );
            PpStartJob( pc );
            if( PpContinueJob( pc ) )
            {
                // Force the canvas of the ScrollArea widget
                // to draw.
                PtDamageWidget( PtValidParent( ABW_my_scrollarea ) );
                PtFlush();
            }
            break;
    }
}
```
PpContinueJob()

}  

// Deactivate the pc and produce the printed output.
PpEndJob( pc );

    case Pt_PRINTSELCANCEL:
        break;
    }

// Release the pc and its resources.
PpReleasePC( pc );

    return Pt_CONTINUE;
}

Classification:

    Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Printing in the Photon Programmer’s Guide
Synopsis:

\[
PpCreatePC() \rightarrow \text{PpPrintContext_t *PpCreatePC( void )};
\]

Library:

\[
\text{ph}
\]

Description:

This function creates a print context. You must create a print context before calling any other Photon Print (Pp) functions.

The print context describes all aspects of a print job required by the printing functions and the print drivers: \texttt{Pp pcl}, \texttt{Pp ps}, \texttt{Pp epc2}, and so on.

Returns:

A pointer to the \texttt{PpPrintContext_t} structure that describes the new print context, or NULL if there isn’t enough memory to allocate one.

Examples:

See \texttt{PpContinueJob()}.  

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PpCreatePC()**

See also:


Printing chapter of the *Programmer’s Guide*
Synopsis:

```
int PpEndJob( PpPrintContext_t *pc );
```

Arguments:

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

Library:

- `ph`

Description:

This function completes the current print job for the provided print context. If the print context is active, it’s deactivated as if `PpSuspendJob()` had been called.

The print context is deactivated, but isn’t destroyed, so it can be used for future print jobs with little or no reconfiguration. To release the print context, call `PpReleasePC()`.

When this function returns, the printed output has been generated and sent to the destination specified in the print context.

Returns:

- 0 Success.
- -1 The print context couldn’t be made active, probably because the required print driver couldn’t be launched. See `errno` for the specific error.

Examples:

`See PpContinueJob()`.
**PpEndJob()**

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


Printing in the Photon *Programmer’s Guide*
Synopsis:

```c
void PpFreePrinterList( char **list );
```

Arguments:

`list` A list of available printers that `PpLoadPrinterList()` previously created.

Library:

`ph`

Description:

This function frees all allocated memory in the given list of printers.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PpLoadDefaultPrinter()`, `PpLoadPrinter()`, `PpLoadPrinterList()`

Printing chapter of the *Programmer’s Guide*
**PpGetCanvas()**

Get the size of the current drawing area of a print context

**Synopsis:**

```c
PhDim_t PpGetCanvas( PpPrintContext_t *pc );
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

**Library:**

`ph`

**Description:**

This function gets the size of the current drawing area associated with the given print context, taking into account the nonprintable area, margins, scale, and either the current source resolution or size.

If neither a source resolution nor source size has been set, 100 dpi is used, and `PpGetCanvas()` fills in the `Pp_PC_SOURCE_RESOLUTION` member of the print context.

**Returns:**

A `PhDim_t` structure that defines the dimensions of the rectangle, or `(0, 0)` if `pc` is invalid.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1112 Chapter 12 • Pp—Printing
See also:

PhDim_t, PpCreatePC(), PpGetPC(), PpPrintContext_t, PpReleasePC(), PpSetCanvas(), PpSetPC()

Printing chapter of the Programmer’s Guide
**PpGetPC()**

*Extract data from a print context*

**Synopsis:**

```c
void * PpGetPC( PpPrintContext_t *pc,
                 int member,
                 const void ** const data);
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.
- `data` the address of a pointer to the data type of the member being queried. This pointer will be set to point to the member within the print context structure. Don’t use it to modify the print context; use only `PpSetPC()`.
- `member` The member of the print context to query, as given in the table below. For a description of the members, see `PpPrintContext_t`.

**Library:**

`ph`

**Description:**

Use this function to query the attributes of a print context.

Don’t extract values directly from the data structure. Your application might not work if the structure changes in the future.

Use the following data types when getting the value of the members of the print context:
### Member Data

<table>
<thead>
<tr>
<th>Member</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pp_PC_COLLATING_MODE</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_COLOR_MODE</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_CONTROL</td>
<td>Address of a PpPCControl_t *</td>
</tr>
<tr>
<td>Pp_PC_COPIES</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_DATE</td>
<td>Address of a char * (string)</td>
</tr>
<tr>
<td>Pp_PC_DEVICE</td>
<td>Address of a char * (string)</td>
</tr>
<tr>
<td>Pp_PC_DITHERING</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_DO_PREVIEW</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_DRIVER</td>
<td>Address of a char * (string)</td>
</tr>
<tr>
<td>Pp_PC_DUPLEX</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_FILENAME</td>
<td>Address of a char * (string)</td>
</tr>
<tr>
<td>Pp_PC_INKTYPE</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_INTENSITY</td>
<td>Address of a ulong_t *</td>
</tr>
<tr>
<td>Pp_PC_JOB_NAME</td>
<td>Address of a char * (string)</td>
</tr>
<tr>
<td>Pp_PC_MARGINS</td>
<td>Address of a PhRect_t *</td>
</tr>
<tr>
<td>Pp_PC_MAX_DEST_SIZE</td>
<td>Address of a ulong_t</td>
</tr>
<tr>
<td>Pp_PC_NAME</td>
<td>Address of a char * (string)</td>
</tr>
<tr>
<td>Pp_PC_NONPRINT_MARGINS</td>
<td>Address of a PhRect_t *</td>
</tr>
<tr>
<td>Pp_PC_ORIENTATION</td>
<td>Address of a char * (value)</td>
</tr>
<tr>
<td>Pp_PC_PAGE_NUM</td>
<td>Address of a ulong_t</td>
</tr>
<tr>
<td>Pp_PC_PAGE_RANGE</td>
<td>Address of a PpPageRange_t *</td>
</tr>
</tbody>
</table>

*continued*
**PpGetPC()**

<table>
<thead>
<tr>
<th>Member</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pp_PC_PAPER_SIZE</td>
<td>Address of a <code>PhDim_t *</code></td>
</tr>
<tr>
<td>Pp_PC_PAPER_SOURCE</td>
<td>Address of a <code>char * (value)</code></td>
</tr>
<tr>
<td>Pp_PC_PAPER_TYPE</td>
<td>Address of a <code>char * (value)</code></td>
</tr>
<tr>
<td>Pp_PC_PREVIEW_APP</td>
<td>Address of a <code>char * (string)</code></td>
</tr>
<tr>
<td>Pp_PC_PRINTER_RESOLUTION</td>
<td>Address of a <code>PhDim_t *</code></td>
</tr>
<tr>
<td>Pp_PC_PROP_APP</td>
<td>Address of a <code>char * (string)</code></td>
</tr>
<tr>
<td>Pp_PC_REVERSED</td>
<td>Address of a <code>char (value)</code></td>
</tr>
<tr>
<td>Pp_PC_SCALE</td>
<td>Address of a <code>PhPoint_t *</code></td>
</tr>
<tr>
<td>Pp_PC_SOURCE_COLORS</td>
<td>Address of a <code>ulong_t *</code></td>
</tr>
<tr>
<td>Pp_PC_SOURCE_OFFSET</td>
<td>Address of a <code>PhPoint_t *</code></td>
</tr>
<tr>
<td>Pp_PC_SOURCE_RESOLUTION</td>
<td>Address of a <code>PhDim_t *</code></td>
</tr>
<tr>
<td>Pp_PC_SOURCE_SIZE</td>
<td>Address of a <code>PhDim_t *</code></td>
</tr>
<tr>
<td>Pp_PC_USER_ID</td>
<td>Address of a <code>char * (string)</code></td>
</tr>
</tbody>
</table>

**Returns:**

A pointer to the requested data, or NULL if an unrecognized member was specified.

**Examples:**

```c
int get_it( PtWidget_t *widget, ApInfo_t *apinfo, 
            PtCallbackInfo_t *cbinfo )
{

    PhDim_t *dim;
    void *pc_data;
    PpPrintContext_t *pc;

    /* Eliminate ‘unreferenced’ warnings */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    pc = PpCreatePC();
```

1116   Chapter 12 • Pp—Printing
PpGetPC()

// Pop up the standard print dialog to fill in the PC
PtPrintSelection( NULL, NULL,
    "Select Printer", pc, 0 );

// Get some stuff from the pc

// A string:
PpGetPC( pc, Pp_PC_NAME, &pc_data );
printf( "printer: %s\n", (char *)pc_data );

// A structure ( PhDim_t ):
PpGetPC( pc, Pp_PC_PAPER_SIZE, &pc_data );
printf( "paper height: %d, width: %d\n",
    ((PhDim_t *)pc_data)->h, ((PhDim_t *)pc_data)->w );

// A long value:
PpGetPC( pc, Pp_PC_INTENSITY, &pc_data );
printf( "intensity: %ld\n", *(long *)pc_data );

// A number stored in a char:
PpGetPC( pc, Pp_PC_COPIES, &pc_data );
printf( "copies : %d\n", *(char *)pc_data );

// Of course, the correct type can be used to
// get the member:
PpGetPC( pc, Pp_PC_PAPER_SIZE, &dim );
printf( "paper height: %d, width: %d\n", dim->h, dim->w );
PpReleasePC( pc );

return Pt_CONTINUE;

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 12 • Pp—Printing 1117
PpGetPC()

See also:


Printing in the Photon Programmer’s Guide


**PpLoadDefaultPrinter()**

Initialize a print context with information for the default printer

**Synopsis:**

```c
int PpLoadDefaultPrinter( PpPrintContext_t *pc );
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

**Library:**

`ph`

**Description:**

This function initializes the provided print context with the information found in `$HOME/.ph/printers/default` for the default printer.

**Returns:**

- `0`: Success.
- `-1`: No default printer could be found, or the printer definition that was loaded didn’t define a destination device or filename, so no output can be generated.

**Examples:**

```c
PpPrintContext_t *pc = PpCreatePC();
PpLoadDefaultPrinter( pc );
PpStartJob( pc );
PpContinueJob( pc );
PpContinueJob( pc );

// Draw stuff

PpSuspendJob( pc );
PpEndJob( pc );
PpReleasePC( pc );
```
**PpLoadDefaultPrinter()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


Printing chapter of the *Programmer’s Guide*
**PpLoadPrinter()**

Initialize a print context with information for a given printer

**Synopsis:**

```c
int PpLoadPrinter( PpPrintContext_t *pc,
                   char const *name );
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.
- `name` The name of the printer whose attributes you want to use to initialize the print context. If `name` is `NULL`, this function loads the attributes of the default printer as specified in `$HOME/.ph/printers/default`.

**Library:**

`ph`

**Description:**

This function initializes the provided print context with information for the printer section named `name`.

**Returns:**

- `0` Success.
- `-1` No name was specified and no default printer could be found, or the printer definition loaded didn’t define a destination device or filename, so no output can be generated.

**Examples:**

```c
PpPrintContext_t *pc = PpCreatePC();
PpLoadPrinter( pc, "GenericPostScriptPrinter@localhost");
PpStartJob( pc );

PpContinueJob( pc );

// Draw stuff
PpSuspendJob( pc );
```
PpLoadPrinter()

PpEndJob( pc );
PpReleasePC( pc );

Classification:
Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PpContinueJob(), PpCreatePC(), PpEndJob(), PpGetPC(),
PpLoadDefaultPrinter(), PpPrintContext_t, PpPrintNewPage(),
PpPrintWidget(), PpReleasePC(), PpSetPC(), PpStartJob(),
PpSuspendJob(), PtPrintSelection()

Printing chapter of the Programmer’s Guide
Synopsis:

char **PpLoadPrinterList( void );

Library:

ph

Description:

This function loads a null-terminated list of the names of printers that are available for printing.

Use PpFreePrinterList() to free the list.

Returns:

A pointer to the list.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PpFreePrinterList(), PpLoadDefaultPrinter(), PpLoadPrinter()

Printing chapter of the Programmer’s Guide
**PpPrintContext_t**

*Data structure describing a print context*

**Synopsis:**

See below.

**Description:**

The `PpPrintContext_t` structure describes a print context. Its members control how printing is to be done. `PpPrintContext_t` is an opaque datatype. Use `PpGetPC()` to extract members, and `PpSetPC()` to change them.

The possible values and meanings for the members of a print context are defined in `/usr/photon/print/printers`.

A print context includes the data described in the sections that follow. Each section indicates the type of data, but see `PpGetPC()` and `PpSetPC()` for details.

**Pp_PC_COLLATING_MODE**

Data type: `char`

The order of pages that are printed in print jobs that are printing more than one copy. The modes may include:

- 1-2-3, 1-2-3, ...
- 1-1-1, 2-2-2, ...

**Pp_PC_COLOR_MODE**

Data type: `char`

The color mode to use — you might want to print in black and white on a color printer. Possible meanings:

- black & white
- color
PpPrintContext_t

Pp_PC_CONTROL (read-only)

Data type: PpPCControl_t

This member is the control structure for the print context.

You can’t use PpSetPC() to set Pp_PC_CONTROL.

The control structure has at least the following members:

char changed_prop[Pp_PC_FLAGSIZE]

This array is treated as a large bit field that indicates which portions of the context have been modified. There’s a bit for each of the modifiable attributes of a print context.

This bit field can be manipulated with the following macros, which are defined in <photon/PpT.h>:

- Pp_clearbit()
- Pp_resetbits()
- Pp_setbit()
- Pp_testbit()

For example:

if( Pp_testbit( control-gt;changed_prop, Pp_PC_NAME) )
    printf( "Print name has been changed\n");

char emitted_prop[Pp_PC_FLAGSIZE]

A large bit field that indicates which of the changed print context attributes have already been written to the destination or temporary file. For example:

if( Pp_testbit( control-gt;emitted_prop, Pp_PC_NAME) )
    printf( "source offset has been emitted\n");
PpPrintContext_t

ulong_t locked_prop[Pp_PC_FLAGSIZE]
A large bitfield that indicates which context attributes are locked. The printer properties application displays locked attributes as ghosted controls, and PpSetPC() won’t let you change them.

int fd
The file descriptor for the current print job’s working file. The working file isn’t opened until PpStartJob() opens the print job. This member is -1 if no files are currently open.

char *tmp_target
The temporary working file that’s used if do_preview is set.

char do_preview
If this is nonzero, the application specified in the print context’s Pp_PC_PREVIEW_APP member is launched when the print job is finished (that is, when PpEndJob() is called).

Pp_PC_COPIES
Data type: char
The number of copies to produce.

Pp_PC_DATE
Data type: char []
The date the print job was started. It’s filled in when the print job is initialized. See PpStartJob() and PpContinueJob().

Pp_PC_DEVICE
Data type: char []
Usually the spooler to use (e.g. /dev/spool/Ph.main_deskjet). If you’re sending the print job directly to a printer (such as /dev/par1), set Pp_PC_FILENAME to be the name of the printer
PpPrintContext_t

(e.g. /dev/par1) and Pp_PC_DRIVER to be the name of the Photon print driver that produces output that the printer recognizes.

If both the device and filename members are set, the output goes to the destination identified by the filename.

Pp_PC_DITHERING

Data type: char

The type of dithering. Possible meanings are defined in the printer configuration file, and include:

- high-speed QNX dithering
- error diffusion
- half toning.

Pp_PC_DO_PREVIEW

Data type: char

If nonzero, the preview application is launched rather submitting the job to a spooler.

Pp_PC_DRIVER

Data type: char[]

The print filter, such as phs-to-pcl, to launch if printing to a file (if a filename is specified). For information about the available print drivers, see “Photon print drivers (phs-to-*)” in the summary of the Utilities Reference.

Pp_PC_DUPLEX

Data type: char

Specifies whether (1) or not (0) to do double-sided printing.
PpPrintContext_t

Pp_PC_FILENAME

Data type: char[

If this member is specified, the final output of the print job is placed in this file instead of being sent to a device.

For raw phs output, you need to pass the file through a spooler device. There’s a simple phs-to-phs filter (similar to cat) for this purpose.

Pp_PC_INKTYPE

Data type: char

The type of ink. Possible meanings:

- black & white cartridge
- color cartridge
- six-color printing.

Pp_PC_INTENSITY

Data type: ulong_t

The printing intensity, expressed as a value between 0% and 100%. A printer’s default intensity is 50%.

Pp_PC_JOB_NAME

Data type: char[

The name of the print job, for identification purposes.

Pp_PC_MARGINS

Data type: PhRect_t

A rectangle that specifies the margins to apply to the pages, in 1/1000ths of an inch:

- ul.x — left margin
- ul.y — top margin
PpPrintContext_t

- \( lr.x \) — right margin
- \( lr.y \) — bottom margin

The orientation of the page doesn’t affect the margins.

**Pp_PC_MAX_DEST_SIZE**

Data type: `ulong_t`

The maximum size of the temporary \texttt{phs} file, in bytes.

**Pp_PC_NAME**

Data type: `char[]`

The name of the printer (e.g. \texttt{R&D main printer}).

**Pp_PC_NONPRINT_MARGINS**

Data type: `PhRect_t`

A rectangle that specifies (in 1/1000ths of an inch) the nonprintable margins of the printer.

**Pp_PC_ORIENTATION**

Data type: `char`

Possible meanings include portrait and landscape, as defined in the printer-configuration file.

**Pp_PC_PAGE_NUM**

Data type: `ulong_t`

The number of the current page being printed.

**Pp_PC_PAGE_RANGE**

Data type: `PpPageRange_t`

The range of pages to be printed. The application should use this information when producing the printed output; only the requested pages need to be printed.
PpPrintContext_t

The page range is a `PpPageRange_t` structure with two `int` members, `from` and `to`. Special meanings for the range are:

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>Print:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>All</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>Selected region</td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td>From page n to the end of the document</td>
</tr>
</tbody>
</table>

The library assumes that the first page printed is the first requested page. If this isn’t the case, use `Pp_PC_PAGE_NUM` to set the page number manually.

**Pp_PC_PAPER_SIZE**

Data type: `PhDim_t`

The dimensions of the paper, including margins and the nonprintable area, in 1/1000ths of an inch. This size is used for clipping and for any scaling that may be applied.

**Pp_PC_PAPER_SOURCE**

Data type: `char`

The paper source, for printers that support more than one paper tray. The possible meanings are:

- don’t care
- automatic
- upper tray
- lower tray
- manual feed
**PpPrintContext_t**

**Pp_PC_PAPER_TYPE**

Data type: **char**

The type of paper. This is in the range 0 through 100. Possible meanings include:

- unknown
- normal quality
- high quality
- draft quality
- transparency

**Pp_PC_PREVIEW_APP**

Data type: **char[]**

The application to launch to preview the print job, usually /usr/photon/bin/preview.

**Pp_PC_PRINTER_RESOLUTION**

Data type: **PhDim_t**

The printing mode, specified in x and y dots per inch. The printer tries to print at this resolution.

**Pp_PC_PROP_APP**

Data type: **char[]**

The application to launch to adjust the printer properties portion of the print context.

**Pp_PC_REVERSED**

Data type: **char**

If this is nonzero, the pages are printed in reverse order. The default is forward.
**PpPrintContext_t**

**Pp_PC_SCALE**

Data type: `PhPoint_t`

The scale to use in x and y:

- 0—maintain the aspect ratio.
- Positive—the scale as a percentage of page size (1/10th percent). For example, 1000 = 100%.
- Negative—the scale as a percentage of source size (1/10th percent). This is useful when printing to a bitmap file, where there’s no “page size.” For a one-to-one mapping of source to destination pixels, use a scaling of -1000.

If the x and y scales are both 0, the source is scaled to be as large as possible to fit on the page and still maintain its x and y aspect ratio.

**Pp_PC_SOURCE_COLORS**

Data type: `ulong_t`

The number of bitplanes per source pixel. If this is \(n\), you’ll have \(2^n\) colors. For example, a value of 8 means 256 colors. This is a hint that helps the print drivers improve the dithering model. The default is 24bpp (True Color).

**Pp_PC_SOURCE_OFFSET**

Data type: `PhPoint_t`

The origin for the print job. For example, if a widget you want to print is at (50,50) but you want it to appear in the upper left corner of the page, set the source offset to (50,50).

This offset affects the entire page; to affect an individual widget, use the `trans` argument to `PpPrintWidget()`.
PpPrintContext_t

Pp_PC_SOURCE_RESOLUTION

Data type: PhDim_t

The resolution, in pixels per inch, of the original source image. This is a hint that helps the print drivers do the best scaling possible. The default is 100 pixels per inch.

Pp_PC_SOURCE_SIZE

Data type: PhDim_t

The dimension of the source image, in pixels. This size is used for clipping and for any scaling that may be applied.

Pp_PC_USER_ID (read-only)

Data type: char[]

The user creating the print job. This member is filled in when the print job is initialized. See PpStartJob() and PpContinueJob().

Classification:

Photon

See also:

PpContinueJob(), PpGetPC(), PpPrintWidget(), PpSetPC(), PpStartJob()

“Photon print drivers (phs-to-*)” in the summary of the Utilities Reference
**PpPrintNewPage()**

*Place a page break in the draw stream for a Print Context*

**Synopsis:**

```
int PpPrintNewPage( PpPrintContext_t *pc );
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

**Library:**

`ph`

**Description:**

This function places a New Page command in the draw stream for the specified print context, followed by all the changes made to the print context since `PpPrintNewPage()` was called or the print context was opened (initialized).

If the print context isn’t currently active, `PpContinueJob()` is called before the New Page command is inserted. The draw context that was active when the call was made is restored before `PpPrintNewPage()` returns.

**Returns:**

- `0` Success.
- `-1` The print context couldn’t be made active, probably because the required working files couldn’t be created. See `errno` for the specific error.

**Classification:**

Photon

---

1134 Chapter 12 • Pp—Printing
PpPrintNewPage()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Printing in the Photon Programmer’s Guide
**PpPrintWidget()**

*Print a widget*

**Synopsis:**

```c
void PpPrintWidget( PpPrintContext_t *pc,
                   PtWidget_t *widget,
                   PhPoint_t const *trans,
                   PhRect_t const *clip_rect,
                   ulong_t resize );
```

**Arguments:**

- `pc` A print context obtained via `PpCreatePC()` and initialized via `PpSetPC()` and `PpStartJob()`.
- `widget` The widget to be printed. This widget doesn’t need to be realized to be printed and won’t be clipped by its parent while printing.
- `trans` If non-NULL, this argument points to a `PhPoint_t` structure that’s used to set the print context’s offsets. These offsets define the amounts by which to translate widgets when drawing them into the print context. Specifying a translation equal to the position of the widget makes the widget print at 0,0 on the printed output.
- `clip_rect` If non-NULL, a pointer to a `PhRect_t` structure that defines the rectangle to be clipped to.
- `resize` A value that indicates any special resizing to be done:
  - `Pt_PP_RESIZE_WIDGET`—set the dimension of the widget to match the drawable area of the destination page. The widget is resized to fit the page.
  - `Pt_PP_NO_RESIZE`—don’t modify the source size of the print context or the widget’s dimensions. It’s important to set the source size of the print context before calling `PpPrintWidget()` with this option.

---

This isn’t implemented yet; set `clip_rect` to NULL.

---

1136 Chapter 12 • Pp—Printing
PpPrintWidget()

- Pt_PP_RESIZE_PC — set the source size of the print context to the size of the widget. The result is scaled to fit the destination page, but the aspect ratio of the widget is preserved.

Library:

ph

Description:

This function prints the specified widget using the provided print context. The widget doesn’t need to be realized in order to be printed.

Examples:

```c
#include <Ph.h>
#include <Pt.h>

int
main()
{
  int n,
  do_preview_only = 1;
  PhArea_t area, sev_area = {{ 0,0 },{400,400}};
  PpPrintContext_t *pc;
  PtArg_t args[10];
  PtWidget_t *window, *button;
  PhDim_t  dim = { 750, 1000 };

  PtInit( NULL );
  pc = PpCreatePC();

  /* Set override on print context to do preview mode only */
  PpSetPC( pc, Pp_PC_DO_PREVIEW, &do_preview_only, 0 );
  PpStartJob( pc );

  PtSetArg( &args[0], Pt_ARG_AREA, &sev_area, 0 );
  window = PtCreateWidget( PtWindow, Pt_NO_PARENT, 1, args );

  n = 0;
  PtSetArg( &args[n++], Pt_ARG_AREA, &sev_area, 0 );
  PtSetArg( &args[n++], Pt_ARG_FILL_COLOR, Pg_BLUE, 0 );
  PtSetArg( &args[n++], Pt_ARG_TEXT_STRING, "Test Button", 0 );
  button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, n, args );
```

Chapter 12 • Pp—Printing 1137
**PpPrintWidget()**

```c
PtRealizeWidget( window );
PpContinueJob( pc );
PtWidgetArea( button, &area);
PpSetCanvas( pc, dim );
PpPrintWidget( pc, button, &area.pos, NULL, Pt_PP_NO_RESIZE );
PpEndJob( pc );
PtMainLoop();
return EXIT_SUCCESS;
}
```

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


Printing in the Photon *Programmer’s Guide*
Synopsis:

```c
void PpReleasePC( PpPrintContext_t *pc );
```

Arguments:

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

Library:

- ph

Description:

This function releases the draw context, shuts down, and frees the resources used by the provided print context.

If the provided print context’s draw context is the current draw context, the default draw context automatically becomes the current draw context.

Examples:

See `PpContinueJob()`.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PpReleasePC()

See also:

PpContinueJob(), PpCreatePC(), PpEndJob(), PpGetCanvas(),
PpGetPC(), PpPrintContext_t, PpPrintNewPage(),
PpPrintWidget(), PpSetCanvas(), PpSetPC(), PpStartJob(),
PpSuspendJob()

Printing in the Photon Programmer's Guide
**PpSetCanvas()**

Set the size of the drawing area for a print context

**Synopsis:**

```c
PhDim_t PpSetCanvas( PpPrintContext_t *pc,
                     PhDim_t size );
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.
- `size` A `PhDim_t` structure that specifies the size of the drawing area.

**Library:**

```
ph
```

**Description:**

This function sets the size of the current drawing area associated with the given print context.

**Returns:**

The dimensions of the rectangle, or (0, 0) if `pc` is invalid.

**Classification:**

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 12 • Pp—Printing 1141
PpSetCanvas()

See also:

PhDim_t, PpCreatePC(), PpGetCanvas(), PpGetPC(), PpPrintContext_t, PpReleasePC(), PpSetPC()

Printing chapter of the Programmer’s Guide
Synopsis:

```c
int PpSetPC( PpPrintContext_t *pc,
             int member,
             void const * const data,
             int lock);
```

Arguments:

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.
- `member` The member of the print context to modify, as given in the table below. For a description of the members, see `PpPrintContext_t`.
- `data` A pointer to the data to be assigned to the specified print context member.
- `lock` If nonzero, the `PtPrintSel` widget and properties applications won’t override or allow modification of the member; its value is locked. Locked members have their controls dimmed and inactive in the `PtPrintSel` widget or `PtPrintSelection()` dialog.

Library:

`ph`

Description:

This function provides a mechanism to modify the attributes of a print context.

Don’t modify the print context directly, as the appropriate changed bits won’t be set and the application may stop working if the print context structure is redefined in the future.

Use the data types given below when setting the value of the members of the print context.
In the following table:

- “String” indicates a `char *` pointer to a null-terminated sequence of characters.
- “char *” indicates a `char *` pointer to a value in the range `0x0` to `0xFF`.

<table>
<thead>
<tr>
<th>Member</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pp_PC_COLLATING_MODE</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_COLOR_MODE</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_CONTROL</td>
<td>Read only: see <code>PpGetPC()</code></td>
</tr>
<tr>
<td>Pp_PC_COPIES</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_DATE</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_DEVICE</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_DITHERING</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_DO_PREVIEW</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_DRIVER</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_DUPLEX</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_FILENAME</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_INKTYPE</td>
<td><code>char *</code></td>
</tr>
<tr>
<td>Pp_PC_INTENSITY</td>
<td><code>ulong_t *</code></td>
</tr>
<tr>
<td>Pp_PC_JOB_NAME</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_MARGINS</td>
<td><code>PhRect_t *</code></td>
</tr>
<tr>
<td>Pp_PC_MAX_DEST_SIZE</td>
<td><code>ulong_t</code></td>
</tr>
<tr>
<td>Pp_PC_NAME</td>
<td>String</td>
</tr>
</tbody>
</table>

continued...
### PpSetPC()

<table>
<thead>
<tr>
<th>Member</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pp_PC_NONPRINT_MARGINS</td>
<td>PhRect_t *</td>
</tr>
<tr>
<td>Pp_PC_ORIENTATION</td>
<td>char *</td>
</tr>
<tr>
<td>Pp_PC_PAGE_NUM</td>
<td>ulong_t</td>
</tr>
<tr>
<td>Pp_PC_PAGE_RANGE</td>
<td>PpPageRange_t *</td>
</tr>
<tr>
<td>Pp_PC_PAPER_SIZE</td>
<td>PhDim_t *</td>
</tr>
<tr>
<td>Pp_PC_PAPER_SOURCE</td>
<td>char *</td>
</tr>
<tr>
<td>Pp_PC_PAPER_TYPE</td>
<td>char *</td>
</tr>
<tr>
<td>Pp_PC_PREVIEW_APP</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_PRINTER_RESOLUTION</td>
<td>PhDim_t *</td>
</tr>
<tr>
<td>Pp_PC_PROP_APP</td>
<td>String</td>
</tr>
<tr>
<td>Pp_PC_REVERSED</td>
<td>char</td>
</tr>
<tr>
<td>Pp_PC_SCALE</td>
<td>PhPoint_t *</td>
</tr>
<tr>
<td>Pp_PC_SOURCE_COLORS</td>
<td>ulong_t *</td>
</tr>
<tr>
<td>Pp_PC_SOURCE_OFFSET</td>
<td>PhPoint_t *</td>
</tr>
<tr>
<td>Pp_PC_SOURCE_RESOLUTION</td>
<td>PhDim_t *</td>
</tr>
<tr>
<td>Pp_PC_SOURCE_SIZE</td>
<td>PhDim_t *</td>
</tr>
<tr>
<td>Pp_PC_USER_ID</td>
<td>String</td>
</tr>
</tbody>
</table>

By default, all members are 0 or NULL.

**Returns:**

- **0** Success.
- **-1** An error occurred. See `errno` for details.
Errors:

EACCES  The specified member couldn’t be changed because it’s locked.
ESRCH   An unknown member was specified.

Examples:

```
set_my_apps_Pp_prefs( PpPrintContext_t *pc )
{
    char do_preview = 1, duplex = some_value;
    PpSetPC( pc, Pp_PC_DO_PREVIEW, &do_preview, 0 );
    PpSetPC( pc, Pp_PC_DUPLEX, &duplex, 0 );
    // etc...
}
```

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Printing in the Photon Programmer’s Guide
**PpStartJob()**

Start a print job

**Synopsis:**

```
int PpStartJob( PpPrintContext_t *pc );
```

**Arguments:**

- `pc` The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

**Library:**

`ph`

**Description:**

This function initializes the current print job. This includes opening the destination device or file and writing out the print-start command and any modified portions of the print context.

The `pc` argument is a pointer to the print context for the print job. This must have been created by `PpCreatePC()`, and may have been configured by calls to `PpSetPC()`, or by the `PtPrintSel` widget.

You normally call this function after setting up the print context with `PpSetPC()` and/or the `PtPrintSelection()` convenience function.

If the `Pp_PC_DEVICE` or `Pp_PC_FILENAME` member of the print context isn’t specified, the printer in `Pp_PC_NAME` is used to determine the print destination. If `Pp_PC_NAME` isn’t specified either, the default printer definition is used. If a print destination still isn’t set in the print context, `PpStartJob()` fails and `errno` is set to `ESRCH`.

Photon draw operations won’t be routed through the print context until you call `PpContinueJob()`.

**Returns:**

- `0` Success.
PpStartJob()

-1 The print context couldn’t be made active, probably because the required working files couldn’t be created. See errno for the specific error.

Errors:

ESRCH No output target is specified in the print context and no printer definition could be found

Examples:

See PpContinueJob().

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:


Printing in the Photon Programmer’s Guide
Synopsis:

```c
void PpSuspendJob( PpPrintContext_t *pc );
```

Arguments:

- `pc`: The pointer to a `PpPrintContext_t` structure that was returned by `PpCreatePC()`.

Library:

- `ph`

Description:

This function deactivates the given print context, if it’s active, making the default draw context active. This means that draw commands are sent to Photon (i.e. printing is turned off).

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:


Printing in the Photon *Programmer’s Guide*
Chapter 13
Pt—Widget Toolkit
The widget toolkit functions let you create or destroy widgets, or manipulate widgets and the relationships between them. These functions gather detailed information about widgets and their environment.

For widget extension functions (which extend the attributes of a widget beyond its resources) and convenience functions (which simplify control over certain widget resources), see the *Widget Reference*. 
PtAddCallback()

Add a single callback entry to a callback list

Synopsis:

```c
void PtAddCallback( PtWidget_t *widget,
                    unsigned long callback_type,
                    PtCallbackF_t *callback,
                    void *data );
```

Arguments:

- `widget` A pointer to the widget that you want to add the callback to.
- `callback_type` The name of the callback list you want to add the function to. For example, Pt_CB_ACTIVATE.
- `callback` A pointer to the function you want to add. The function takes this form:

  ```c
  int (*callback)(PtWidget_t *, void *,
                     PtCallbackInfo_t *)
  ``

- `data` A pointer to data that you want to pass as the second argument to the function.

Library:

- ph

Description:

This function adds a callback to the callback list indicated by `callback_type`. 
Some types of callback resources have special routines that you should use instead of this one:

- **Pt_CB_FILTER**
  
  `PtAddFilterCallback()` or `PtAddFilterCallbacks()`

- **Pt_CB_HOTKEY**
  
  `PtAddHotkeyHandler()`

- **Pt_CB_RAW**
  
  `PtAddEventHandler()` or `PtAddEventHandlers()`

---

**Examples:**

```c
#include <stdlib.h>
#include <Pt.h>

int activated( PtWidget_t *widget, void *data,
               PtCallbackInfo_t *info)
{
    // suppress compiler warnings concerning unused arguments.
    widget = widget, data = data, info = info;

    PtExit( 0 );
    return Pt_CONTINUE;
}

int main()
{
    PtArg_t args;
    PtWidget_t *window, *button;

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT,
                                  0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    PtSetArg( &args, Pt_ARG_TEXT_STRING, "Press Me To Quit", 0 );
    button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT,
                             1, &args );

    // add an activate callback to the button.
```
PtAddCallback()

PtAddCallback( button, Pt_CB_ACTIVATE, activated, NULL );
PtRealizeWidget( window );
PtMainLoop();
//unnecessary
PtRemoveCallback( button, Pt_CB_ACTIVATE, activated, NULL );
return EXIT_SUCCESS;

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:
PtAddCallbacks(), PtAddEventHandler(), PtAddEventHandlers(), PtAddFilterCallback(), PtAddFilterCallbacks(), PtAddHotkeyHandler(), PtRemoveCallback(), PtRemoveCallbacks()

PtCallbackInfo_t in the Photon Widget Reference

Managing Widgets in Application Code chapter of the Photon Programmer’s Guide
**PtAddCallbacks()**

*Add several callback entries to a callback list*

**Synopsis:**
```c
void PtAddCallbacks(
    PtWidget_t *widget,
    unsigned long callback_type,
    PtCallback_t const *callback_defs,
    unsigned int num_callbacks);
```

**Library:**
```
ph
```

**Description:**
This function adds the number of callbacks specified by `num_callbacks` to the callback list specified by `callback_type` (e.g., `Pt_CB_ACTIVATE`).

Some types of callback resources have special routines that you should use instead of this one:

- **Pt_CB_FILTER**
  ```
  PtAddFilterCallback() or PtAddFilterCallbacks()
  ```

- **Pt_CB_HOTKEY**
  ```
  PtAddHotkeyHandler()
  ```

- **Pt_CB_RAW**
  ```
  PtAddEventHandler() or PtAddEventHandlers()
  ```

The `callback_defs` argument contains the address of an array of `PtCallback_t` structures. See the Photon Widget Reference.

**Examples:**
```c
PtWidget_t *widget
PtCallback_t callbacks[]={
    my_first_callback, NULL,
    my_second_callback, "Number 2",
    my_last_callback, NULL
};
```
PtAddCallbacks()

:
PtAddCallbacks( widget, Pt_CB_ACTIVATE, callbacks, 3 );

Classification:

    Photon

    Safety
     Interrupt handler      No
     Signal handler         No
     Thread                 No

See also:

PtAddCallback(), PtAddEventHandler(), PtAddEventHandlers(),
PtAddFilterCallback(), PtAddFilterCallbacks(),
PtAddHotkeyHandler(), PtRemoveCallback(), PtRemoveCallbacks(),

PtCallback_t in the Photon Widget Reference

Managing Widgets in Application Code chapter of the Photon
Programmer’s Guide
PtAddClassStyle()
Add a style to a widget class

Synopsis:

```c
int PtAddClassStyle(
    PtWidgetClassRef_t * const ref,
    PtWidgetClassStyle_t *style);
```

Library:

`ph`

Description:

This function adds the given style to the specified widget class, `ref`. If a style of the same name already exists in the widget class, the contents of the new style replace the old style, and the new style is freed.

To further manipulate the given style, you must get a new pointer to it by calling `PtFindClassStyle()` or `PtGetWidgetStyle()`.

A style is a collection of override methods that can change how a widget looks and behaves. Styles can also add widget resources.

Returns:

The index of the style, or -1 if it couldn’t be added.

Examples:


Classification:

Photon

Safety

Interrupt handler  No

continued...
PtAddClassStyle()

**Safety**

<table>
<thead>
<tr>
<th>Signal handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetStyleMembers(), PtSetWidgetStyle(),

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**PtAddData()**

*Add data to the provided data chain*

**Synopsis:**

```c
int PtAddData( PtDataHdr_t **ptr,
               long type,
               long subtype,
               void *data,
               long len,
               PtDataRemoveF_t *remove );
```

**Arguments:**

- **ptr**
  The address of the pointer to the chain the data should be added to.

- **type**
  A unique data type.

- **subtype**
  A subtype that’s used to distinguish multiple blocks of data added as the same type. This argument shouldn’t be -1, because -1 has special meaning when searching for a specific block within the provided data chain.

- **data**
  A pointer to the data to be added to the provided data chain.

- **len**
  The size of the block of data added, or 0 if it isn’t required.

**Library:**

- **ph**

**Description:**

This function adds a piece of data to the provided data chain. The data provided must be in a block of memory created by `malloc()`. You can retrieve this data by calling `PtFindData()`, or `PtFindNextData()`.
**PtAddData()**

**Returns:**

0 on success, or -1 if an error occurred (e.g. out of memory).

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtFindData(), PtFindNextData(), PtRemoveData(), PtUnlinkData()*
**PtAddEventHandler()**

Add a single Pt_CB_RAW entry to a widget

### Synopsis:

```c
void PtAddEventHandler( PtWidget_t *widget,
                       unsigned long event_mask,
                       PtCallbackF_t *callback,
                       void *data );
```

### Library:

ph

### Description:

This function adds the specified callback to the Pt_CB_RAW callback list that belongs to `widget`. The widget invokes this callback whenever an event type that matches one of the bits in `event_mask` intersects with the widget.

The widget needs to have Pt_GETS_FOCUS set in its Pt_ARG_FLAGS in order to receive key events.

The `callback` argument points to a function that takes this form:

```c
int (*callback)(
    PtWidget_t *widget,
    void *data,
    PtCallbackInfo_t *info)
```

The `data` argument that you pass to `PtAddEventHandler()` is passed as the `data` argument to the callback function.

### Examples:

```c
PtWidget_t *widget;

PtRawCallback_t callbacks[] = {
    Ph_EV_PTR_MOTION_BUTTON, motion_button_callback, NULL,
    Ph_EV_BUT_PRESS | Ph_EV_BUT_RELEASE, start_end_callback,
    "some data"
};
```
PtAddEventHandler()  

//add both event handlers  
PtAddEventHandlers( widget, callbacks, 2 );  
...  
//remove both event handlers  
PtRemoveEventHandlers( widget, callbacks, 2 );  
...  
//add the motion button event handler  
PtAddEventHandler( widget, Ph_EV_PTR_MOTION_BUTTON,  
    motion_button_callback, NULL );  
...  
//remove the motion button event handler  
PtRemoveEventHandler( widget, Ph_EV_PTR_MOTION_BUTTON,  
    motion_button_callbacks, NULL )  
...  
//add both event handlers  
PtAddEventHandlers( widget, callbacks, 2 );  
...  
//remove the motion button event handler  
PtRemoveEventHandler( widget, Ph_EV_PTR_MOTION_BUTTON,  
    motion_button_callbacks, NULL )  
...  
// at this point widget still has the Ph_EV_BUT  
// Press/Release event handler

Classification:

Photon

Safety  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddCallback(), PtAddCallbacks(), PtAddEventHandlers(),  
PtAddFilterCallback(), PtAddFilterCallbacks(),  
PtAddHotkeyHandler(), PtRemoveEventHandler(),  
PtRemoveEventHandlers()
PtAddEventHandler()

PtCallbackInfo_t, PtRawCallback_t, Pt_CB_RAW
(PtWidget) in the Photon Widget Reference

**PtAddEventHandlers()**

Add several Pt_CB_RAW entries to a widget

**Synopsis:**

```c
void PtAddEventHandlers(  
    PtWidget_t *widget,  
    PtRawCallback_t const *callback_defs,  
    unsigned int num_handlers );
```

**Library:**

`ph`

**Description:**

This function adds the number of event handlers specified by `num_handlers` to the `Pt_CB_RAW` callback list that belongs to `widget`. The widget needs to have `Pt_GETS_FOCUS` set in its `Pt_ARG_FLAGS` in order to receive key events. The `callback_defs` argument points to an array of `PtRawCallback_t`, structures. See the Photon Widget Reference.

**Examples:**

See `PtAddEventHandler()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
*PtAddEventHandlers()*

**See also:**

*PtAddCallback(), PtAddCallbacks(), PtAddEventHandler(), PtAddFilterCallback(), PtAddFilterCallbacks(), PtAddHotkeyHandler(), PtRemoveEventHandler(), PtRemoveEventHandlers()*

*PtRawCallback_t, Pt_CB_RAW* in the Photon Widget Reference

PtAddFilterCallback()

Add a single Pt_CB_FILTER callback to a widget

Synopsis:

```c
void PtAddFilterCallback( PtWidget_t *widget,
                         unsigned long event_mask,
                         PtCallbackF_t *callback,
                         void *data );
```

Library:

`ph`

Description:

This function adds the specified callback to the `Pt_CB_FILTER` callback list that belongs to `widget`. The widget invokes this callback whenever an event type that matches one of the bits in `event_mask` intersects with the widget.

The widget needs to have `Pt_GETS_FOCUS` set in its `Pt_ARG_FLAGS` in order to receive key events.

The `callback` argument points to a function that takes this form:

```c
int (*callback)(PtWidget_t *, void *, PtCallbackInfo_t *)
```

Examples:

```c
PtWidget_t *widget;

PtRawCallback_t callbacks[] = {
    Ph_EV_PTR_MOTION_BUTTON, motion_button_callback, NULL,
    Ph_EV_BUT_PRESS | Ph_EV_BUT_RELEASE, start_end_callback, "some data"
};
...
//add both event handlers
PtAddFilterCallbacks( widget, callbacks, 2 );
...
//remove both event handlers
PtRemoveFilterCallbacks( widget, callbacks, 2 );
...
PtAddFilterCallback()

// add the motion button event handler
PtAddFilterCallback( widget, Ph_EV_PTR_MOTION_BUTTON,
                   motion_button_callback, NULL );
...
// remove the motion button event handler
PtRemoveFilterCallback( widget, Ph_EV_PTR_MOTION_BUTTON,
                        motion_button_callbacks, NULL );
...
// add both event handlers
PtAddFilterCallbacks( widget, callbacks, 2 );
...
// remove the motion button event handler
PtRemoveFilterCallback( widget, Ph_EV_PTR_MOTION_BUTTON,
                        motion_button_callbacks, NULL );

// at this point widget still has the Ph_EV_BUT
// Press/Release event handler

Classification:

Photon

Safety

Interrupt handler    No
Signal handler       No
Thread               No

See also:

PtAddCallback(), PtAddCallbacks(), PtAddEventHandler(),
PtAddEventHandlers(), PtAddFilterCallbacks(),
PtAddHotkeyHandler(), PtRemoveFilterCallback(),
PtRemoveFilterCallbacks()

PtRawCallback_t, Pt_CB_FILTER in the Photon Widget Reference

PtCallbackInfo_t in the Photon Widget Reference

“Event handlers” in the Managing Widgets in Application Code
chapter of the Photon Programmer’s Guide
**PtAddFilterCallbacks()**

Add several Pt_CB_FILTER entries to a widget

**Synopsis:**

```c
void PtAddFilterCallbacks(
    PtWidget_t *widget,
    PtRawCallback_t const *callbackdefs,
    unsigned int num_handlers
);
```

**Library:**

```
ph
```

**Description:**

This function adds the number of event handlers specified by
`num_handlers` to the Pt_CB_FILTER callback list that belongs to
`widget`.

> The widget needs to have Pt_GETS_FOCUS set in its
> `Pt_ARG_FLAGS` in order to receive key events.

The `callback_defs` argument points to an array of
`PtRawCallback_t`, structures. See the Photon Widget Reference.

**Examples:**

See `PtAddFilterCallback()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtAddFilterCallbacks()**

See also:

*PtAddCallback(), PtAddCallbacks(), PtAddEventHandler(), PtAddEventHandlers(), PtAddFilterCallback(), PtAddHotkeyHandler(), PtRemoveFilterCallback(), PtRemoveFilterCallbacks()*

*PtRawCallback_t, Pt_CB_FILTER* in the Photon Widget Reference

**PtAddHotkeyHandler()**

*Add a single hotkey handler entry to a widget*

**Synopsis:**

```c
void PtAddHotkeyHandler( PtWidget_t *widget,
                        unsigned key_sym_cap,
                        unsigned key_mods,
                        short flags,
                        void *data,
                        PtCallbackF_t *callback);
```

**Library:**

ph

**Description:**

This function adds the specified callback to the *Pt_CB_HOTKEY* callback list that belongs to the specified widget. The widget will invoke this callback whenever all three of the following conditions are met:

- The widget’s window has focus.
- The widget is selectable (not required if the widget is a window).
- The widget’s window receives a key event that matches `key_sym_cap` and `key_mods` (that is, the key is not consumed by another widget).

The `flags` argument can contain the following:

- **Pt_HOTKEY_SYM**
  
  Interpret `key_sym_cap` as a key sym; the default is to interpret it as a key cap.

- **Pt_HOTKEY_IGNORE_MODS**
  
  Ignore the `key_mods` argument. This flag is typically used in menus, where you want both upper- and lowercase letters to be accepted as hot keys.
PtAddHotkeyHandler()

Pt_HOTKEY_CHAINED

Let the hotkey be bound to more than one widget. The callbacks for all widgets with the hotkey are invoked (instead of just the callback for the last widget to register the hotkey).

The callback argument points to a function that takes this form:

```c
int (*callback) (PtWidget_t *, void *, PtCallbackInfo_t *)
```

Hotkey entries are stacked. As a result, the last hotkey attached is the first to be checked.

If a hotkey callback is triggered, the key event is consumed and no other hotkey callbacks are invoked. If callback is NULL, the widget’s Pt_CB_ACTIVATE callback list is invoked when the hotkey is pressed.

Note the following:

- Key caps, key mods, and key syms are defined in `<photon/PkKeydef.h>`.
- Key mods are prefixed with Pk_KM_.
- Key caps and key syms are prefixed with only Pk_.

Hotkeys are handled at the window level. So if two widgets within the same window happen to register the same hotkey definition without specifying the Pt_HOTKEY_CHAINED flag, only the callback for the last duplicate hotkey is invoked.

Hotkey callbacks are automatically registered when the owner widget is realized, and automatically removed when the owner widget is unrealized.

Examples:

```c
PtWidget_t *widget1, *widget2, *widget3, *window;

:;

// add a hotkey to the window on the key sym for Escape.

PtAddHotkeyHandler( window, Pk_Escape, 0,
```
PtAddHotkeyHandler()

    Pt_HOTKEY_SYM,
    NULL, escape_callback );

    // add a hotkey handler for the digit "1" to
    // widget1 that ignores the states of Ctrl/Alt/Shift.
    PtAddHotkeyHandler( widget1, Pk_1, 0,
                            Pt_HOTKEY_IGNORE_MODS,
                            NULL, one_callback );

    // add a hotkey handler for the digit "2" to widget2
    // that will be triggered only if the CTRL modifier
    // is pressed when "2" is hit.
    PtAddHotkeyHandler( widget2, Pk_2, Pk_KM_CTRL, 0,
                            NULL, ctrl_2_callback );

    // add a hotkey handler for the digit 3 to widget3.
    // When triggered, widget3's activate callback will be
    // invoked with a reason type of Pt_CB_ACTIVATE and a
    // reason subtype of Pt_CB_HOTKEY.
    PtAddHotkeyHandler( widget3, Pk_3, 0, 0, NULL, NULL );

    // Remove the hotkey handlers.
    PtRemoveHotkeyHandler( window, Pk_Escape, 0,
                            Pt_HOTKEY_SYM,
                            NULL, escape_callback );
    PtRemoveHotkeyHandler( widget1, Pk_1, 0,
                            Pt_HOTKEY_IGNORE_MODS,
                            NULL, one_callback );
    PtRemoveHotkeyHandler( widget2, Pk_2, Pk_KM_CTRL, 0,
                            NULL, ctrl_2_callback );
    PtRemoveHotkeyHandler( widget3, Pk_3, 0, 0,
                            NULL, NULL );

Classification:

    Photon

    Safety
    Interrupt handler  No
    Signal handler     No

    continued...
**PtAddHotkeyHandler()**

### Safety

| Thread | No |

### See also:

*PtAddCallback(), PtAddCallbacks(), PtAddEventHandler(),
PtAddEventHandler(), PtAddFilterCallback(),
PtAddFilterCallbacks(), PtRemoveHotkeyHandler()*

*PtCallbackInfo_t* in the Photon Widget Reference

“Hotkey callbacks” in the Editing Resources and Callbacks in PhAB chapter of the Photon Programmer’s Guide.
**PtAddResponseType()**

Add data to the response chain

**Synopsis:**

```c
PhTransportLink_t *
PtAddResponseType(
    PtTransportCtrl_t *ctrl,
    PtRequestables_t *requestable,
    char *type,
    char *desc,
    int unsigned transport,
    char *packing_type,
    void *vdata,
    int unsigned vdata_size,
    int unsigned const flags );
```

**Library:**

ph

**Description:**

`PtAddResponseType()` adds data to the response chain of the transport-control structure pointed to by `ctrl`. This chain is a queue of data that can be or has been requested by the destination of a drag-and-drop operation.

The source of a drag-and-drop operation can call `PtAddResponseType()` when it’s packing the data to be dragged or when a destination actually requests the data. If a destination has already requested the data, the library automatically sends the data to the destination immediately.

This function searches a list of the data that has already been requested and, if the request for this data has been made, provides the data to the requester.

- **ctrl** A pointer to the `PtTransportCtrl_t` structure that controls the drag-and-drop operation.
- **requestable** A pointer to the description of the requestable data.
**PtAddResponseType()**

**type**
A descriptive type name, such as `image`, `text`, `filename`, or `files`. This is simply added to the header for the packed data.

**desc**
The specifics of what’s in the data. The extractor uses a regular-expression match against the description to determine if the data should be unpacked or discarded. This is simply added to the header for the packed data.

**transport**
The available transport types that can be specified when requesting data from the source:

- **Ph_TRANSPORT_INLINE**
  The data being transported is in memory and can be unpacked immediately.

- **Ph_TRANSPORT_FILEREF**
  The data being transported is in the temporary file(s) named in the inlined data.

- **Ph_TRANSPORT_SHMEM**
  The data being transported is in the temporary shared object(s) named in the inlined data.

- **Ph_TRANSPORT_STREAM**
  The data being transported will be inlined a small piece at a time.

- **Ph_TRANSPORT_NAMED_STREAM**
  The data being transported will be inlined a small piece at a time. The streamed data is named so multiple streams of data can be transferred serially.

- **Ph_TRANSPORT_FILE_STREAM**
  The contents of files streamed using extended named streams. This is like the named stream but with extra information with each data block, including file information and so on.
**PtAddResponseType()**

The requester of data must choose one of the available request transport types when requesting delivery of additional data.

- **packing_type** The packing method to be used (Ph_PACK_RAW, Ph_PACK_STRING, or Ph_PACK_STRUCT).
- **vdata** A pointer to the data to be transported.
- **vdata_size** The size of the data pointed to by **vdata**.
- **flags** Flags that affect the operation:
  - Ph_DONT_COPY — refer to the data to be transported instead of physically copying the data to the transport control’s stream buffers.

If this bit is set, any modifications to the data that occur between packing and actual transport of the data will be reflected in the data transported.

**Returns:**

A pointer to the **PhTransportLink_t** structure containing the data just added, or NULL if there isn’t enough memory.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtAddResponseType()

See also:

PhTransportLink_t, PtTransportCtrl_t,
PtTransportRequestable(), PtTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
**PtAlert()**

*Display a message and request a response*

**Synopsis:**

```c
int PtAlert( PtWidget_t *parent,
            PhPoint_t const *location,
            char const *title,
            PhImage_t const *image,
            char const *message,
            char const *msgFont,
            int btnCount,
            char const **buttons,
            char const **btnFonts,
            int defBtn,
            int escBtn,
            int flags );
```

**Arguments:**

- **parent**  
  A pointer to the parent widget of the dialog (usually a window). By setting the `flags`, you can block the parent and/or position the dialog relative to it.

- **location**  
  A pointer to a `PhPoint_t` structure that specifies the location of the dialog relative to the parent or console, depending on the `flags`. If `location` is NULL, the dialog is centered.

- **title**  
  The title for the dialog. If you don’t want a title bar, set this argument to NULL.

- **image**  
  A pointer to a `PhImage_t` structure that specifies an icon to display beside the message. If you don’t want an icon, set this argument to NULL.

- **message**  
  The message to display.

- **msgFont**  
  The font for the message text; the default is `TextFont09`. You should create the font name by calling `PfGenerateFontName()`.

- **btnCount**  
  The number of buttons to display.
**PtAlert()**

**buttons** A pointer to an array of strings to be displayed in the buttons. This array must contain at least `btnCount` strings.

All button-text arguments let you define shortcut keys. Place an ampersand (&) in front of the character that you want to be the shortcut. For example, if you specify &Yes, the Y is underlined in the button, and you can select the button by pressing y or Y.

**btnFonts** A pointer to an array of strings containing the fonts to be used in the buttons. If this argument is NULL, `TextFont09` is used for all the buttons. Otherwise, this array must contain at least `btnCount` fonts. You should create the font names by calling `PfGenerateFontName()`.

**defBtn** The number of the button that initially has focus when the dialog is realized. The left button’s index is 1.

**escBtn** The number of the button that’s bound to the Esc key. If you wish to disable the Esc key, set this argument to 0. If Esc is enabled, the close button is included in the dialog’s titlebar (if there is one). Closing the dialog in this manner is the same as pressing the Esc key; the dialog closes and the `escBtn` button is selected.

**flags** Flags that specify the behavior for the dialog. This can be up to one of the following:

- Pt_CENTER — center the dialog.
- Pt_LEFT — left-align the dialog (the default).
- Pt_RIGHT — right-align the dialog.

with any combination of the following:

- Pt_BLOCK_ALL — block all of the application’s windows while the dialog is displayed.
- Pt_BLOCK_PARENT — block the widget specified by the `parent` argument (if non-NULL).
**PtAlert()**

- **Pt_ESC_DISABLE** — disable the ESC key as a means of dismissing the dialog.
- **Pt_MODAL** — the same as **Pt_BLOCK_ALL**.
- **Pt_RELATIVE** — position the dialog relative to the given parent widget. If this bit isn’t set or `parent` is `NULL`, the dialog is positioned relative to the current console.

**Pt_BLOCK_ALL** overrides **Pt_BLOCK_PARENT**.

**Library:**

```
ph
```

**Description:**

This function displays a dialog that displays a message and contains any number of buttons so that you can respond.

*File Not Saved* can be displayed by **PtAlert()**.

**Returns:**

The number of the button that you selected.
Examples:

```c
char const *btns[] = { "&Save it", "&Discard changes", "&Cancel" };
char Helvetica14[MAX_FONT_TAG];

switch( PtAlert( base_wgt, NULL, "File Not Saved", NULL,
    "The file hasn’t been saved.\n\nWhat do you want to do with it?",
    PfGenerateFontName("Helvetica", 0, 14, Helvetica14),
    3, btns, NULL, 1, 3, Pt_BLOCK_ALL ) ) {

    case 1:
        /* save */
        break;

    case 2:
        /* discard changes */
        break;

    case 3:
        /* cancel */
        return;
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

ApError(), PfGenerateFontName(), PhImage_t, PhPoint_t, PtNotice(), PtPassword(), PtPrompt()

“Dialog modules” in the Working with Modules chapter of the Photon Programmer’s Guide
PtAllowExit()
Allow a Photon application to exit

Synopsis:

```c
void PtAllowExit( void );
```

Library:

```c
ph
```

Description:

`PtAllowExit()` lets Photon know that it’s safe to exit your application. In a multithreaded application, any thread can call `PtExit()`, but another thread might be in the middle of an important operation, such as writing a file. To prevent this situation from arising, call `PtPreventExit()` before starting the operation, and call `PtAllowExit()` when it’s done.

Instead of calling `PtPreventExit()` and `PtAllowExit()` directly, you’re better off calling `PtEnter()` and `PtLeave()` with `Pt_DELAY_EXIT` set in the flags. For a discussion of the difference between these functions and using `Pt_DELAY_EXIT`, see “Exiting a multithreaded program” in the Parallel Operations chapter of the Photon Programmer’s Guide.

Classification:

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>
PtAllowExit()

See also:

PtEnter(), PtExit(), PtLeave(), PtPreventExit()

Parallel Operations chapter of the Photon Programmer’s Guide
**PtAppAddCallback()**

*Add a callback to an application's callback list*

**Synopsis:**

```c
void PtAppAddCallback(
    unsigned long cb_type,
    int (*func)(void*,PtCallbackInfo_t*),
    void *data);
```

**Arguments:**

- **cb_type**  
The name of the callback list you want to add the function to. For example, *Pt_CB_APP_EXIT*.

- **func**  
A pointer to the function you want to add. The function takes this form: `int (*func)(PtCallbackInfo_t *, void *)`.

- **data**  
A pointer to data that you want to pass as the second argument to the function.

**Library:**

*ph*

**Description:**

This function adds a callback to the application’s callback list. For a list of application callbacks, see *PtAppSetResources()*.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer's Guide.

**Returns:**

- 0  
  Success.

- -1  
  An error occurred.
**PtAppAddCallback()**

**Examples:**

```c
int exit_cb(void *data,
            PtCallbackInfo_t *cbinfo)
{
    printf( "I\'m exiting\n" );
    return( Pt_CONTINUE );
}
...
PtAppAddCallback(Pt_CB_APP_EXIT, exit_cb, NULL);
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtAppGetResources()`, `PtAppGetResource()`,
`PtAppRemoveCallback()`, `PtAppSetResource()`, `PtAppSetResources()`,
`PtSetArg()`. 
**PtAppAddEventHandler()**

*Add an event handler to an application’s callback list*

**Synopsis:**

```c
void PtAppAddEventHandler( unsigned long event_mask,
                          int (*func)(void *,PtCallbackInfo_t *),
                          void *data);
```

**Arguments:**

- `event_mask` The event type that causes the application to invoke the event handler `func`.
- `func` A pointer to the event handler function you want to add. The function takes this form: `int (*func)(void *,PtCallbackInfo_t *)`.
- `data` A pointer to data that you want to pass as the second argument to the function.

**Library:**

`ph`

**Description:**

This function adds an event handler to the application’s `Pt_CB_RAW` callback list. The application invokes this handler whenever an event type that matches one of the bits in `event_mask` intersects with the application.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*continued...*
PtAppAddEventHandler()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppGetResources(), PtAppGetResource(), PtAppRemoveCallback(), PtAppRemoveEventHandler(), PtAppSetResource(), PtAppSetResources(), PtSetArg().

**PtAppAddFd(), PtAppAddFdPri()**

Install a file-descriptor function

**Synopsis:**

```c
int PtAppAddFd( PtAppContext_t app,  
    int fd,  
    unsigned mode,  
    PtFdProc_t fun,  
    void *data);

int PtAppAddFdPri( PtAppContext_t app,  
    int fd,  
    unsigned mode,  
    PtFdProc_t fun,  
    void *data,  
    int priority);
```

**Arguments:**

- **app**
  The address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

- **fd**
  The file descriptor to attach an FD handler to.

- **mode**
  Defines what kind of conditions the application is interested in:

  - Pt_FD_READ Data available for reading.
  - Pt_FD_WRITE Buffer space available for writing.
  - Pt_FD_OBAND Out-of-band data available.

  These values correspond to conditions defined for the ionotify() or select() functions. You can OR two or all three values together. You can change the mode later by calling PtAppSetFdMode().

- **fun**
  Defines the FD handler function to be called. This function is of type `PtFdProc_t`:

```c
typedef int PtFdProcF_t( int fd, void *data, unsigned mode );
typedef PtFdProcF_t *PtFdProc_t;
```
**PtAppAddFd(), PtAppAddFdPri()**

The `fd` and `data` arguments have the same value as the `fd` and `data` arguments to `PtAppAddFd()`. The `mode` argument indicates which conditions were actually met.

The `fun` function should return `Pt_CONTINUE` to remain on the list of `fd functions`, or `Pt_END` to be removed automatically from it.

### `data`
A pointer to data that you want to pass as the second argument to the FD handler function.

### `priority`
`PtAppAddFdPri()` only.
Specifies the priority of the Photon pulse that’s created (see `PtAppCreatePulse()`).

**Library:**

ph

**Description:**

These functions install an “FD function” that informs the application about device events.

If your application needs to perform I/O such as reading from or writing to a pipe, you should add an **FD handler**. An FD handler is a function that’s called by the main event loop when a given file descriptor (FD) is ready for input or output.

Multiple FD functions attached to the same file descriptor aren’t supported. `PtAppAddFd()` fails with `errno` set to EBUSY if you try to attach another function to the same FD.

**Returns:**

```
0   Success.
-1  An error occurred.
```
PtAppAddFd(), PtAppAddFdPri()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppCreatePulse(), PtAppRemoveFd(), PtAppSetFdMode(), PtFdProc_t

“Other I/O mechanisms” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtAddFilterCallback()**

Add a filter callback to an application's callback list

**Synopsis:**

```c
void PtAddFilterCallback(
    unsigned long event_mask,
    int (*func)(void *,PtCallbackInfo_t *),
    void *data);
```

**Arguments:**

- `event_mask` The event type that causes the application to invoke the filter callback `func`.
- `func` A pointer to the event handler function you want to add. The function takes this form: `int (*func)(void *,PtCallbackInfo_t *)`.
- `data` A pointer to data that you want to pass as the first argument to the function.

**Library:**

ph

**Description:**

This function adds an event handler to the application’s `Pt_CB_FILTER` callback list. The application invokes this handler whenever an event type that matches one of the bits in `event_mask` intersects with the application.

For more information, see the Manipulating Resources in Application Code chapter of the Photon *Programmer’s Guide*.

**Classification:**

Photon
PtAddFilterCallback()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppGetResources(), PtAppGetResource(),
PtAppRemoveCallback(), PtAppRemoveFilterCallback(),
PtAppSetResource(), PtAppSetResources(), PtSetArg().

**PtAppAddHotkeyHandler()**

*Add a hotkey callback to an application's callback list*

**Synopsis:**

```c
void PtAppAddHotkeyHandler(
    unsigned key_sym_cap,
    unsigned key_mods,
    short flags,
    void *data,
    int (*event_f)(void *,PtCallbackInfo_t *));
```

**Arguments:**

- `key_sym_cap` The key cap or key sym (depending on `flags`) to match.
- `key_mods` The key modifiers to match.
- `flags` Can contain:
  - `Pt_HOTKEY_SYM` Interpret `key_sym_cap` as a key sym; the default is to interpret it as a key cap.
  - `Pt_HOTKEY_IGNORE_MODS` Ignore the `key_mods` argument. This flag is typically used where you want both uppercase and lowercase letters to be accepted as hot keys.
  - `Pt_HOTKEY_CHAINED` The key event can be handled by more than one application callback. By default, the key event is consumed by the first handler that handles it, and no other handlers are invoked.
- `data` A pointer to data you want passed as the first argument to the hotkey handler `event_f`.
- `event_f` A pointer to the hotkey handler you want to add.
PtAppAddHotkeyHandler()

Library:

ph

Description:

This function adds a key event handler to the application’s Pt_CB_HOTKEY callback list. The application invokes the last-added handler on this list whenever a key event type that matches key_sym_cap and key_mods intersects with the application.

Note the following:

- Key caps, key mods, and key syms are defined in <photon/PkKeydef.h>.
- Key mods are prefixed with Pk_KM_.
- Key caps and key syms are prefixed with only Pk_.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppGetResources(), PtAppGetResource(), PtAppRemoveCallback(), PtAppRemoveHotkeyHandler()
PtAppSetResource(), PtAppSetResources(), PtSetArg().

**PtAppAddInput()**

*Add an input processing function*

**Synopsis:**

```c
PtInputId_t *PtAppAddInput(
    PtAppContext_t app_context,
    pid_t pid,
    PtInputCallbackProc_t input_func,
    void *data );
```

**Library:**

ph

**Description:**

This routine adds a function to a `PtMainLoop()` input-event processing chain.

If you want to attach an input handler to a process on a remote node, use `PtAppAddInputRemote()`.

The `app_context` argument is the address of the application context, a structure that manages all the data associated with this application. Specify NULL for `app_context`, so that the default context is used.

The input function is executed whenever the application receives a message from process `pid`. If `pid` is negative, it’s the ID of a Photon pulse.

If you specify a `pid` of 0, the input function is called for every non-Photon event message that’s received, but only if there’s no input function that catches messages from the sending `pid` specifically.

The `rcvid` argument that the input function gets may have a different value:

- In case of real processes under QNX Neutrino, the input function gets the `rcvid`—which is the value that you’ll need for replying, just like the PID under QNX 4. To retrieve the ID of the process, use `PtGetRcvidPid()`.

- In the case of Photon pulses, the `rcvid` argument won’t necessarily have the same value as the pulse PID, either. The `rcvid` matches
the pulse PID on bits defined by \_NOTIFY\_DATA\_MASK (see \_ionotify() in the QNX Neutrino Library Reference), but the other bits will be taken from the Neutrino pulse or signal that was received.

The \texttt{input\_func} argument points to the input function to be invoked. The function takes this form:

\begin{verbatim}
int (*input\_func) (void *data, int rcvid,
                     void *message, size_t size);
\end{verbatim}

The arguments are:

- \texttt{data} A pointer to any extra data you want to pass to the input handler.
- \texttt{rcvid} The rcvid of the process that sent the message.
- \texttt{message} A pointer to the message sent.
- \texttt{size} The size of the message buffer. If the actual message is longer than the buffer, load the rest of the message by calling \textit{MsgRead()}. If you application knows the maximum size of a message that can be possibly sent to it, you can use \textit{PtResizeEventMsg()} to ensure that the buffer is large enough.

Receiving a large Photon event may make the buffer bigger than was established by \textit{PtResizeEventMsg()}.

You can declare the function to be of type \texttt{PtInputCallbackProcF\_t} to take advantage of the compiler’s type-checking.
If the input function changes the display, it should call \textit{PtFlush()} to make sure the display is updated.

The input function must return one of the following:

\begin{itemize}
  \item \texttt{Pt\_CONTINUE}
    The input handler doesn’t recognize the message. If there are other input handlers attached to the same process ID, they’re called. If there are no input handlers attached specifically to this process ID, or if all input handlers attached specifically to this process ID return \texttt{Pt\_CONTINUE}, the library looks for input handlers attached to pid 0. If all the input handlers return \texttt{Pt\_CONTINUE}, the library replies to the message with an \texttt{ENOSYS}.
  \item \texttt{Pt\_END}
    The message has been recognized and processed and the input handler needs to be removed from the list. No other input handlers are called for this message.
  \item \texttt{Pt\_HALT}
    The message has been recognized and processed but the input handler needs to stay on the list. No other input handlers are called for this message.
\end{itemize}

\textit{name\_attach()} and \textit{PtAppAddInput()}

\textit{PtAppAddInput()} and \textit{name\_attach()} both try to create a channel with \texttt{\_NTO\_CHF\_COID\_DISCONNECT} and \texttt{\_NTO\_CHF\_DISCONNECT} set (see the QNX Neutrino \textit{Library Reference}). If your application calls both functions, you need to let Photon use the same channel as \textit{name\_attach()}. To do this, call these functions in this order:

\begin{itemize}
  \item \textit{name\_attach()}
  \item \textit{PhChannelAttach()}
  \item \textit{PtAppAddInput()}
\end{itemize}
**PtAppAddInput()**

See the Examples section of `PhChannelAttach()` for a sample of code that illustrates the correct order. If you want to create a separate channel for Photon, it doesn’t matter whether you create it and give it to `PhChannelAttach()` before or after calling `name_attach()`. But keep in mind that since certain mechanisms in Photon library expect the Photon channel to have the two DISCONNECT flags, they might not work properly if it doesn’t. One such mechanism is the detection of broken connections (see `PtConnectionClientSetError()` and `PtConnectionServerSetError()` and anything that relies on it.

**Returns:**

A pointer to a `PtInputId_t` structure that uniquely identifies the specified input function for the given application context. If an error occurs, the function returns NULL.

**Examples:**

See the example given in “Photon pulses” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*

**Classification:**

*Photon*

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhChannelAttach()`, `PtAppAddInputRemote()`, `PtMainLoop()`, `PtAppCreatePulse()`, `PtAppRemoveInput()`, `PtResizeEventMsg()`
Interprocess Communication chapter of the Photon Programmer’s Guide

`name_attach()` in the QNX Neutrino Library Reference
**PtAppAddInputRemote()**

Add an input processing function for a remote node

**Synopsis:**

```c
int PtAppAddInputRemote( PtAppContext_t app_context,
                        int nd,
                        pid_t pid,
                        PtInputCallbackProc_t input_func,
                        void *data );
```

**Arguments:**

- **app_context**
  The address of the application context, a structure that manages all the data associated with this application. Specify NULL for `app_context`, so that the default context is used.

- **nd**
  The node descriptor for the remote node running the process you want to attach an input processing function to. For more information see “Node descriptors” in `ConnectAttach()` in the Neutrino Library Reference.

- **pid**
  The process ID you want to attach the input handler to. The input function is executed whenever the application receives a message from process `pid` on node `nd`. If `pid` is negative, it’s the ID of a Photon pulse, and `nd` must be 0.

  If you specify a `pid` of 0, the input function is called for every non-Photon event message that’s received, but only if there’s no input function that catches messages from the sending `pid` specifically. If `pid` is 0, you must specify a `nd` of 0 as well. Your input handler will get all messages from processes on any node, and all pulses (except those that were already handled and returned Pt_END or Pt_HALT).

- **input_func**
  A pointer to the input function to be invoked (see below).

- **data**
  A pointer to any extra data you want to pass to the input handler.
PtAppAddInputRemote()

Library:

ph

Description:

This routine adds a function to a PtMainLoop() input-event processing chain for a process running on a remote node.

To attach an input handling function to a process on your node, use PtAppAddInput().

The input_func argument points to the input function to be invoked. The function takes this form:

```
int (*input_func)(void *data, int rcvid, void *message, size_t size);
```

The arguments are:

- **data**: The pointer to the data argument passed to PtAppAddInputRemote().
- **rcvid**: For a message from another process, rcvid is the receive ID. Pass it to MsgReceive() to send a reply (see MsgReceive() in the QNX Neutrino Library Reference). To retrieve the pid of the process and the node descriptor of the machine it’s running on, use PtGetRcvidPid(). For a Photon pulse, it’s the value from the pulse structure (see _pulse in the QNX Neutrino Library Reference).
- **message**: A pointer to the message sent.
- **size**: The size of the message buffer. If the actual message is longer than the buffer, load the rest of the message by calling MsgRead(). If you application knows the maximum size of a message that can be possibly sent to it, you can use PtResizeEventMsg() to ensure that the buffer is large enough.
PtAppAddInputRemote()

Receiving a large Photon event may make the buffer bigger than was established by PtResizeEventMsg().

You can declare the function to be of type PtInputCallbackProcF_t to take advantage of the compiler’s type-checking.

If the input function changes the display, it should call PtFlush() to make sure the display is updated.

The input function must return one of the following:

Pt_CONTINUE

The input handler doesn’t recognize the message. If there are other input handlers attached to the same process ID, they’re called. If there are no input handlers attached specifically to this process ID, or if all input handlers attached specifically to this process ID return Pt_CONTINUE, the library looks for input handlers attached to pid 0. If all the input handlers return Pt_CONTINUE, the library replies to the message with an ENOSYS.

Pt_END

The message has been recognized and processed and the input handler needs to be removed from the list. No other input handlers are called for this message.

Pt_HALT

The message has been recognized and processed but the input handler needs to stay on the list. No other input handlers are called for this message.

name_attach() and PtAppAddInputRemote()

PtAppAddInputRemote() and name_attach() both try to create a channel with _NTO_CHF_COID_DISCONNECT and _NTO_CHF_DISCONNECT set (see the QNX Neutrino Library)
PtAppAddInputRemote()

Reference). If your application calls both functions, you need to let Photon use the same channel as _name_attach(). To do this, call these functions in this order:

- _name_attach()
- PhChannelAttach()
- PtAppAddInputRemote()

If you want to create a separate channel for Photon, it doesn’t matter whether you create it and give it to PhChannelAttach() before or after calling _name_attach(). But keep in mind that since certain mechanisms in Photon library expect the Photon channel to have the two DISCONNECT flags, they might not work properly if it doesn’t. One such mechanism is the detection of broken connections (see PtConnectionClientSetError() and PtConnectionServerSetError()) and anything that relies on it.

Returns:

A pointer to a _PtInputId_t structure that uniquely identifies the specified input function for the given application context. If an error occurs, the function returns NULL.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtAppAddInputRemote()

See also:
PhChannelAttach(), PtAppAddInput(), PtAppCreatePulse(), PtAppRemoveInput(), PtGetRcvIdPidNd(), PtMainLoop(), PtResizeEventMsg()

Interprocess Communication chapter of the Photon Programmer’s Guide
pulse, MsgReceive(), and name_attach() in the QNX Neutrino Library Reference
Synopsis:

```c
int PtAppAddSignalProc( PtAppContext_t app,
                        sigset_t const *set,
                        PtSignalProc_t func,
                        void *data);
```

Library:

```c
ph
```

Description:

This function adds Photon signal handling to the context `app`.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

All signals in the set `set` are trapped and directed to a function that synchronizes with the Photon widget library and, at the next safe point, invokes the user-supplied function `func()`.

The user callback function is declared as follows:

```c
int func(int signal, void *data)
```

You can declare the function to be of type `PtSignalProcF_t` to take advantage of the compiler’s type-checking.

It’s invoked with the signal number and user data as parameters. It should return Pt_CONTINUE to remain installed, or Pt_END to have the callback removed for this signal.

You can add more than one function for a set of signals or set of intersecting signals. All handlers for a signal are called, but the order they’re called in is unspecified.

The Photon widget library isn’t signal-safe—normal signal handling functions must not call Photon library functions or alter Photon globals. Because this mechanism synchronizes with the widget library before calling the user function, no such limitations are placed on processing within handler functions installed via this routine.
PtAppAddSignalProc()

Returns:

0  Success.
-1  An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtAppRemoveSignal()

Interprocess Communication in the Photon Programmer’s Guide
**Synopsis:**

```c
PtWorkProcId_t *PtAppAddWorkProc(
    PtAppContext_t   app_context,
    PtWorkProc_t    work_func,
    void *data );
```

**Library:**

ph

**Description:**

This function adds a WorkProc entry to the WorkProc (background) process stack. The entry becomes the current WorkProc entry.

WorkProc functions don’t run concurrently; only the one at the top of the stack runs.

There is one exception to this rule. If the work procedure that’s at the top of the stack is running already, the next one is called. This is only possible if the already running procedure allows the Photon library to start another one, perhaps by calling a modal function like `PtModalBlock()`, `PtFileSelection()` or `PtAlert()`, or calling `PtLeave()` while you have other threads ready to process events.

When there are no events pending from Photon, the current WorkProc entry’s function is invoked by `PtMainLoop()`.

The `app_context` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

The `work_func` argument points to the WorkProc function to be invoked when no Photon events are pending. The function takes this form:

```c
int (*work_func)(void *data)
```

You can declare the function to be of type `PtWorkProcF_t` to take advantage of the compiler’s type-checking.
PtAppAddWorkProc()

- If the WorkProc function changes the display, it should call 
  PtFlush() to make sure the display is updated.

- If the WorkProc function pointed to by work_func returns 
  Pt_CONTINE, it stays on the WorkProc stack. If it returns 
  Pt_END, it’s removed from the WorkProc stack and the next 
  WorkProc entry on the stack becomes the current entry.

Returns:

A pointer to a PtWorkProcId_t structure that identifies the specified 
WorkProc entry for the given application context. If an error occurs, 
the function returns NULL.

Examples:

This example comes from the Rebound demo. These are the callbacks 
that start and stop the rebounding work procedure, 
rebound_process().

// From src/start_rebound.c
int
start_rebound( PtWidget_t *widget, void *data, 
               PtCallbackInfo_t *cbinfo )
{
    PtArg_t args[2];

    if(stopped) {
        if ( delay_value == 0 ) {
            if ( !bkgd_id ) // is one running?
                bkgd_id = PtAppAddWorkProc( NULL, rebound_process, 
                                            ABW_rb_pane );
            PtSetArg( &args[0], Pt_ARG_TIMER_INITIAL, 0, 0 );
            PtSetResources( ABW_timer_wgt, 1, args );
        } else {
            if ( bkgd_id )
                PtAppRemoveWorkProc( NULL, bkgd_id );
            PtSetArg( &args[0], Pt_ARG_TIMER_INITIAL, 1, 0 );
            PtSetArg( &args[1], Pt_ARG_TIMER_REPEAT, 
                      SPEED_MULTIPLY*delay_value, 0 );
            PtSetResources( ABW_timer_wgt, 2, args );
        }
        stopped = 0;
    }

1210   Chapter 13 • Pt—Widget Toolkit
PtAppAddWorkProc()

}  
  return( Pt_CONTINUE );
}

// From src/stop_rebound.c
int
stop_rebound( PtWidget_t *widget, void *data,
              PtCallbackInfo_t *cbinfo )
{
  PtArg_t args[1];

  if ( bkgd_id ) {
    PtAppRemoveWorkProc( NULL, bkgd_id );
    bkgd_id = NULL;
  }
  PtSetArg( &args[0], Pt_ARG_TIMER_INITIAL, 0, 0 );
  PtSetResources( ABW_timer_wgt, 1, args );
  stopped = 1;
  return( Pt_CONTINUE );
}

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtMainLoop(), PtAppRemoveWorkProc(), PtSetParentWidget(), PtWorkProcF_t

Parallel Operations in the Photon Programmer’s Guide
**PtAppCreatePulse()**

Create a Photon pulse

**Synopsis:**

```c
pid_t PtAppCreatePulse( PtAppContext_t app,
                        int priority );
```

**Arguments:**

- `app` The address of the application context, a structure that manages all the data associated with this application. Specify this as NULL, so as to use the default context.
- `priority` The priority of the pulse. If `priority` is -1, the pulse’s priority is the same as that of the calling process.

**Library:**

ph

**Description:**

This function creates a Photon pulse. Under QNX Neutrino, `PtAppCreatePulse()` creates a Neutrino pulse.

**Returns:**

A pulse PID (a negative value that’s guaranteed never to be -1), or 0 if an error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1212  Chapter 13  ●  Pt—Widget Toolkit
See also:

PtAppDeletePulse(), PtAppPulseTrigger(), PtChannelCreate(), PtPulseArm()

MsgDeliverEvent() in the QNX Neutrino Library Reference

Interprocess Communication in the Photon Programmer’s Guide
**PtAppDeletePulse()**

*Delete a Photon pulse*

**Synopsis:**

```c
int PtAppDeletePulse( PtApplicationContext_t app,
                      pid_t pulse_pid );
```

**Library:**

*ph*

**Description:**

This function deletes the Photon pulse identified by `pulse_pid` (and the proxy in QNX 4). The `pulse_pid` identifies a pulse created by `PtAppCreatePulse()`.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

If the application creates and destroys pulses frequently, the pulse ID will be reused eventually. At least several hundred pulses can be safely created and destroyed before that happens. If a pulse ID is reused but notifications are still generated with the old pulse, they may or may not be delivered to the new pulse handler.

**Returns:**

- 0 Success
- -1 An error occurred

**Classification:**

Photon

**Safety**

Interrupt handler  No

*continued...*
PtAppDeletePulse()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppCreatePulse(), PtAppPulseTrigger(), PtChannelCreate(), PtPulseArm()

MsgDeliverEvent() in the QNX Neutrino Library Reference

Interprocess Communication in the Photon Programmer’s Guide
PtAppGetResource()

Retrieve one resource value for an application

Synopsis:

```c
#define PtAppGetResources(type, value, len)
```

Arguments:

- **type**: The resource manifest (E.G, `Pt_CB_APP_EXIT`).
- **value**: The address of a pointer to the appropriate data type (see the “New resources” tables in the Photon Widget Reference).
- **len**: Depends on the resource type.

Library:

`ph`

Description:

This macro sets a pointer to a resource value within the application. `PtAppGetResource()` doesn’t support the nonpointer method of getting resources. For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide. For a list of application callbacks, see `PtAppSetResources()`.
WARNING: Because PtAppGetResource() returns pointers directly into the internals of the widget, don’t modify those values directly. If you wish to retrieve the value of a given resource and then modify that value:

1. Get the resource.
2. Copy the resource to a temporary variable.
3. Modify the temporary variable.
4. Using the modified copy, set the resource.

Returns:

0   Success.
-1  An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtAppGetResource()

See also:

PtAppAddCallback(), PtAppGetResources(),
PtAppRemoveCallback(), PtAppSetResources(), PtAppSetResource(),
PtSetArg().
**Synopsis:**

```c
int PtAppGetResources(int n_args,
                       PtArg_t *args)
```

**Arguments:**

- `n_args` The number of items in the `args` array.
- `args` Indicates which resources are retrieved.

**Library:**

`ph`

**Description:**

This function sets pointers to resource values within the application. You must initialize the `args` array with `PtSetArg()` or `Pt_Arg()` before calling `PtAppGetResources()`. The Pt type of a resource determines how that resource should be set or queried. You use the Pt type when setting a resource entry with `PtSetArg()`. For a list of application callbacks, see `PtAppSetResources()`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon *Programmer’s Guide.*


**PtAppGetResources()**

---

WARNING: Because *PtAppGetResources()* returns pointers directly into the internals of the widget, don’t modify those values directly. If you wish to retrieve the value of a given resource and then modify that value:

1. Get the resource.
2. Copy the resource to a temporary variable.
3. Modify the temporary variable.
4. Using the modified copy, set the resource.

---

**Returns:**

- 0  Success.
- -1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtAppAddCallback(*), *PtAppGetResource(*), *PtAppRemoveCallback(*), *PtAppSetResources(*), *PtAppSetResource(*), *PtSetArg(*).
**PtAppInit()**

Initialize an application and create the main window

### Synopsis:

```c
PtWidget_t *PtAppInit( PtAppContext_t *app_context,
                      int *argc,
                      char **argv,
                      int num_args,
                      PtArg_t const *args );
```

### Library:

`ph`

### Description:

This function:

- Initializes the connection to the Photon Manager.
- Initializes the widget library.
- Creates a default application context (`app_context`) if one doesn’t exist.
- Creates a main window using the `num_args` and `args` arguments—these are the same arguments passed to `PtCreateWidget()`.

### Returns:

A pointer to the main window, or NULL if an error occurs.

### Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtApplInit()

See also:

PhAttach(), PtArg_t, PtCreateWidget(), PtInit(), PtMainLoop()

“Basic steps” in the Programming Photon without PhAB chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PtAppPulseTrigger( PtAppContext_t app,
                        pid_t pulse );
```

**Library:**

`ph`

**Description:**

This function allows an application to deliver a Photon pulse to itself:

- In QNX Neutrino, this function uses (`pulse | _NOTIFY_COND_MASK`) to identify the pulse (and its handler), but the value of the `pulse` argument is delivered.

- In QNX 4, the value of `pulse` must match the pulse PID exactly.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as `NULL`, so that the default context is used. The `pulse` argument is a pulse ID returned by `PtAppCreatePulse()`.

**Returns:**

- `0` Success.
- `-1` An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtAppPulseTrigger()

See also:

PtAppCreatePulse(), PtAppDeletePulse(), PtChannelCreate(), PtPulseArm()

MsgDeliverEvent() in the QNX Neutrino Library Reference

Interprocess Communication in the Photon Programmer’s Guide
**PtAppRemoveCallback()**

Remove a callback from an application's callback list

**Synopsis:**

```c
void PtAppRemoveCallback(unsigned long cb_type,
                         int (*func)(void*,PtCallbackInfo_t*),
                         void *data)
```

**Arguments:**

- `cb_type` The type of the callback you want to remove. For example, `Pt_CB_APP_EXIT`.
- `func` A pointer to the callback function you want to remove.
- `data` A pointer to the data associated with the callback.

**Library:**

`ph`

**Description:**

This function removes the first callback entry that matches `func` and `data`. It removes the entry from the `cb_type` callback list that belongs to the application. For a list of application callbacks, see `PtAppSetResources()`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtAppRemoveCallback()

See also:

PtAppAddCallback(), PtAppGetResources(), PtAppGetResource(),
PtAppRemoveHotkeyHandler(), PtAppRemoveFilterCallback(),
PtAppRemoveEventHandler(), PtAppSetResource(),
PtAppSetResources(), PtSetArg().
PtAppRemoveEventHandler()

Remove an event handler from an application’s event callback list

Synopsis:

```c
void PtAppRemoveEventHandler(
    unsigned long event_mask,
    int (*func)(void *, PtCallbackInfo_t *),
    void *data);
```

Arguments:

- `event_mask`: The event type for the handler you want to remove.
- `func`: A pointer to the callback you want to remove.
- `data`: A pointer to data for the callback you want to remove.

Library:

`ph`

Description:

This function removes the first event handler that matches `event_mask`, `func`, and `data` from the application’s `Pt_CB_RAW` callback list.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 13 • Pt—Widget Toolkit  1227
PtAppRemoveEventHandler()

See also:

PtAppAddEventHandler(), PtAppGetResources(),
PtAppGetResource(), PtAppRemoveCallback(),
PtAppRemoveFilterCallback(), PtAppRemoveHotkeyHandler(),
PtAppSetResource(), PtAppSetResources(), PtSetArg().

Synopsis:

```c
int PtAppRemoveFd( PtAppContext_t app, int fd );
```

Library:

`ph`

Description:

This function removes an FD function, `fd`, from the list of input handlers for the application. You'll use an FD function when using pipes in a Photon application.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

Returns:

- 0 Success.
- -1 An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtAppAddFd()`, `PtAppAddFdPri()`, `PtAppSetFdMode()`
PtAppRemoveFd()

“Other I/O mechanisms” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtAppRemoveFilterCallback()**

*Remove a callback from an application’s filter callback list*

### Synopsis:

```c
void PtAppRemoveFilterCallback(
    unsigned long event_mask,
    int (*func)(void *, PtCallbackInfo_t *),
    void *data);
```

### Arguments:

- **event_mask**
  The event mask for the callback you want to remove.

- **func**
  A pointer to the event handler function you want to remove.

- **data**
  A pointer to data for the callback you want to remove.

### Library:

`ph`

### Description:

This function removes the first event handler that matches `event_mask`, `func`, and `data` from the application’s `Pt_CB_FILTER` callback list.

For more information, see the Manipulating Resources in Application Code chapter of the Photon *Programmer’s Guide*.

### Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtAppRemoveFilterCallback() 

See also: 

PtAppAddFilterCallback(), PtAppGetResources(), PtAppGetResource(), PtAppRemoveCallback(), PtAppRemoveEventHandler(), PtAppRemoveHotkeyHandler(), PtAppSetResource(), PtAppSetResources(), PtSetArg().

**PtAppRemoveHotkeyHandler()**

Remove a hotkey handler from an application's hotkey callback list

**Synopsis:**

```c
void PtAppRemoveHotkeyHandler(
    unsigned key_sym_cap,
    unsigned key_mods,
    short flags,
    void *data,
    int (*event_f)(void *, PtCallbackInfo_t *));
```

**Arguments:**

- `key_sym_cap` The `key_sym_cap` for the handler you want to remove.
- `key_mods` The `key_mods` for the handler you want to remove.
- `flags` The `flags` for the handler you want to remove.
- `data` A pointer to data for the callback you want to remove.
- `event_f` A pointer to the callback you want to remove.

**Library:**

`ph`

**Description:**

This function removes the first hotkey handler that matches `key_sym_cap`, `key_mods`, `flags`, `data`, and `event_f` from the application's `Pt_CB_HOTKEY` callback list.

Note the following:

- Key caps, key mods, and key syms are defined in `<photon/PkKeydef.h>`.
- Key mods are prefixed with `Pk_KM_`.
- Key caps and key syms are prefixed with only `Pk_`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon *Programmer's Guide.*
PtAppRemoveHotkeyHandler()

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppAddHotkeyHandler(), PtAppGetResources(),
PtAppGetResource(), PtAppRemoveCallback(),
PtAppRemoveEventHandler(), PtAppRemoveFilterCallback(),
PtAppSetResource(), PtAppSetResources(), PtSetArg().

Synopsis:

```c
void PtAppRemoveInput( PtAppContext_t app_context,
                       PtInputId_t *input_id );
```

Library:

ph

Description:

This routine removes an input entry from the input-event processing chain.

The `app_context` argument indicates which application context the input entry will be removed from. If you specify NULL, the function tries to remove the entry from the default context.

The `input_id` argument points to a `PtInputId_t` structure that describes the input entry to be removed. (This structure was returned by a previous call to `PtAppAddInput`.)

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtAppAddInput()`, `PtMainLoop()`

“Receiving QNX messages” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtAppRemoveSignal()**

*Remove specific signal handling from a context*

**Synopsis:**

```c
int PtAppRemoveSignal( PtAppContext_t app,
                      sigset_t const *set,
                      PtSignalProc_t proc,
                      void *data );
```

**Library:**

ph

**Description:**

This function removes Photon signal handling from the context `app`:

- If `set` is NULL, this function removes all items that match `proc` and `data`.

- If `set` isn’t NULL, this function removes one instance of each specified signal from a list item that matches `proc` and `data`. In other words, if you attach it twice, you have to remove it twice.

Currently only the default application context (`app == NULL`) is supported.

**Returns:**

- `0` Success.
- `-1` Nothing has been removed.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler No</td>
</tr>
<tr>
<td>Signal handler No</td>
</tr>
</tbody>
</table>

continued...
PtAppRemoveSignal()

Safety
Thread No

See also:

PtAppAddSignalProc(), PtSignalProc_t

Interprocess Communication in the Photon Programmer’s Guide
**PtAppRemoveWorkProc()**

*Remove a WorkProc processing function*

**Synopsis:**

```c
void PtAppRemoveWorkProc(
    PtAppContext_t app_context,
    PtWorkProcId_t *WorkProc_id );
```

**Library:**

`ph`

**Description:**

This routine removes a WorkProc function from the WorkProc event-processing stack.

The `app_context` argument indicates which application context the WorkProc function will be removed from. Currently, only the default context (`app_context == NULL`) is supported.

The `WorkProc_id` argument points to a `PtWorkProcId_t` structure that describes the WorkProc entry to be removed. (This structure was returned by a previous call to `PtAppAddWorkProc()`.)

**Examples:**

See `PtAppAddWorkProc()`.

**Classification:**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 13 • Pt—Widget Toolkit
See also:

PtAppAddWorkProc(), PtMainLoop()

Parallel Operations in the Photon Programmer’s Guide
**PtAppSetFdMode()**

*Change the mode that’s of interest to an FD handler*

**Synopsis:**

```c
int PtAppSetFdMode( PtAppContext_t app,
                    int fd,
                    unsigned mode );
```

**Library:**

`ph`

**Description:**

This function changes the mode that’s of interest to the handler for the given file descriptor, `fd`. You’ll use an FD function when using pipes in a Photon application.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as `NULL`, so that the default context is used.

The `mode` argument defines what kind of conditions the application is interested in:

- `Pt_FD_READ` Data available for reading.
- `Pt_FD_WRITE` Buffer space available for writing.
- `Pt_FD_OBAND` Out-of-band data available.

These values correspond to conditions defined for the `ionotify()` or `select()` functions. You can OR the values together.

**Returns:**

- `0` Success.
- `-1` An error occurred; `errno` is set.
**PtAppSetFdMode()**

**Errors:**

- **EINVAL**  The *fd* or *mode* argument is invalid.
- **ESRCH**  There’s no handler registered for the file descriptor.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtAppAddFd(), PtAppAddFdPri(), PtAppRemoveFd()*

“Other I/O mechanisms” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
PtAppSetResource()

Set a single application callback resource

Synopsis:

```
#define PtAppSetResource( type, value, len ) ...
```

Arguments:

- **type**: The resource manifest.
- **value**: The value of the argument being passed.
- **len**: Depends on the type of resource.

Library:

```
ph
```

Description:

This function sets a resource for the application. The `type` argument contains the resource manifest and `value` contains the value of the argument being passed. The way the `len` argument is used depends on the resource type. For a list of application callbacks, see `PtAppSetResources()`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer's Guide.

Returns:

- **0**: Success.
- **-1**: An error occurred.

Examples:

```
int exit_cb(void *data,
            PtCallbackInfo_t *cbinfo)
{
    printf( "%\'m exiting\n" );
    return( Pt_CONTINUE );
};
...
PtAppCallback_t exit_callback = {exit_cb, NULL};
PtAppSetResource(Pt_CB_APP_EXIT, &exit_callback, 0);
```
**PtAppSetResource()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtAppAddCallback()`, `PtAppGetResources()`, `PtAppGetResource()`, `PtAppRemoveCallback()`, `PtAppSetResources()`, `PtSetArg()`.
**PtAppSetResources()**

Set application-level resources

**Synopsis:**

```c
int PtAppSetResources(int n_args,
                      PtArg_t const *args);
```

**Arguments:**

- `n_args`: The number of items in the `args` array.
- `args`: An array of PtArg_t

**Library:**

ph

**Description:**

This function sets the resources specified in the `args` array for the application. This function operates identically to its widget counterpart, `PtSetResources()`, except it doesn’t have a widget argument.

If you’re setting only one resource, it’s easier to call `PtAppSetResource()`.

If you are setting a single callback resource, it is easier to use `PtAppAddCallback()`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

The following resources can be set for an application:

<table>
<thead>
<tr>
<th>Resource</th>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Pt_CB_APP_EXIT</code></td>
<td>PtAppCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
<tr>
<td><code>Pt_CB_APP_WCLASS_CREATED</code></td>
<td>PtAppCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

continued...
### PtAppSetResources()

<table>
<thead>
<tr>
<th>Resource</th>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_CB_FILTER</td>
<td>PtAppRawCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
<tr>
<td>Pt_CB_HOTKEY</td>
<td>PtAppHotkeyCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
<tr>
<td>Pt_CB_RAW</td>
<td>PtAppRawCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Pt_CB_APP_EXIT

<table>
<thead>
<tr>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtAppCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

A list of PtAppCallback_t structures that define the callbacks invoked when the application is exiting via a call to PtExit(). All exit points from the library call PtExit(), but it is up to the application designer to exit via this method also. Callbacks should unconditionally return Pt_CONTINUE — other return values may be interpreted differently in future versions.

Although an application might call PtExit() more than once (for example, from different threads in a multi-threaded application), the callbacks in this list are invoked only once.

Each callback is passed a PtCallbackInfo_t structure that contains at least the following members:

- reason      Pt_CB_APP_EXIT
- reason_subtype
  - 0 (not used).
- event       NULL
- cbdata      NULL
**PtAppSetResources()**

### Pt_CB_APP_WCLASS_CREATED

<table>
<thead>
<tr>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtAppCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

A list of `PtAppCallback_t` structures that define the callbacks invoked immediately after a new widget class is created. Each callback is passed a `PtCallbackInfo_t` structure that contains at least the following members:

- `reason` `Pt_CB_APP_WCLASS_CREATED`
- `reason_subtype` 0 (not used).
- `event` NULL
- `cbdata` A pointer to a `PtAppWClassCallback_t` structure that contains at least `PtWidgetClass_t *wclass`, which indicates the class that has been created.

### Pt_CB_FILTER

<table>
<thead>
<tr>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtAppRawCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

A list of `PtAppRawCallback_t` structures that define the callbacks invoked when an event that matches the provided event mask is to be passed to the application. Application-level event filters are invoked prior to any widget processing (including the widget event filters), and can preempt events in a similar fashion.
Your application must have at least one region open for this resource to work.

Each callback is passed a PtCallbackInfo_t structure. See the Pt_CB_FILTER resource for PtWidget for a description of what this structure contains.

**Pt_CB_RAW**

<table>
<thead>
<tr>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtAppRawCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

A list of PtAppRawCallback_t structures that define the callbacks that the application invokes if the event it receives matches the event mask provided in the PtAppRawCallback_t structure. Application-level raw callbacks are invoked after all widget processing has been done, if the event has not been consumed.

Your application must have at least one region open for this resource to work.

Each callback is passed a PtCallbackInfo_t structure. See the Pt_CB_RAW resource for PtWidget for a description of what this structure contains.

**Pt_CB_HOTKEY**

<table>
<thead>
<tr>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtAppHotkeyCallback_t*</td>
<td>Link</td>
<td>NULL</td>
</tr>
</tbody>
</table>

A list of PtAppHotkeyCallback_t structures. If the application receives a key event that matches a structure’s key cap and key modifiers, the application calls the function specified in that structure.
PtAppSetResources()

Application-level hotkey handlers are invoked after widget processing is completed, and after any application-level Pt_CB_FILTER callbacks.

Your application must have at least one window for this resource to work. Also, you cannot bind application-level hotkey handlers to widgets. Therefore you must supply a callback function (this is different from the widget-level hotkey handlers).

Each callback is passed a PtCallbackInfo_t structure. See the Pt_CB_HOTKEY resource for PtWidget for a description of what this structure contains.

PtAppCallback_t

An application callback structure.

typedef struct Pt_app_callback {
    int (*event_f)( void *, PtCallbackInfo_t * ),
    void *data;
} PtAppCallback_t;

The PtAppCallback_t structure lets you specify an application’s callbacks. This structure contains at least:

event_f A pointer to the callback function.

data A pointer to data you want to pass as the second parameter to the callback function when it’s invoked.

The callback function takes the following arguments:

void *client_data
    The data from the PtAppCallback_t structure.

PtCallbackInfo_t *cbinfo
    A pointer to a common Photon callback structure. The structure provides information related to the widget callback being
invoked, the Photon event, and some widget-specific callback data. The format of the data varies with the widget class and callback type. For more information, see PtCallbackInfo_t.

Callback functions should return Pt_CONTINUE unless the description of the widget’s callback resource tells you to return something else.

PtAppRawCallback_t

The PtAppRawCallback_t structure lets you specify event handlers (raw and filter callbacks) for your application. You use this structure when setting the Pt_CB_RAW or Pt_CB_FILTER resource of your application.

typedef struct Pt_app_raw_callback {
    unsigned long event_mask;
    int (*event_f)( void *, PtCallbackInfo_t * );
    void *data;
} PtAppRawCallback_t;

The structure contains at least the following members:

- **event_mask**: A bitmap that specifies which events trigger the function specified in *event_f*. See PhEvent_t in the Photon Library Reference.
- **event_f**: A pointer to the callback function.
- **data**: A pointer to data that you want to be passed as the second argument to the callback function.

PtAppHotkeyCallback_t

An application hotkey callback structure.

typedef struct Pt_app_hotkey_callback {
    unsigned short key_sym;
    short flags;
    unsigned long key_mods;
    void *data;
    int (*event_f)( void *, PtCallbackInfo_t * );
} PtAppHotkeyCallback_t;

The PtAppHotkeyCallback_t structure lets you specify hotkeys or hotkey handlers, or both, for your application. It contains at least the following members:
**PtAppSetResources()**

*key_sym_cap* Depending on the specified flags, this member contains either the symbol or cap of the key to be interpreted as a hotkey. For valid *key_sym_cap* values, see `<photon/PkKeyDef.h>`.

*flags* Determines how *key_sym_cap* is interpreted and whether or not *keyMods* is used. Valid bits include:

- **Pt_HOTKEY_SYM**
  Interpret *key_sym_cap* as a key symbol; the default is to interpret it as a key cap.

- **Pt_HOTKEY_IGNORE_MODS**
  Ignore the *keyMods* argument. This flag is typically used in menus, where you want both upper- and lowercase letters to be accepted as hotkeys.

*keyMods* Key modifiers that must be active for the key to be considered a hotkey. If the Pt_HOTKEY_IGNORE_MODS flag is set, this member is ignored.

For valid key modifiers, see `<photon/PkKeyDef.h>`. All key-modifier manifests begin with `Pk_KM_`.

*data* A pointer to any data that you want to pass as the second argument to the callback function.

*event_f* A pointer to the hotkey function.

**Returns:**

- **0** At least one of the resources was applied to the application.
- **-1** The application wasn’t modified because it doesn’t contain the given resources or the values of the resources were the same as those already stored in the application.
Examples:

Set the \textit{Pt\_CB\_APP\_EXIT} callback resource for an application:

\begin{verbatim}
int exit_cb(void *data, 
PtCallbackInfo_t *cbinfo)
{
    printf( "I\'m exiting\n" );
    return( Pt_CONTINUE );
}
...
PtAppCallback_t exit_callbacks[] = {{exit_cb, NULL}};
PtArg_t args[1];
PtSetArg( &args[0], Pt_CB_APP_EXIT, exit_callbacks,
        sizeof(exit_callbacks)/sizeof(exit_callbacks[0]));
PtAppSetResources( 1, args );
\end{verbatim}

Classification:

Photon

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\hline
\end{tabular}
\end{center}

See also:

\begin{itemize}
\item PtAppAddCallback(), PtAppGetResources(), PtAppGetResource(),
\item PtAppRemoveCallback(), PtAppSetResource(), PtSetArg()
\end{itemize}
**PtArg_t**

*Argument structure used for getting and setting widget resources*

**Synopsis:**

```c
typedef struct Pt_arg_entry {
    long type;
    long value;
    long len;
} PtArg_t;
```

**Description:**

You use the `PtArg_t` structure extensively when dealing with widget resources. It’s the first argument in the `PtSetArg()` macro.

This structure contains at least the following members:

- **type**: The resource type (for example, `Pt_ARG_TEXT_STRING`) to be set or queried.
- **value**: Either the value to set the resource to or, if you’re querying the resource, the address of a pointer.
- **len**: The purpose of this member is determined by the Pt type of the resource.

For more information, see the Manipulating Resources in Application Code chapter of the Photon *Programmer’s Guide*.

**Classification:**

Photon

**See also:**

- `Pt_ARG()`, `PtCreateWidget()`, `PtGetResources()`, `PtSetArg()`, `PtSetResources()`


**Pt_ARG()**

*Macro for creating statically initialized argument lists*

**Synopsis:**

```c
#define Pt_ARG( type, value, len ) { ... }
```

**Library:**

`ph`

**Description:**

This macro lets you have statically initialized argument lists as an alternative to using `PtSetArg()`. For example, instead of:

```c
PhPoint_t pos;
PtArg_t args[ 2 ];
pos.x = 100;
pos.y = 150;

PtSetArg( &args[0], Pt_ARG_POS, &pos, 0 );
PtSetArg( &args[1], Pt_ARG_TEXT_STRING, "Blah", 0 );
PtCreateWidget( PtLabel, NULL, 2, args );
```

you can write:

```c
static const PhPoint_t pos = { 100, 150 }; static const PtArg_t args[] = {
    Pt_ARG( Pt_ARG_POS, &pos, 0 ),
    Pt_ARG( Pt_ARG_TEXT_STRING, "Blah", 0 )
};

PtCreateWidget( PtLabel, NULL,
                   sizeof(args) / sizeof(args[0]), args );
```

This makes adding or removing items easier and safer because the compiler counts the items in the array for you. And as a bonus, it generates less code than the first version.

If you have to calculate some of the values at runtime, you’ll need to use `PtSetArg()` to initialize the argument list.
**Pt_ARG()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**Caveats:**

`Pt_ARG()` is a macro.

**See also:**

`PtArg_t, PtGetResource(), PtGetResources(), PtSetArg(), PtSetResource(), PtSetResources()`

Synopsis:

```c
void PtBkgdHandlerProcess( void );
```

Library:

```c
ph
```

Description:

This function processes all outstanding Photon events, calling `PtProcessEvent()` for each event. You should call this routine periodically during a costly or complex processing loop when you won’t be giving control to the widget library. This gives the widget library an opportunity to redraw widgets that have been damaged or exposed if, for example, the user drags a window around.

It’s safe to call `PtBkgdHandlerProcess()` in callbacks, work procedures, and input procedures, but *not* in a widget’s Draw method or a `PtRaw` widget’s drawing function.

Examples:

```c
{  
    int done = 0;
    while ( !done )  
    {  
        /* Handle all pending Photon events */
        PtBkgdHandlerProcess( );
        /* Do some work, setting done if finished */
    }
}
```

Classification:

Photon
PtBkgdHandlerProcess()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:
Parallel Operations chapter of the Photon Programmer’s Guide
PtBlit()
Blit an area within a widget

Synopsis:

```c
int PtBlit( PtWidget_t *widget,
           PhRect_t const *source,
           PhPoint_t const *delta );
```

Library:

ph

Description:

This function blits the area that’s defined by the PhRect_t structure pointed to by source (relative to the widget’s origin) by an offset specified by the PhPoint_t structure pointed to by delta. Effects of the blit are limited to only the visible portions of the widget’s canvas.

Returns:

0 Success.

-1 The blit failed, possibly because the widget wasn’t realized or the Photon Manager wasn’t running.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

`PhBlit()`, `PhPoint_t`, `PhRect_t`, `PtClippedBlit()`
**PtBlockAllWindows()**

*Block all windows except the one with a given widget*

### Synopsis:

```c
PtBlockedList_t *PtBlockAllWindows(
    PtWidget_t *skip,
    unsigned short cursor,
    PgColor_t cursor_color);
```

### Library:

`ph`

### Description:

`PtBlockAllWindows()` blocks all windows belonging to the calling application, except the window that contains the widget pointed to by `skip` (if not NULL).

If `cursor` isn’t zero, this function sets the windows’ cursors to that value and their cursor color to `cursor_color` (if different from `Pg_TRANSPARENT`). `Ph_CURSOR_NOINPUT` is a typical choice for the cursor; for a list of the other cursors, see `<photon/PhCursor.h>`.

### Returns:

A pointer to a control structure that can be passed to `PtUnblockWindows()` to undo the changes, or NULL if an error occurred or there was nothing to do.

### Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtBlockAllWindows()

See also:

ApModalWait(), PgColor_t, PtBlockWindow(), PtMakeModal(), PtUnblockWindows()

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
**Synopsis:**

```
PtBlockedList_t *PtBlockWindow(
    PtWidget_t *window,
    unsigned short cursor,
    PgColor_t cursor_color);
```

**Library:**

ph

**Description:**

`PtBlockWindow()` blocks the given \emph{window} (if not NULL).

If \emph{cursor} isn’t zero, this function sets the window’s cursor to that value and its cursor color to `cursor_color` (if different from `Pg_TRANSPARENT`). `Ph_CURSOR_NOINPUT` is a typical choice for the cursor; for a list of the other cursors, see `<photon/PhCursor.h>`.

**Returns:**

A pointer to a control structure that can be passed to `PtUnblockWindows()` to undo the changes, or NULL if an error occurred or there was nothing to do.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtBlockWindow()

See also:

ApModalWait(), PgColor_t, PtBlockAllWindows(), PtMakeModal(), PtUnblockWindows()

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
**PtCalcAbsPosition()**

*Calculate the position of a widget based on a position and another widget*

**Synopsis:**

```c
int PtCalcAbsPosition( PtWidget_t *reference,
                       PhPoint_t const *pos,
                       PhDim_t const *dim,
                       PhPoint_t *new_pos);
```

**Library:**

`ph`

**Description:**

This function calculates the position of a new widget with dimensions `dim`, according to the `reference` widget and `pos` values passed.

The `reference` argument is a pointer to a widget you wish to position the new widget relative to. The `pos` argument is an offset from the reference widget or from the top left corner of the screen. The `dim` argument gives the dimensions of the new widget to be positioned and must not be NULL. The `new_pos` argument must be a pointer to a `PhPoint_t` structure. The calculated position is stored in it.

The position is calculated as follows:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Position</th>
<th>Position returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>center of the screen</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>offset <code>pos</code> from the top left corner of the screen</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>center of <code>reference</code> widget</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>offset <code>pos</code> from the <code>reference</code> widget’s top left corner</td>
</tr>
</tbody>
</table>
Returns:

0    Success
-1   An error occurred. The dim or new_pos values might have been NULL.

Examples:

If you have a main application window and want to center a dialog on it, you could do the following:

```c
PtWidget_t  *window;       // The main window
PhDim_t      dim = { 100, 100 }; // The size the dialog
                           // is going to be
PhPoint_t    pos;          // The position of the dialog,
                           // to be determined
int          err;
...
// make a main window
window = PtCreateWidget(PtWindow, Pt_NO_PARENT,
                        n_args, args);
...
er = PtCalcAbsPosition(window, NULL, &dim, &pos);
...
// Create the dialog and position it at 'pos' --
// it will be centered on the window.
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
PtCalcAbsPosition()

See also:

PhDim_t, PhPoint_t
**PtCalcCanvas()**

*Calculate the drawable canvas for a widget*

**Synopsis:**

```c
PhRect_t *PtCalcCanvas( PtWidget_t *widget,
                        PhRect_t *canvas_rect);
```

**Library:**

`ph`

**Description:**

This function determines the canvas rectangle for the specified widget and caches it in the widget’s internal memory. This canvas rectangle describes the area *inside* the widget’s border and any margins.

If `canvas_rect` isn’t NULL, `PtCalcCanvas()` copies the canvas rectangle into the `PhRect_t` structure it points to.

**Returns:**

A pointer to the `PhRect_t` structure that defines the canvas. If `canvas_rect` is non-NULL, this is the same pointer as `canvas_rect`.

**CAUTION:** If you pass NULL for `canvas_rect`, `PtCalcCanvas()` returns a pointer into the widget’s internal memory. Don’t modify the contents of the structure or free the memory.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
PtCalcCanvas()

See also:

PhRect_t, PtWidgetExtent()
**PtCalcSurface()**

*Force a control surface to calculate its geometry*

**Synopsis:**

```c
void PtCalcSurface( PtWidget_t *widget,
                    PtSurface_t *surface );
```

**Library:**

`ph`

**Description:**

`PtCalcSurface()` forces a control surface to calculate its geometry.

The `widget` argument specifies the widget owning the surface, while `surface` points to the data structure that describes the control surface. Both pointers must not be NULL. This function is useful if the geometry of one surface depends on that of another.

This function call amounts to a no-op if the geometry for the specified surface has already been calculated for this pre/post-extent cycle.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtCalcSurfaceByAction()`, `PtCalcSurfaceById()`, `PtSurfaceCalcBoundingBox()`, `PtSurfaceCalcBoundingBoxById()`, `PtSurfaceExtent()`, `PtSurfaceExtentById()`, `PtSurfaceRect()`, `PtSurfaceRectById()`
Control Surfaces chapter of the Photon *Programmer's Guide*
**PtCalcSurfaceByAction()**

*Force all control surfaces associated with an action to calculate their geometry*

**Synopsis:**

```c
void PtCalcSurfaceByAction(
    PtWidget_t *widget,
    PtWidgetClassRef_t const *cref,
    ushort_t action_id);
```

**Library:**

`ph`

**Description:**

`PtCalcSurfaceByAction()` forces all surfaces belong to the given widget that are associated with an action to calculate their geometry. The `cref` and `action_id` specify the class and manifest of the action associated with surfaces to be calculated.

The geometry is calculated only for surfaces that haven’t already calculated it for this pre/post-extent cycle.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtCalcSurface()`, `PtCalcSurfaceById()`, `PtSurfaceCalcBoundingBox()`, `PtSurfaceCalcBoundingBoxById()`, `PtSurfaceExtent()`, `PtSurfaceExtentById()`, `PtSurfaceRect()`, `PtSurfaceRectById()`
PtCalcSurfaceByAction()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtCalcSurfaceById()**

*Force the control surface with a given ID to calculate its geometry*

**Synopsis:**

```c
PtSurface_t *PtCalcSurfaceById(
    PtWidget_t *widget,
    uchar_t surface_id);
```

**Library:**

`ph`

**Description:**

`PtCalcSurfaceById()` forces a control surface to calculate its geometry.

The `widget` argument specifies the widget owning the surface, while `surface_id` specifies the numeric ID of the surface to calculate. This function is useful if the geometry of one surface depends on that of another.

This function call amounts to a no-op if the geometry for the specified surface has already been calculated for this pre/post-extent cycle.

**Returns:**

A pointer to the specified surface on success, or NULL if the surface couldn’t be found.

Since control surfaces are maintained internally as an array, it’s not uncommon for them to shift around in memory as surfaces are added and removed, thereby possibly invalidating a pointer returned by this function. As such, all surface pointers should be regarded as transient, and you should retrieve an updated copy whenever there is a chance that the widget’s surface configuration might have changed.
PtCalcSurfaceById()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtCalcSurface(), PtCalcSurfaceByAction(),
PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById(),
PtSurfaceExtent(), PtSurfaceExtentById(), PtSurfaceRect(),
PtSurfaceRectById()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtCancelDnd()**

Cancel a drag-and-drop operation

**Synopsis:**

```c
int PtCancelDnd( PhRid_t rid,
                unsigned flags,
                PhPoint_t const *pos,
                unsigned ig,
                unsigned long handle );
```

**Arguments:**

- **rid**
  The region ID.

- **flags**
  A combination of the following bits:
  
  - Pt_DND_SILENT — the initiator gets no notifications of drag-and-drop progress (valid only if no request data types were added to the `PtTransportCtrl_t` structure).
  
  - Pt_DND_LOCAL — the drop can occur only within the context of the application that initiated the drag-and-drop. That is to say that the user can’t drop the data on any other application. This is very useful for allowing the dragging and dropping of private data or pointer references that are meaningful only within a single application’s context.

- **pos**
  The position where the drag started.

- **ig**
  The input group.

- **handle**
  A number that you can use to identify a transaction.

**Library:**

ph

**Description:**

`PtCancelDnd()` cancels a drag-and-drop operation.
**PtCancelDnd()**

**Returns:**

- 0  Success.
- -1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhPoint_t, PtCreateTransportCtrl(), PtDndFetch_t, PtDndSelect(), PtTransportCtrl_t, PtTransportType()

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PtChannelCreate()**

*Make sure the widget library is using a channel*

**Synopsis:**

```c
int PtChannelCreate( void );
```

**Library:**

ph

**Description:**

In QNX 4, this function simply returns 1.

In QNX Neutrino, this function makes sure that the widget library is using a channel (rather than realtime signals) for notification. It returns the channel number (or -1 on failure).

**Returns:**

There’s no such thing as a channel in QNX 4, so this function simply returns 1.

In QNX Neutrino, this function returns the channel number, or -1 on failure.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtCheckSurfaces()**
*Match an event with the control surfaces belonging to a widget*

**Synopsis:**

```c
int PtCheckSurfaces( PtWidget_t *widget, PhPoint_t *point, PhEvent_t *event );
```

**Library:**

`ph`

**Description:**

This function matches the given event with the control surfaces belonging to `widget`. If the event corresponds to a control surface, the surface’s action is invoked.

`PtCheckSurfaces()` is usually called only from `PtEventHandler()`, although you can call it with a fake event if you need to.

**Returns:**

`Pt_CONTINUE`

The event wasn’t processed by a control surface.

`Pt_END` The event was processed by a control surface.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtCheckSurfaces()

See also:

PhEvent_t, PtEventHandler(), PhPoint_t

Control Surfaces chapter of the Photon Programmer's Guide
**Synopsis:**

```c
int PtChildType( PtWidget_t *parent,
                 PtWidget_t *child );
```

**Library:**

ph

**Description:**

This function returns Pt_IMMEDIATE_CHILD if the provided child is the immediate child of parent or Pt_SUBORDINATES_CHILD if the child is a child of a procreated child of parent. Otherwise Pt_FALSE is returned.

**Returns:**

- **Pt_IMMEDIATE_CHILD**
  - The provided child is the immediate child of parent.
- **Pt_SUBORDINATES_CHILD**
  - The child is a child of a procreated child of parent.
- **Pt_FALSE**
  - Neither of the above is true.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtChildType()

See also:

PtWidgetBrotherBehind(), PtWidgetBrotherInFront(),
PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetParent()

“Ordering widgets” in the Managing Widgets in Application Code
chapter of the Photon Programmer’s Guide
Synopsis:

    int PtClearWidget(PtWidget_t *widget);

Arguments:

    widget       A pointer to the container widget that you want to clear.

Library:

    ph

Description:

This function destroys all the nonprocreated widgets within the specified container-class widget.

Procreated widgets are widgets that are created as part of a compound widget, as opposed to those that your application creates. For example, a PtScrollContainer can have:

- procreated children — the scrollbars and the basic canvas.
- nonprocreated children — the widgets that your application adds inside the PtScrollContainer. These are the only children that PtClearWidget() destroys.

If the specified widget isn’t a container, no action is taken.

Returns:

    0       Successful completion.
    -1      An error occurred (the widget wasn’t a container).

Examples:

```c
#include <stdlib.h>
#include <Pt.h>

int main()
{
    PtWidget_t *group, *window;
```
PtClearWidget()

PtArg_t argt;
PtArg_t args[3];
PhPoint_t pos = { 10, 10 };

if (PtInit(NULL) == -1)
    exit(EXIT_FAILURE);

PtSetArg( &argt, Pt_ARG_POS, &pos, 0 );
window = PtCreateWidget( PtWindow, Pt_NO_PARENT, 1, &argt );

PtSetArg( &args[0], Pt_ARG_POS, &pos, 0 );
PtSetArg( &args[1], Pt_ARG_GROUP_ORIENTATION, Pt_GROUP_VERTICAL, 0 );
group = PtCreateWidget( PtGroup, window, 1, &argt );

PtSetArg( &argt, Pt_ARG_TEXT_STRING, "Button", 0 );
PtCreateWidget( PtButton, group, 1, &argt );

// using same string as previous button...
PtCreateWidget( PtButton, group, 1, &argt );
PtCreateWidget( PtButton, group, 1, &argt );
PtRealizeWidget( window );
PtContainerHold( group );
PtClearWidget( group );
// destroys all widgets within the group, clearing it...

// add new children to the group
PtSetArg( &argt, Pt_ARG_TEXT_STRING, "New Button", 0 );
PtRealizeWidget( PtCreateWidget( PtButton, group, 1, &argt ) );
PtSetArg( &argt, Pt_ARG_TEXT_STRING, "New Button2", 0 );
PtRealizeWidget( PtCreateWidget( PtButton, group, 1, &argt ) );

// force the group to re-align its children and resize.
PtExtentWidget( group );
PtContainerRelease( group );
PtMainLoop();
return EXIT_SUCCESS;
PtClearWidget()

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtContainerHold(), PtContainerRelease(), PtDestroyWidget(), PtExtentWidget(), PtWidgetChildBack()
PtClipAdd()
Add a clipping rectangle to the stack

Synopsis:

```c
int PtClipAdd( PtWidget_t *widget,
               PhRect_t *rect );
```

Description:

This function adds a clipping rectangle to the clipping stack. The rectangle added to the clipping stack is the intersection of the last rectangle on the stack and the rectangle defined in the PhRect_t pointed to by rect.

Prior to entering a widget’s Draw method, the canvas rectangle derived from the PtBasic-class level is pushed onto the clipping stack. This prevents any children from drawing beyond the canvas of the parent container.

A widget can, however, draw beyond its own canvas or extent unless additional clipping is performed. PtAttemptResize() and PtResizeCanvas() set the Pt_UCLIP bit of a widget’s resize flags if the widget requires additional clipping (to prevent it from drawing beyond its own canvas).

Returns:

The current level of stack clipping.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt handler</td>
</tr>
<tr>
<td></td>
<td>Signal handler</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
</tr>
</tbody>
</table>
See also:

\texttt{PhRect\_t, PtCalcCanvas(), PtClipRemove()}

\texttt{PtAttemptResize(), PtResizeCanvas()} in \textit{Building Custom Widgets}
**PtClippedBlit()**

Blit areas within a widget, with clipping

**Synopsis:**

```c
int PtClippedBlit( PtWidget_t *widget,
                  PhTile_t const *src,
                  PhPoint_t const *delta,
                  PhTile_t const *clip );
```

**Library:**

`ph`

**Description:**

This function blits the areas inside `widget` specified by the `PhTile_t` tile list `src` (areas are relative to the widget’s origin) by an offset specified by `delta`. Effects of the blit are limited by the tiles specified by `clip`.

**Returns:**

- 0  Success.
- -1  The blit failed, possibly because the widget wasn’t realized or the Photon Manager wasn’t running.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

1286  Chapter 13 • Pt—Widget Toolkit
PtClippedBlit()

See also:

PhBlit(), PhPoint_t, PhTile_t, PtBlit()
**PtClipRemove()**

*Take a clipping rectangle off the stack*

**Synopsis:**

```c
int PtClipRemove();
```

**Description:**

This function pops the last clipping rectangle off the clipping stack.

**Returns:**

The current level of stack clipping.

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

- `PtCalcCanvas()`, `PtClipAdd()`
- `PtAttemptResize()`, `PtResizeCanvas()` in *Building Custom Widgets*
PtCondTimedWait()  
_block a thread on a conditional variable, with a time limit_

**Synopsis:**

```c
int PtCondTimedWait(
    pthread_cond_t *cond,
    const struct timespec *abstime);
```

**Arguments:**

- `cond` The condition variable to wait on.
- `abstime` A pointer to a `timespec` structure that specifies the absolute time by which the thread must unblock.

**Library:**

`ph`

**Description:**

`PtCondTimedWait()` is an equivalent of `pthread_cond_timedwait()` that uses the Photon library lock instead of a mutex, which has the effect of an implicit `PtLeave()` when you block, and `PtEnter()` when you unblock.

The calling thread is blocked until:

- another thread performs a signal or broadcast on the condition variable (using `pthread_cond_signal()` or `pthread_cond_broadcast()`)
  - Or:
  - the absolute time specified by `abstime` has passed
  - Or:
  - a signal is delivered to the thread.

In all cases, the thread reacquires the Photon library lock before being unblocked.
PtCondTimedWait()

Returns:

- **EOK**: Success, or the call was interrupted by a signal.
- **EAGAIN**: Insufficient system resources are available to wait on the condition.
- **EFAULT**: A fault occurred trying to access the buffers provided.
- **EINVAL**: One or more of `cond`, `mutex` or `abstime` was invalid. Concurrent waits or timed waits on `cond` used different mutexes. The current thread doesn’t own `mutex`.
- **ETIMEDOUT**: The time specified by `abstime` has passed.

Classification:

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

- `PtCondWait()`, `PtEnter()`, `PtLeave()`
- `pthread_cond_timedwait()` in the QNX Neutrino Library Reference
- “Threads” in the Parallel Operations chapter of the Photon Programmer’s Guide
PtCondWait()
Block a thread on a conditional variable

Synopsis:

```
int PtCondWait( pthread_cond_t *cond );
```

Arguments:

- `cond`: The condition variable to wait on.

Library:

```
ph
```

Description:

PtCondWait() is an equivalent of `pthread_cond_wait()` that uses the Photon library lock instead of a mutex, which has the effect of an implicit PtLeave() when you block, and PtEnter() when you unblock.

The calling thread is blocked until:

- another thread performs a signal or broadcast on the condition variable (using `pthread_cond_signal()` or `pthread_cond_broadcast()`)
  Or:

- a signal is delivered to the thread.

In all cases, the thread reacquires the Photon library lock before being unblocked.

The implicit PtLeave() call that PtCondWait() makes before blocking turns the calling thread into an event non-reader. If you passed the Pt_DELAY_EXIT flag to PtEnter() before calling PtCondWait(), it also disables the effect of that flag. Before returning, PtCondWait() turns the thread back into an event reader if it was an event reader before, but will not turn the Pt_DELAY_EXIT flag back on. In particular, this means that if another thread has called PtExit(), this function does not return, even if a third thread signals the condvar.
PtCondWait()

Returns:

- **EOK** Success, or the call was interrupted by a signal.
- **EAGAIN** Insufficient system resources are available to wait on the condition.
- **EFAULT** A fault occurred trying to access the buffers provided.
- **EINVAL** One or more of `cond` or `mutex` is invalid.
  Concurrent waits on `cond` used different mutexes.
  The current thread doesn’t own `mutex`.

Classification:

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

- `PtCondTimedWait()`, `PtEnter()`, `PtLeave()`
- `pthread-cond_wait()` in the QNX Neutrino Library Reference
- “Threads” in the Parallel Operations chapter of the Photon Programmer’s Guide
**PtConnectionAddEventHandlers()**

Add a set of server event handlers to a client connection object

**Synopsis:**

```c
int PtConnectionAddEventHandlers(
    PtConnectionClient_t *connection,
    PtConnectionEventHandler_t const *handlers,
    unsigned nhandlers);
```

**Library:**

ph

**Description:**

This function adds a set of server event handlers to a client connection object. The `handlers[]` argument points to an array of `PtConnectionEventHandler_t` structures:

```c
typedef int PtConnectionEventFunc_t(
    PtConnectionClient_t *connection, void *user_data,
    unsigned long type, void const *msg, unsigned len );

typedef struct Pt_connection_event_handler {
    unsigned long type;
    PtConnectionEventFunc_t *fun;
} PtConnectionEventHandler_t;
```

The array must be sorted with respect to the `type` field; it also must not be destroyed or modified as long as the connection object using it exists.

If you add multiple tables to a connection object, they’re searched in the reverse order: a call to `PtConnectionAddEventHandlers()` can override handlers that were attached by a previous call. An event handler should return Pt_End to “consume” the event, or `Pt_CONTINUE` to continue the search.

A special value of zero in the `type` field means “any type.” When a notification from the server arrives, all the tables are searched for an exact match on the type, and if the event isn’t consumed, the tables are searched again for a handler with type 0.
PtConnectionAddEventHandlers()

Returns:

0   Success.
-1   An error occurred; errno is set.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtConnectionAddMsgHandlers(), PtConnectionNotify(),
PtConnectionReply(), PtConnectionReplymx(),
PtConnectionResizeEventBuffer()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtConnectionAddMsgHandlers()**

Add a set of message handlers to a server connection object

**Synopsis:**

```c
int PtConnectionAddMsgHandlers(
    PtConnectionServer_t *connection,
    PtConnectionMsgHandler_t const handlers[],
    unsigned nhandlers);
```

**Library:**

`ph`

**Description:**

This function adds a set of message handlers to a server connection object. The `handlers[]` argument points to an array of `PtConnectionMsgHandler_t` structures:

```c
typedef void const *PtConnectionMsgFunc_t(
    PtConnectionServer_t *connection, void *user_data,
    unsigned long type, void const *msg, unsigned len,
    unsigned *reply_len);

typedef struct Pt_connection_msg_handler {
    unsigned long type;
    PtConnectionMsgFunc_t *fun;
} PtConnectionMsgHandler_t;
```

These structures describe your message handler functions. Your message handler should have these arguments:

- **connection** a pointer to the connection structure
- **user_data** a pointer to the connection object’s user data. This data is set with `PtConnectionServerSetUserData()`.
- **type** the type of the message
- **msg** a pointer to the message
- **len** the size of `msg`, in bytes
- **reply_len** a pointer to the size of the returned value, in bytes
PtConnectionAddMsgHandlers()

The array must be sorted with respect to the type field; it also must not be destroyed or modified as long as the connection object using it exists.

If you add multiple tables to a connection object, they’re searched in the reverse order: a call to PtConnectionAddMsgHandlers() can override handlers that were attached by a previous call.

A special value of zero in the type field means “any type.” When a message from a client arrives, all the tables are searched for an exact match on the type, and if this search fails, the tables is searched again for a handler with type 0.

A message handler can do one of three things:

- Just return NULL. This means that the handler hasn’t handled the message and another handler should be looked for in the tables.

- Set up a reply message, set *reply_len to the length of it, and return the pointer to the reply.

- Reply to the client by calling PtConnectionReply(), and then return any non-NULL value.

A message handler shouldn’t perform any blocking operations and isn’t allowed to process Photon events (e.g. don’t call PtBkgdHandlerProcess() from an event handler). There’s no way to delay the reply to the client — if none of the handlers returns a non-NULL value, the library sends a zero-length reply.

Returns:

- 0 Success.

- 1 An error occurred; errno is set.

Classification:

Photon
**PtConnectionAddMsgHandlers()**

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

*PtConnectionAddEventHandler(), PtConnectionReply(), PtConnectionReplymx(), PtConnectionSend(), PtConnectionSendmx()*

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionClientDestroy()**

*Destroy a client connection object*

**Synopsis:**

```c
void PtConnectionClientDestroy(
    PtConnectionClient_t *connection);
```

**Library:**

`ph`

**Description:**

This function destroys the client connection object. Attempting to send any messages or notifications to a partner whose connection object has been destroyed isn’t safe; preventing such attempts is the responsibility of a higher level protocol.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtConnectionClientGetUserData(), PtConnectionClientSetError(), PtConnectionClientSetUserData(), PtConnectionServerDestroy()*

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
PtConnectionClientGetUserData()

Get the client’s user data pointer from a connection object

Synopsis:

```c
void *PtConnectionClientGetUserData(
    PtConnectionClient_t *connection);
```

Library:

ph

Description:

A connection object contains a user data pointer that’s passed to its message handlers. This function lets you retrieve that pointer.

Returns:

A `void *` pointer to the data.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtConnectionClientDestroy(), PtConnectionClientSetError(),
PtConnectionClientSetUserData(),
PtConnectionServerGetUserData(),
PtConnectionServerSetUserData()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtConnectionClientSetError()**

*Set the error-handler function for the client-side of a connection*

**Synopsis:**

```c
PtConnectionClientErrorFunc_t *
PtConnectionClientSetError(  
    PtConnectionClient_t *connection,  
    PtConnectionClientErrorFunc_t *func );
```

**Library:**

`ph`

**Description:**

This function sets up an error handling function for the client-side of a connection. The prototype of the handler is:

```c
typedef int PtConnectionClientErrorFunc_t(  
    PtConnectionClient_t *connection,  
    int err,  
    enum PtConnectionClientError where );
```

The `where` argument indicates where the error occurred:

- **Pt_CONNECTION_SEND_FAILED**
  The `MsgSend()` to the server failed.

- **Pt_CONNECTION_REALLOC_REPLY**
  The `realloc()` failed to expand the reply buffer.

- **Pt_CONNECTION_CLIENT_BROKEN**
  The server has died or closed its part of the connection.

The error handler is called when certain errors occur; the handler can return `Pt_CONTINUE` to retry, or `Pt_END` to fail. The default error handler returns `Pt_END`. 
PtConnectionClientSetError()

If your application has created its own channel without
_NTO_CHF_COID_DISCONNECT and _NTO_CHF_DISCONNECT
set, this mechanism won’t work. For more information, see
PhChannelAttach().

Returns:

A pointer to the previous error-handler function.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtConnectionClientDestroy(), PtConnectionClientGetUserData(), PtConnectionClientSetUserData(), PtConnectionServerSetError()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtConnectionClientSetUserData()**

Set the client's user data pointer in a connection object

**Synopsis:**

```c
void PtConnectionClientSetUserData(
    PtConnectionClient_t *connection,
    void *data);
```

**Library:**

ph

**Description:**

A connection object contains a user data pointer that’s passed to its message handlers. This function lets you set that pointer.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtConnectionClientDestroy(), PtConnectionClientGetUserData(), PtConnectionClientSetError(), PtConnectionServerGetUserData(), PtConnectionServerSetUserData()*

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionFindId()**

Find the connector with a given ID

### Synopsis:

```c
PtConnectionClient_t *PtConnectionFindId(
    PhConnectorId_t id,
    int subtype,
    unsigned flags);
```

### Library:

`ph`

### Description:

This function connects to the connector with the given ID, and creates a client-connection object.

The `subtype` and `flags` arguments should be set to zero for now.

### Returns:

A pointer to the client-connection object.

### Classification:

**Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

*PtConnectionFindName(), PtConnectionTmpName(), PtConnectionWaitForName()*

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionFindName()**

*Find the connector with a given name*

**Synopsis:**

```c
PtConnectionClient_t *PtConnectionFindName(
    const char *name,
    int subtype,
    unsigned flags);
```

**Library:**

ph

**Description:**

This function connects to the connector with the given name, and creates a client-connection object.

The `subtype` and `flags` arguments should be set to zero for now.

**Returns:**

A pointer to the client-connection object.

**Classification:**

Photon

---

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PtConnectionFindId()`, `PtConnectionTmpName()`, `PtConnectionWaitForName()`
- “Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionFlush()**

*Send all pending notifications to the client*

**Synopsis:**

```c
int PtConnectionFlush(
    PtConnectionServer_t *connection
);```

**Library:**

ph

**Description:**

This function waits for a message from the client and sends all the pending notifications in the reply. If there are no notifications to be sent, the function doesn’t wait.

This function may process Photon events or other messages while waiting for the message from the client. If a received Photon event invokes a callback that calls `PtConnectionFlush()` on the same connection again, the second call fails with `errno` set to EBUSY.

**Returns:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred; <code>errno</code> is set.</td>
</tr>
</tbody>
</table>

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtConnectionFlush()

See also:

PtConnectionNotify(), PtConnectionReply(), PtConnectionReplymx(),
PtConnectionSend(), PtConnectionSendmx()

“Connections” in the Interprocess Communication chapter of the
Photon Programmer’s Guide
Synopsis:

```c
int PtConnectionNotify(
    PtConnectionServer_t *connection,
    unsigned long type,
    void const *message,
    unsigned length,
    unsigned flags);
```

Library:

`ph`

Description:

This function sends an event notification to the client. The possible values of `flags` include:

- **Pt_CONNECTION_NOTIFY_NOFLUSH**
  
  Unless you set this flag, `PtConnectionNotify()` calls `PtConnectionFlush()` if there isn’t enough room for the notification in the buffer.

- **Pt_CONNECTION_NOTIFY_RESIZE**
  
  Resize the buffer if necessary.

- **Pt_CONNECTION_NOTIFY_FLUSH**
  
  If you set this flag, `PtConnectionNotify()` calls `PtConnectionFlush()` after appending the notification to the buffer.

If the server sets both `Pt_CONNECTION_NOTIFY_RESIZE` and `Pt_CONNECTION_NOTIFY_NOFLUSH`, it might run out of memory if the client is slow or unresponsive. If you do set both bits, make sure that your protocol prevents you from sending too many notifications. (For example, require the client to send an acknowledgment for each received notification, and don’t send a new notification unless the previous one has been acknowledged.)
PtConnectionNotify()

Returns:

  0  Success.
-1  An error occurred; errno is set.

Classification:

  Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtConnectionAddEventHandlers(), PtConnectionAddMsgHandlers(), PtConnectionFlush(), PtConnectionReply(), PtConnectionReplymx(), PtConnectionResizeEventBuffer(), PtConnectionSend(), PtConnectionSendmx()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PtConnectionReply(
    PtConnectionServer_t *connection,
    int len,
    void const *buf );
int PtConnectionReplymx(
    PtConnectionServer_t *connection,
    int rparts,
    iov_t *riov );
```

**Library:**

`ph`

**Description:**

The server in a connection uses these functions to reply to a message sent by a client with a call to `PtConnectionSend()` or `PtConnectionSendmx()`.

Under QNX 4, the `iov_t` type is a synonym for `struct _mxfer_entry`. The `riov[0]` entry is reserved for headers used internally by the library; the “real” reply buffer are defined by the contents of `riov[1]` through `riov[rparts-1].`

**Returns:**

- 0 Success.
- -1 An error occurred; `errno` is set.

**Classification:**

Photon

**Safety**

Interrupt handler  No

`continued…`
PtConnectionReply(), PtConnectionReplymx()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtConnectionFlush(), PtConnectionNotify(), PtConnectionSend(), PtConnectionSendmx()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtConnectionResizeEventBuffer()**

Resize the buffer used to store notifications

**Synopsis:**

```c
int PtConnectionResizeEventBuffer(
    PtConnectionServer_t *connection,
    unsigned length);
```

**Library:**

ph

**Description:**

This function allocates or reallocates a buffer that notifications are stored in until the client responds to a pulse.

**Returns:**

- 0 Success.
- -1 An error occurred; errno is set.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtConnectionAddEventHandlers()*，*PtConnectionNotify()*

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionSend(), PtConnectionSendmx()**

Send a message to a server

**Synopsis:**

```c
int PtConnectionSend(  
    PtConnectionClient_t *connection,  
    unsigned long type,  
    const void *smsg,  
    void *rmsg,  
    unsigned snbytes,  
    unsigned rnbytes );
```

```c
int PtConnectionSendmx(  
    PtConnectionClient_t *connection,  
    unsigned long type,  
    int sparts,  
    iov_t *siov,  
    int rparts,  
    iov_t *riov );
```

**Library:**

ph

**Description:**

The client of a connection uses these functions to send a message to the server. The server uses `PtConnectionReply()` and `PtConnectionReplymx()` to reply.

The arguments are similar to kernel functions, except that the message type isn’t considered to be part of the message.

Under QNX 4, the `iov_t` type is a synonym for `struct _mxfer_entry`. The `siov[0]` and `riov[0]` entries are reserved for headers used internally by the library; the “real” message and reply buffer are defined by the contents of `siov[1]` through `siov[sparts-1]` and `riov[1]` through `riov[rparts-1]`.

**Returns:**

The number of bytes in the reply, or -1 if the send failed (`errno` is set).
**PtConnectionSend(), PtConnectionSendmx()**

**Classification:**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtConnectionFlush(), PtConnectionNotify(), PtConnectionReply(), PtConnectionReplymx()*

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionServerDestroy()**

Destroy a server connection object

**Synopsis:**

```c
void PtConnectionServerDestroy(
    PtConnectionServer_t *connection);
```

**Library:**

`ph`

**Description:**

This function destroys the server connection object. Attempting to send any messages or notifications to a partner whose connection object has been destroyed isn’t safe; preventing such attempts is the responsibility of a higher level protocol.

**Classification:**

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No

**See also:**

`PtConnectionClientDestroy()`, `PtConnectionServerGetUserData()`, `PtConnectionServerSetError()`, `PtConnectionServerSetUserData()`

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionServerGetUserData()**

*Get the server’s user data pointer from a connection object*

**Synopsis:**

```c
void *PtConnectionServerGetUserData(
    PtConnectionServer_t *connection);
```

**Library:**

ph

**Description:**

A connection object contains a user data pointer that is passed to its message handlers. This function lets you retrieve that pointer.

**Returns:**

A `void *` pointer to the data.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PtConnectionClientGetUserData()`
- `PtConnectionClientSetUserData()`, `PtConnectionServerDestroy()`, `PtConnectionServerSetError()`, `PtConnectionServerSetUserData()`

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionServerSetError()**

*Set the error-handler function for the server-side of a connection*

**Synopsis:**

```c
PtConnectionServerErrorFunc_t *
PtConnectionServerSetError(
    PtConnectionServer_t *connection,
    PtConnectionServerErrorFunc_t *func);
```

**Library:**

`ph`

**Description:**

This function sets up an error handling function for the server-side of a connection. The prototype of the handler is:

```c
typedef int PtConnectionServerErrorFunc_t(
    PtConnectionServer_t *connection,
    int err,
    enum PtConnectionServerError where);
```

The `where` argument indicates where the error occurred:

- **Pt_CONNECTION_REPLY_FAILED**
  The `MsgReply()` to the client failed.

- **Pt_CONNECTION_REALLOC_RECEIVE**
  The `realloc()` failed to expand the receive buffer.

- **Pt_CONNECTION_SERVER_BROKEN**
  The client has died or closed its part of the connection.

- **Pt_CONNECTION_MSGREAD_FAILED**
  The `MsgRead()` from the client failed.

The error handler is called when certain errors occur; the handler can return Pt_CONTINUETO retry, or Pt_END to fail. The default error handler returns Pt_END.
If your application has created its own channel without
_NONO_CHF_COID_DISCONNECT and _NONO_CHF_DISCONNECT
set, this mechanism won’t work. For more information, see
PhChannelAttach().

Returns:
A pointer to the previous error-handler function.

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtConnectionClientSetError(), PtConnectionServerDestroy(),
PtConnectionServerGetUserData(),
PtConnectionServerSetUserData()

“Connections” in the Interprocess Communication chapter of the
Photon Programmer’s Guide
**PtConnectionServerSetUserData()**

Set the server’s user data pointer in a connection object

**Synopsis:**

```c
void PtConnectionServerSetUserData(
    PtConnectionServer_t *connection,
    void *data );
```

**Library:**

`ph`

**Description:**

A connection object contains a user data pointer that is passed to its message handlers. This function lets you set that pointer.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtConnectionClientGetUserData()`, `PtConnectionClientSetUserData()`, `PtConnectionServerDestroy()`, `PtConnectionServerGetUserData()`, `PtConnectionServerSetError()`

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**PtConnectionTmpName()**

Create a temporary name for a server

**Synopsis:**

```c
char *PtConnectionTmpName( char name[32] );
```

**Library:**

`ph`

**Description:**

This function creates a temporary name that can be given to a new server. The `name` argument is the buffer in which to build the name.

If the same client may generate more temporary names in the future, the server should destroy its connector as soon as the client has connected — currently, there’s only room for a few unique temporary names per client.

**Returns:**

A pointer to `name`, or NULL if a name couldn’t be generated.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtConnectionFindId()`, `PtConnectionFindName()`, `PtConnectionWaitForName()`

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
PtConnectionWaitForName()
Try to connect to the server with a given name

Synopsis:

typedef int PtConnectionWaitFunc_t(
    PtConnectionClient_t *connection,
    void *user_data,
    unsigned n);

int PtConnectionWaitForName(
    const char *name,
    int subtype,
    unsigned flags,
    PtConnectionWaitFunc_t *cb,
    void *usrdata);

Library:

ph

Description:

This function lets you set up a timer that periodically tries to connect to a server. This mechanism is meant to be used when you might need to spawn the server and then wait until it’s ready to accept a connection.

Each attempt consists of a call to PtConnectionFindName(), followed by a call to your callback. The callback should check its connection argument: if it’s non-NULL, it means that this attempt was successful. The callback should then set up the connection object and return zero.

If the connection argument to your callback is NULL, the connection couldn’t be established this time. The callback should return the number of milliseconds to wait before making another attempt. Returning zero means “don’t retry.”

The n argument to the callback is just a serial number that gets incremented after each failed attempt. It can be useful if you want to spawn the server after the first attempt fails:

int connection_callback(
    PtConnectionClient_t *connection,
    void *user_data, unsigned n) {
if ( connection )
    PtConnectionAddEventHandlers( connection, tab, N );
else
    switch ( n ) {
        default :
            return 200; /* retry after 1/5 of a second */
        case 0 :
            /* First attempt failed -- spawn the server */
            if ( PtSpawn( ... ) > 0 )
                return 1000; /* Retry after one second */
            else
                warn( "Couldn't spawn the server" );
        case 9 :
            /* ten attempts failed -- give up */
    }
    return 0;
}

(A better way would be to let it keep trying until the spawned server terminates rather than just have a fixed number of retries — but we’ll leave that as an exercise for the reader.)

Returns:

0 Success.

-1 An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtConnectionWaitForName()

See also:

PtConnectionFindId(), PtConnectionFindName(), PtConnectionTmpName(),

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
PtConnectorCreate()
Create a connector

Synopsis:

```c
PtConnector_t *PtConnectorCreate(
    const char *name,
    PtConnectorCallbackFunc_t *cb,
    void *data);
```

Arguments:

- `name` The name to register, or NULL for a nameless connector. If `name` isn’t NULL, it must be unique; an attempt to create a connector with a name that’s already registered will fail.
- `cb` A callback to call for each client that connects. `PtConnectorCallbackFunc_t` is a function type:

  ```c
typedef void PtConnectorCallbackFunc_t(
    PtConnector_t *,
    PtConnectionServer_t *,
    void *);
```

  The callback is called each time a new server connection object is created as a response to a connection request from a client.
- `data` Extra data to pass to the callback as the last argument.

Library:

- `ph`

Description:

This function creates a connector.

Returns:

A pointer to a `PtConnector_t` structure that describes the connector.
PtConnectorCreate()

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtConnectorDestroy(), PtConnectorGetId()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
PtConnectorDestroy()

Synopsis:

```c
int PtConnectorDestroy( PtConnector_t *connector );
```

Library:

ph

Description:

This function destroys the given connector.

Destroying a connector doesn’t affect existing connections. If a server is only capable of talking to one client, it’s a good idea to destroy the connector as soon as the client has connected.

Returns:

0 Success.

-1 An error occurred; errno is set.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtConnectorCreate(), PtConnectorGetId()

“Connections” in the Interprocess Communication chapter of the Photon Programmer’s Guide
**PtConnectorGetId()**

*Get the ID of a connector*

**Synopsis:**

```c
PhConnectorId_t PtConnectorGetId(
    PtConnector_t const *connector );
```

**Library:**

`ph`

**Description:**

This function obtains the connector ID that a client can pass to `PtConnectionFindId()` in order to connect.

**Returns:**

The connector ID.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtConnectorCreate()`, `PtConnectorDestroy()`

“Connections” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PtConsoleSwitch( int console );
```

**Library:**

ph

**Description:**

This function causes PWM to switch the current display to the virtual console `console` (where `console` is a number in the range 0 to 8):

```
  0  1  2
  3  4  5
  6  7  8
```

*Numbering for Photon's virtual consoles.*

Virtual consoles are numbered from 0 through 8, with 0 at the top left, and 8 at the lower right. The coordinates of the top left corner of console 0 are (0,0). The coordinates of the other consoles depend on your screen size.

**Returns:**

- 0 Success.
- -1 An error occurred. Check the value of *errno* for more information.

**Classification:**

Photon
PtConsoleSwitch()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtWindowConsoleSwitch()

Window Management chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
PtWidget_t * PtContainerBox( PtWidget_t *container,
                           PtWidget_t *start,
                           PhRect_t const *rect );
```

Library:

```
ph
```

Description:

This function returns a pointer to the first widget within the specified container that intersects with the rectangle defined in the `PhRect_t` structure pointed to by `rect`.

The widget identified by `start` tells the function where to start looking for intersections. First, the function checks `start`’s brother behind. Then, it then checks the brother behind that, and so on.

Returns:

If no widget after `start` intersects with `rect` or if the provided container pointer doesn’t actually point to a container, the function returns NULL.

Examples:

```c
PtWidget_t *target_widget, *my_pane;
...

// In my_pane’s RAW callback:
int
my_raw_cb( PtWidget_t *container, void *data,
           PtCallbackInfo_t *cbinfo )
{
  ...
  rect = PhGetRects( cbinfo->event );
  if( target_widget = PtContainerBox( widget,
                        PtWidgetChildFront( container ), &rect ) )
    PtDestroyWidget( target_widget );
  ...
```
PtContainerBox()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhGetRects(), PhRect_t, PtContainerHit(), PtWidgetChildFront()
PtContainerFindFocus()

Find the currently focused widget in the same disjoint widget as a widget

Synopsis:

```c
Pt.Widget_t *PtContainerFindFocus(
    Pt.Widget_t *family_member);
```

Library:

ph

Description:

This function finds the focused widget for the disjoint widget that contains `family_member`, or the focused widget in `family_member` if `family_member` is a disjoint widget. In other words, in a multi-window application, the function will find the focused widget in the same window as `family_member`.

Returns:

A pointer to the currently focused widget, or NULL if `family_member` is passed as NULL.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtContainerGiveFocus()`, `PtContainerNullFocus()`, `PtContainerFocusNext()`, `PtContainerFocusPrev()`, `PtGlobalFocusNext()`, `PtGlobalFocusNextFrom()`, `PtGlobalFocusPrev()`, `PtGlobalFocusPrevFrom()`, `PtIsFocused()`
**PtContainerFocusNext()**  
*Give focus to the next Pt_GETS_FOCUS widget*

**Synopsis:**

```c
PtWidget_t *PtContainerFocusNext(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

This function gives focus to the next widget that has Pt_GETS_FOCUS set in its Pt_ARG_FLAGS and is in the same container as the currently focused widget in `widget`'s family. If no widget has the Pt_GETS_FOCUS flag set, the container’s focus is nullified (that is, none of its children will have focus).

The `event` argument is a pointer to a `PhEvent_t` structure that describes the event that’s passed to the lost-focus callback of the widget losing focus and to the got-focus callback of the widget getting focus.

If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

A pointer to the newly focused widget.

**Examples:**

See `PtContainerGiveFocus()`.

**Classification:**

Photon
PtContainerFocusNext()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhEvent_t, PtContainerFocusPrev(), PtContainerGiveFocus(), PtContainerNullFocus()
**PtContainerFocusPrev()**

*Give focus to the previous Pt_GETS_FOCUS widget*

**Synopsis:**

```c
PtWidget_t *PtContainerFocusPrev(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

This function gives focus to the previous widget that has Pt_GETS_FOCUS set in its Pt_ARG_FLAGS and is in the same container as the currently focused widget in `widget`'s family. If no widget has the Pt_GETS_FOCUS flag set, the container’s focus is nullified (that is, none of its children will have focus).

The `event` argument is a pointer to a `PhEvent_t` structure that describes the event that’s passed to the lost-focus callback of the widget losing focus and to the got-focus callback of the widget getting focus.

If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

A pointer to the newly focused widget.

**Examples:**

See `PtContainerGiveFocus()`.

**Classification:**

Photon
PtContainerFocusPrev()

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PhEvent_t, PtContainerFocusNext(), PtContainerGiveFocus(), PtContainerNullFocus(), PtWidgetExtent()
**PtContainerGiveFocus()**

*Give focus to a widget*

**Synopsis:**

```c
PtWidget_t *PtContainerGiveFocus(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

This function gives focus to the specified widget, even if the widget’s Pt(GETS_FOCUS) flag isn’t set. *PtContainerGiveFocus()* is the same as *PtGiveFocus().*

If the widget is a *PtWindow*, use *PtWindowFocus()* instead of this function — see the Photon *Widget Reference.*

The *event* argument is a pointer to a *PhEvent_t* structure that describes the event that will be passed to the lost-focus callback of the widget losing focus and to the got-focus callback of the widget getting focus. If *event* is NULL, this function generates a *PhEvent_t* structure filled with zeros for you.

**Returns:**

A pointer to the newly focused widget. This is usually the same as the *widget* argument, but it could be NULL if one of the following is true:

- The *widget* argument is NULL.
- The given widget is disjoint (e.g. a window).
- The widget is blocked; that is, it has Pt(BLOCKED) set in its *Pt_ARG_FLAGS resource* (see *PtWidget* in the Photon *Widget Reference*).
- The widget has been destroyed before the attempt to give it focus.
PtContainerGiveFocus()

- The event passed (if not NULL) has already caused focus to change, as indicated by:

  \[ \text{event->processing_flags} \& \text{Ph\_DIRECTED\_FOCUS} \]

This function could also return a pointer to a different widget if that widget for some reason refused to relinquish focus (i.e. its _Pt_CB\_LOST\_FOCUS callback returned Pt\_END — see PtBasic in the Photon Widget Reference). This usually happens if the requirements of an entry field haven’t been met and must be met before any other action can be taken.

The widget library never refuses to relinquish focus. If a widget does this, it’s because of a _Pt_CB\_LOST\_FOCUS callback in your application.

Examples:

```c
#include <stdlib.h>
#include <Pt.h>

int first( PtWidget_t *widget, void *data, 
            PtCallbackInfo_t *cbinfo )
{
    //like selecting texta
    PtWidget_t *text = (PtWidget_t *)data; 
    PtContainerGiveFocus( text, cbinfo->event );
    return Pt_CONTINUE;
}

int next( PtWidget_t *widget, void *data, 
          PtCallbackInfo_t *cbinfo )
{
    //like hitting tab
    PtGlobalFocusNext( widget, cbinfo->event );
    return Pt_CONTINUE;
}

int prev( PtWidget_t *widget, void *data, 
          PtCallbackInfo_t *cbinfo )
{
    //like hitting shift-tab
    PtGlobalFocusPrev( widget, cbinfo->event );
    return Pt_CONTINUE;
}
```
int none( PtWidget_t *widget, void *data,
PtCallbackInfo_t *cbinfo )
{
    PtContainerNullFocus( PtFindDisjoint (widget),
        cbinfo->event );
    return Pt_CONTINUE;
}

int main()
{
    PtWidget_t *window, *texta, *textb, *textc,
        *button;
    PhPoint_t pos = { 0, 0 };
    PhArea_t area = { {0, 0}, {100, 20} };
    PhRect_t rect;
    int n;
    PtArg_t argt[5];
    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);
    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT,
        0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);
    n=0 ;
    PtSetArg( &argt[n], Pt_ARG_AREA, &area, 0 ); n++;
    PtSetArg( &argt[n], Pt_ARG_TEXT_STRING,
        "First Field", 0 ); n++;
    texta = PtCreateWidget( PtText, Pt_DEFAULT_PARENT,
        n, argt );
    PtExtentWidget( texta );
    PtWidgetExtent( texta, &rect );
    area.pos.y = rect.lr.y + 10;

    n = 0;
    PtSetArg( &argt[n], Pt_ARG_AREA, &area, 0 ); n++;
    PtSetArg( &argt[n], Pt_ARG_TEXT_STRING,
        "Second Field", 0 ); n++;
    textb = PtCreateWidget( PtText, Pt_DEFAULT_PARENT,
        n, argt );
    PtExtentWidget( textb );
    PtWidgetExtent( textb, &rect );
    area.pos.y = rect.lr.y + 10;

    n = 0;
    PtSetArg( &argt[n], Pt_ARG_AREA, &area, 0 ); n++;
    PtSetArg( &argt[n], Pt_ARG_TEXT_STRING,
        "Third Field", 0 ); n++;

PtContainerGiveFocus()

textc = PtCreateWidget( PtText, Pt_DEFAULT_PARENT, n, argt );
PtExtentWidget( textc );
PtWidgetExtent( textc, &rect );
pos.y += rect.lr.y + 15;

n = 0;
PtSetArg( &argt[n], Pt_ARG_POS, &pos, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_FLAGS, Pt_FALSE, Pt_GETS_FOCUS );
n++;
PtSetArg( &argt[n], Pt_ARG_TEXT_STRING, "First", 0 ); n++;
button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, n, argt );
PtAddCallback (button, Pt_CB_ACTIVATE, first, (void *)texta);
PtExtentWidget( button );
PtWidgetExtent( button, &rect );
pos.x += rect.lr.x + 10;

n = 0;
PtSetArg( &argt[n], Pt_ARG_POS, &pos, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_FLAGS, Pt_FALSE, Pt_GETS_FOCUS );
n++;
PtSetArg( &argt[n], Pt_ARG_TEXT_STRING, "Next", 0 ); n++;
button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, n, argt );
PtAddCallback (button, Pt_CB_ACTIVATE, next, NULL);
PtExtentWidget( button );
PtWidgetExtent( button, &rect );
pos.x += rect.lr.x + 10;

n = 0;
PtSetArg( &argt[n], Pt_ARG_POS, &pos, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_FLAGS, Pt_FALSE, Pt_GETS_FOCUS );
n++;
PtSetArg( &argt[n], Pt_ARG_TEXT_STRING, "Prev", 0 ); n++;
button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, n, argt );
PtAddCallback (button, Pt_CB_ACTIVATE, prev, NULL);
PtExtentWidget( button );
PtWidgetExtent( button, &rect );
pos.x += rect.lr.x + 10;

n = 0;
PtSetArg( &argt[n], Pt_ARG_POS, &pos, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_FLAGS, Pt_FALSE, Pt_GETS_FOCUS );
n++;
PtSetArg( &argt[n], Pt_ARG_TEXT_STRING, "First", 0 ); n++;
button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, n, argt );
PtAddCallback (button, Pt_CB_ACTIVATE, first, (void *)texta);
PtExtentWidget( button );
PtWidgetExtent( button, &rect );
pos.x += rect.lr.x + 10;

Chapter 13 • Pt—Widget Toolkit 1339
PtContainerGiveFocus()

    "None", 0 ); n++;
    button = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT,
        n, argt );
    PtAddCallback (button, Pt_CB_ACTIVATE, none, NULL);

    PtRealizeWidget( window );
    PtMainLoop();
    return EXIT_SUCCESS;
}

Classification:

    Photon

    Safety

    | Interrupt handler | No |
    |-------------------|----|
    | Signal handler    | No |
    | Thread            | No |

See also:

    PhEvent_t, PtContainerFocusNext(), PtContainerFocusPrev(),
    PtContainerNullFocus(), PtGiveFocus()

    PtWindowFocus() in the Photon Widget Reference
PtContainerHit()
Find the nth widget in an area

Synopsis:

PtWidget_t *PtContainerHit( PtWidget_t *container,
                           unsigned n,
                           PhRect_t const *rect );

Library:

ph

Description:

This function returns a pointer to the nth widget within the specified
container that intersects with the rectangle provided in the PhRect_t
structure pointed to by rect.

The coordinates of the rectangle are relative to the given container’s
canvas. If no widget intersects with rect, or if there are fewer than n
intersections, the function returns NULL.

Examples:

PtWidget_t *target_widget, *my_pane;

// In my_pane’s RAW callback:
my_raw_cb( PtWidget_t *container, void *data,
           PtCallbackInfo_t *cbinfo )
{
    PhRect_t *rect;
    PtWidget_t *container;
    PtWidget_t *target_widget;

    // ...
    rect = PhGetRects( cbinfo->event );
    container = PtFindContainer( widget );
    target_widget = PtContainerHit( container, 1, rect );
    if (target_widget)
        PtDestroyWidget( target_widget );

    // ...
}
PtContainerHit()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhGetRects(), PhRect_t, PtContainerBox()
Synopsis:

```c
int PtContainerHold( PtWidget_t *container_widget );
```

Library:

`ph`

Description:

This function increments the flux count for the given container, to prevent repairs to the specified container and to all its children. You can still modify the widgets, but the damage to them isn’t recorded.

You typically use this function when you’re about to make a lot of changes to the container and its children, and you don’t want to update the display until you’re done. When you want the container to be repaired, call `PtContainerRelease()`.

Returns:

The container widget’s current flux count, or -1 if an error occurred.

Examples:

See `PtClearWidget()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtContainerHold()

See also:

PtContainerRelease(), PtFlush(), PtIsFluxing()

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon Programmer’s Guide
PtContainerNullFocus()

Nullify the focus of a widget

Synopsis:

PtWidget_t *PtContainerNullFocus(
    PtWidget_t *widget,
    PhEvent_t *event);

Library:

ph

Description:

This function nullifies the focus of the specified widget’s parent. As a result, none of parent’s children has focus.

The event argument is a pointer to a PhEvent_t, structure that describes the event to be passed to the lost-focus callback of the parent widget and any of its children that were part of the focus chain at the time of this function call.

If event is NULL, this function generates a PhEvent_t structure filled with zeros for you.

Returns:

A pointer to the widget where the focus chain stops. On successful completion, this is widget’s parent.

Examples:

See PtContainerGiveFocus().

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
</tbody>
</table>

continued...
PtContainerNullFocus()

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PhEvent_t, PtContainerFocusNext(), PtContainerFocusPrev(), PtContainerGiveFocus()
**PtContainerRelease()**

*Decrement the flux count for a container, potentially damaging the container*

**Synopsis:**

```c
int PtContainerRelease( PtWidget_t *container_widget );
```

**Library:**

`ph`

**Description:**

This function decrements the flux count for the specified container widget. To increment the flux count (to delay updates to the display), call `PtContainerHold()`.

When the count reaches 0, `PtContainerRelease()` repairs the widgets by damaging the entire container.

**Returns:**

The container widget’s current flux count, or -1 if an error occurred.

**Examples:**

See `PtClearWidget()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtContainerRelease()

See also:

PtContainerHold(), PtFlush(), PtIsFluxing()

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon Programmer’s Guide
PtCRC()
Calculate a CRC for a block of data

Synopsis:

```c
long PtCRC( const char *buffer,
            int nbytes );
```

Library:

ph

Description:

This function generates a 32-bit cyclic redundancy check or CRC on the `nbytes` of data pointed to by `buffer`.

We recommend that bitmaps and images have a CRC on the image data and the palette. This CRC is used extensively by phrelay (see the QNX Neutrino Utilities Reference) to cache images.

You can call `PtCRCValue()` to calculate a running CRC checksum.

Returns:

The cyclic redundancy check.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtCRC()

See also:

\begin{itemize}
\item PtCRCValue(), PxLoadImage()
\item “Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
\end{itemize}
PtCRCValue()
Calculate a running CRC checksum

Synopsis:

```c
long PtCRCValue( long crc32val,
                unsigned char next_val );
```

Library:

ph

Description:

This function lets you maintain your own 32-bit cyclic redundancy check or CRC checksum. The `crc32val` is 0 or the value calculated by a previous call to `PtCRCValue()`, while `next_val` is the next byte of data for which to calculate the CRC.

We recommend that bitmaps and images have a CRC on the image data and the palette. This CRC is used extensively by `phrelay` (see the QNX Neutrino Utilities Reference) to cache images.

You can call `PtCRC()` to calculate a CRC for a block of data.

Returns:

The cyclic redundancy check.

Examples:

This is a slower implementation of `PtCRC()` for a 512-byte data segment:

```c
unsigned char data[512];
unsigned char *ptr;
long crcval = 0;
int i;

for (ptr = data, i = 0; i < sizeof( data );
     i++, ptr++) {
    crcval = PtCRCValue( crcval, *ptr );
}
```
PtCRCValue()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtCRC(), PxLoadImage()

“Images” in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**PtCreateActionSurface()**

Create a control surface within a widget, bound to a widget action

**Synopsis:**

```c
PtSurface_t *PtCreateActionSurface(
    PtWidget_t *widget,
    uchar_t surface_id,
    PtWidgetClassRef_t const *cref,
    ushort_t compound_action_id,
    ushort_t flags,
    ushort_t npoints,
    PhPoint_t *points,
    PtSurfaceDraw_f draw_f,
    PtSurfaceCalc_f calc_points_f );
```

**Library:**

`ph`

**Description:**

The function creates an action control surface within the given widget. The control surface is bound to a widget action.

The `surface_id` argument lets you specify the ID assigned to the surface. If you specify 0 for this argument, the next available surface ID for that widget is assigned to the surface. If you specify a surface ID that already exists within the widget, the function fails and returns 0.

The `cref` argument specifies the widget class that owns the action to which you're binding the surface. If you specify NULL for this argument, `PtCreateActionSurface()` assumes that the action belongs to the same class as `widget`.

The `compound_action_id` argument specifies the action to bind to the surface. Refer to the widget documentation for a list of actions that each widget supports.

The `flags` argument gives you additional control over how the surface behaves. The valid bits are:
**PtCreateActionSurface()**

**Pt** _SURFACE_RELEASE_POINTS_

The array of points specifying the widget’s geometry will be freed when the surface is destroyed.

**Pt** _SURFACE_HIDDEN_

Make the surface initially “hidden.” When surfaces are hidden, they don’t draw or respond to events. You should manipulate this flag only when creating the surface. To hide the surface after it’s created, use *PtHideSurface()* or *PtHideSurfaceByAction()*.

**Pt** _SURFACE_CONSUME_EVENTS_

The surface always consumes the events it receives. Setting this bit makes a surface opaque to events that it’s accepting, regardless of whether or not the surface actually uses the event. This functionality may prove useful if you wish to disable a portion of a widget.

**Pt** _SURFACE_DISABLED_

Prevent a surface from taking any action as a result of an event. If this flag is set, no events are consumed by the surface, unless the *Pt**_SURFACE_CONSUME_EVENTS* flag is set.

**Pt** _SURFACE_USE_ORIGIN_

The surface should use an adjustable origin that allows you to change the position of the surface without having to move the points that define the surface. This feature is useful if the surface is defined by a large number of points.

**Pt** _SURFACE_PARENT_RELATIVE_

By default, all events and drawing of a surface are relative to the upper-left corner of the widget’s canvas. Setting this flag adjusts this origin so that the surface uses the same coordinate system as its associated widget.

The *npoints* argument specifies the number of points that define the surface. Special values for this argument include:
PtCreateActionSurface()

- Pt_SURFACE_RECT for a rectangular surface
- Pt_SURFACE_ELLIPSE for an elliptical surface.

Otherwise the surface is polygonal with npoints vertexes.

The points argument points to an array of PhPoint_t structures that define the vertexes for the surface. For rectangular or elliptical surfaces, this needs to be only two points specifying the bounding box for the surface. For polygonal surfaces, the array must allow two points at the beginning of the array to store the bounding box of the surface, followed by npoints elements to specify the actual vertexes of the polygon. Additionally, if the Pt_SURFACE_USE_ORIGIN flag is set, the array must allow one element (located directly after the bounding box) to specify the origin. Points are stored in this fashion to optimize performance, memory requirements and simplicity of the most common cases.

The draw_f argument specifies a draw function for the surface, which must be of the following prototype:

```c
void draw_f( PtWidget_t *widget,
             PtSurface_t *surface,
             PhTile_t const *damage );
```

The damage argument points to a list of PhTile_t structures that specifies the areas of the control surface that were damaged.

If you don’t want the surface to draw anything, specify NULL for draw_f.

Similarly, the calc_points_f argument lets you specify a geometry-calculation function for the surface, which must be of the following prototype:

```c
void calc_points_f ( PtWidget_t *widget,
                     PtSurface_t *surface,
                     uchar_t post );
```

The post argument indicates when the function is being called:

- Zero Before the corresponding widget’s extent function.
- Nonzero After the widget’s extent function has completed.
PtCreateActionSurface()

If the widget’s extent depends on the geometry of a surface, you will want to perform the work if \( post \) is 0. If the surface’s geometry depends on the widget’s extent, you will want to perform the work if \( post \) is nonzero.

Returns:

A pointer to a PtSurface_t structure that describes the control surface, or NULL if the operation failed due to lack of memory, or incorrect parameters.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhPoint_t, PhTile_t, PtCreateSurface(), PtDestroyAllSurfaces(), PtDestroySurface()  
Control Surfaces chapter of the Photon Programmer’s Guide
Synopsis:

```c
PtWidgetClassStyle_t *PtCreateClassStyle(
    char *name );
```

Library:

`ph`

Description:

This function creates a new class style and calls the style `name`.

Returns:

A pointer to a new class style, or NULL if the wasn’t enough memory to create it.

Examples:

This example is based on the one in “Widget Styles” in the Managing Widgets in Application Code chapter of the Photon Programmer’s Guide:

```c
PtWidgetClassStyle_t *new_style =
    PtCreateClassStyle("blue");

PtSetStyleMember( new_style, Pt_STYLE_DRAW,
                  blue_draw);
PtAddClassStyle( PtButton, new_style);
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtCreateClassStyle() 

See also: 
PtAddClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetStyleMembers(), PtSetWidgetStyle() 

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference 

Synopsis:

```c
PtSurface_t PtCreateSurface(
    PtWidget_t *widget,
    uchar_t surface_id,
    ushort_t flags,
    ushort_t npoints,
    PhPoint_t *points,
    long event_mask,
    PtSurfaceCallback_f event_f,
    PtSurfaceDraw_f draw_f,
    PtSurfaceCalc_f calc_points_f);
```

Library:

```
ph
```

Description:

This function creates a regular control surface within the given widget.

The `surface_id` argument lets you specify the ID assigned to the surface. If you specify 0 for this argument, the next available surface ID for that widget is assigned to the surface. If you specify a surface ID that already exists within the widget, the function fails and returns 0.

The `flags` argument gives you additional control over how the surface behaves. The valid bits are:

- **Pt_SURFACE_RELEASE_POINTS**
  The array of points specifying the widget’s geometry will be freed when the surface is destroyed.

- **Pt_SURFACE_HIDDEN**
  Make the surface initially “hidden.” When surfaces are hidden, they don’t draw or respond to events. You should manipulate this flag only when creating the surface. To hide the surface after it’s created, use `PtHideSurface()` or `PtHideSurfaceByAction()`.
PtCreateSurface()

Pt_SURFACE_CONSUME_EVENTS

The surface always consumes the events it receives. Setting this bit makes a surface opaque to events that it’s accepting, regardless of whether or not the surface actually uses the event. This functionality may prove useful if you wish to disable a portion of a widget.

Pt_SURFACE_DISABLED

Prevent a surface from taking any action as a result of an event. If this flag is set, no events are consumed by the surface, unless the Pt_SURFACE_CONSUME_EVENTS flag is set.

Pt_SURFACE_USE_ORIGIN

The surface should use an adjustable origin that allows you to change the position of the surface without having to move the points that define the surface. This feature is useful if the surface is defined by a large number of points.

Pt_SURFACE_PARENT_RELATIVE

By default, all events and drawing of a surface are relative to the upper-left corner of the widget’s canvas. Setting this flag adjusts this origin so that the surface uses the same coordinate system as its associated widget.

The npoints argument specifies the number of points that define the surface. Special values for this argument include:

- Pt_SURFACE_RECT for a rectangular surface
- Pt_SURFACE_ELLIPSE for an elliptical surface.

Otherwise the surface is polygonal with npoints vertexes.

The points argument points to an array of PhPoint_t structures that define the vertexes for the surface. For rectangular or elliptical surfaces, this needs to be only two points specifying the bounding box for the surface. For polygonal surfaces, the array must allow two points at the beginning of the array to store the bounding box of the surface, followed by npoints elements to specify the actual vertexes of
the polygon. Additionally, if the Pt_SURFACE_USE_ORIGIN flag is set, the array must allow one element (located directly after the bounding box) to specify the origin. Points are stored in this fashion to optimize performance, memory requirements and simplicity of the most common cases.

The event_mask argument specifies the event types that the surface is sensitive to, and event_f specifies the function to call when an event is received that corresponds to one of the specified types. This function has the following prototype:

```c
int event_f( PtWidget_t *widget,
            PtSurface_t *surface,
            PhEvent_t const *event );
```

The expected return values for event_f are consistent with those of raw callbacks:

- Pt_CONSUME: Consume the event and prevent further propagation of it.
- Pt_CONTINUE: Allow the event to be passed up to the widget’s parent.

The draw_f argument specifies a draw function for the surface, which must be of the following prototype:

```c
void draw_f( PtWidget_t *widget,
             PtSurface_t *surface,
             PhTile_t const *damage );
```

The damage argument points to a list of PhTile_t structures that specifies the areas of the control surface that were damaged.

If you don’t want the surface to draw anything, specify NULL for draw_f.

Similarly, the calc_points_f argument lets you specify a geometry-calculation function for the surface, which must be of the following prototype:

```c
void calc_points_f ( PtWidget_t *widget,
                     PtSurface_t *surface,
                     uchar_t post );
```
PtCreateSurface()

This function’s post argument indicates when the function is being called:

Zero Before the corresponding widget’s extent function.
Nonzero After the widget’s extent function has completed.

If the widget’s extent depends on the geometry of a surface, you will want to perform the work if post is 0. If the surface’s geometry depends on the widget’s extent, you will want to perform the work if post is nonzero.

Returns:
A pointer to a PtSurface_t structure that describes the control surface, or NULL if the operation failed due to lack of memory, or incorrect parameters.

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:
PhEvent_t, PhPoint_t, PhTile_t, PtCreateActionSurface(), PtDestroyAllSurfaces(), PtDestroySurface()
Control Surfaces chapter of the Photon Programmer’s Guide
**PtCreateTransportCtrl()**

*Create a transport control structure for use with Drag and Drop*

**Synopsis:**

```c
PtTransportCtrl_t *PtCreateTransportCtrl();
```

**Library:**

`ph`

**Description:**

This function creates and initializes a control structure for the request and inline transport mechanism used for Drag and Drop. A widget that wants to act as the source in a drag-and-drop operation typically calls *PtCreateTransportCtrl() in its Pt_CB_ARM callback.*

**Returns:**

A pointer to the `PtTransportCtrl_t` structure, or NULL if it couldn’t be created.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtDndFetch_t, PtDndSelect(), PtInitDnd(), PtReleaseTransportCtrl(), PtTransportCtrl_t, PtTransportType()*

Drag and Drop chapter of the Photon *Programmer’s Guide*
**PtCreateWidget()**

Create a widget

**Synopsis:**

```c
PtWidget_t *PtCreateWidget(
    PtWidgetClassRef_t *class,
    PtWidget_t *parent,
    unsigned n_args,
    PtArg_t const *args);
```

**Library:**

`ph`

**Description:**

This function creates a widget in the current widget hierarchy. The `class` argument points to the desired widget class.

The `parent` argument specifies the parent widget. It can be a pointer to the parent widget or one of:

- `Pt_DEFAULT_PARENT` — use the default parent, which is the most recently created container.
- `Pt_NO_PARENT`

The `n_args` argument contains the number of arguments being passed to the widget library and the `args` argument points to an array containing `n_args` `PtArg_t` entries.

Since this function modifies and allocates only local data structures, it doesn’t result in any interaction with the Photon Manager. The user doesn’t see the widget until it’s realized.

Widgets that belong to the `PtContainer` class become the current parent widget when created. If you’re creating multiple `PtContainer`-class widgets, make sure each one is placed in the correct container. To do this, either specify the desired parent in `parent` or call `PtSetParentWidget()`.
Returns:

A pointer to the newly created widget, or null if an error occurs.

Examples:

```c
#include <stdlib.h>
#include <Pt.h>

int main ()
{
    PtWidget_t *window, *group1, *group2;
    PhPoint_t pos;
    PtArg_t argt[5];

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT,
            0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    /* Create a group as the child of the default parent (i.e.
            the window). */
    pos.x = pos.y = 0;
    PtSetArg( &argt[0], Pt_ARG_POS, &pos, 0 );
    group1 = PtCreateWidget( PtGroup, Pt_DEFAULT_PARENT, 1, argt );

    /* Create a group, specifying the window as the parent. */
    pos.x += 150;
    group2 = PtCreateWidget( PtGroup, window, 1, argt );

    /* Create some buttons as children of the default parent
            (i.e. the second group). */
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child2a", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child2b", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );

    /* Set the default parent to be group1 and create some
            children. */
    PtSetParentWidget( group1 );
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child1a", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child1b", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );

    /* Create a child, specifying the second group as the
            parent. */
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child2c", 0 );
```

Chapter 13 • Pt—Widget Toolkit 1365
PtCreateWidget()

    PtCreateWidget( PtButton, group2, 1, argt );
    
    PtRealizeWidget (window);
    PtMainLoop();
    return EXIT_SUCCESS;

Classification:

    Photon

    Safety
    
    Interrupt handler     No
    Signal handler        No
    Thread                No

See also:

    PtArg_t, PtDestroyWidget(), PtGetParentWidget(),
    PtReparentWidget(), PtSetArg(), PtSetParentWidget(),
    PtWidgetParent()

    “Creating widgets” in the Managing Widgets in Application Code
chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PtDamageExtent( PtWidget_t *widget,
                   PhRect_t const *extent );
```

Library:

`ph`

Description:

This function marks the specified `widget` as damaged and adds `extent` to the clipping list that will be used the next time the widget engine redraws this widget. The rectangle specified by the `PhRect_t` structure pointed to by `extent` is relative to the widget’s origin.

All widgets in front of the damaged widget that intersect with `extent` will be redrawn. If the damaged widget’s fill color is transparent, all widgets behind it that intersect `extent` will be redrawn. In all cases, the clipping will be set to `extent`.

The widget library takes care of updating widgets whenever resources are modified; you don’t normally need to use this function unless you’re using a `PtRaw` widget and want it to redraw and repair part or all of itself.

If you want the widget to be redrawn immediately, call `PtFlush()` after calling `PtDamageExtent()`.

Returns:

- 0 Success.
- -1 An error occurred. This function fails if the widget isn’t a container and doesn’t reside in a container, or if there isn’t enough memory to expand the damage list.

Examples:

See `PhBlit()`.
PtDamageExtent()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t, PtDamageWidget(), PtFlush()
**PtDamageSurface(), PtDamageSurfaceById()**

Mark a control surface as damaged so that it will be redrawn

**Synopsis:**

```c
void PtDamageSurface( PtWidget_t *widget,
                      PtSurface_t *surface );

void PtDamageSurfaceById( PtWidget_t *widget,
                           uchar_t surface_id );
```

**Library:**

ph

**Description:**

These functions flag a control surface to be redrawn. The `widget` argument specifies the widget owning the surface. The functions differ in how they identify the control surface:

*PtDamageSurface()*

Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

*PtDamageSurfaceById()*

Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtDamageSurface(), PtDamageSurfaceByld()

See also:

PtDamageSurfaceByAction()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtDamageSurfaceByAction()**

*Damage all surfaces that are associated with an action*

**Synopsis:**

```c
void PtDamageSurfaceByAction(  
    PtWidget_t *widget,  
    PtWidgetItem_t const *cref,  
    ushort_t action_id );
```

**Library:**

ph

**Description:**

`PtDamageSurfaceByAction()` flags to be redrawn all surfaces belonging to the given widget that are associated with an action. The `cref` and `action_id` specify the class and manifest of the action associated with the surfaces to be damaged.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtDamageSurface()`, `PtDamageSurfaceById()`

Control Surfaces chapter of the Photon *Programmer’s Guide*
PtDamageWidget()
Mark a widget as damaged so it will be redrawn

Synopsis:

```c
int PtDamageWidget( PtWidget_t *widget );
```

Library:

```
ph
```

Description:

This function adds the specified widget’s extent to the damage list of the widget’s first window parent. This effectively marks the widget as being damaged so that it will be redrawn.

The widget library takes care of updating widgets whenever resources are modified; you don’t normally need to use this function unless you’re using a PtRaw widget and want it to redraw and repair itself.

If you want the widget to be redrawn immediately, call PtFlush() after calling PtDamageWidget().

Returns:

- 0 Successful completion.
- -1 An error occurred.

Examples:

Set some global variables used in a PtRaw widget’s draw function and then damage the widget:

```c
grid_color = Pg_BLACK;
line_color1 = Pg_BLUE;
line_color2 = Pg_RED;
PtDamageWidget( my_raw_widget );
```

1372 Chapter 13 • Pt—Widget Toolkit
PtDamageWidget()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtDamageExtent(), PtFlush()
**PtDestroyAllSurfaces()**

*Destroy all of a widget’s control surfaces*

**Synopsis:**

```c
void PtDestroyAllSurfaces( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This function destroys all control surfaces of the given widget.

Generally this function is used only internally by the widget library, however you may call it if you wish.

---

All surfaces of a widget are automatically destroyed when the widget is destroyed.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtCreateActionSurface()`, `PtCreateSurface()`, `PtDestroySurface()`

Control Surfaces chapter of the Photon *Programmer’s Guide*


**PtDestroySurface()**

*Destroy a control surface*

**Synopsis:**

```c
void PtDestroySurface( PtWidget_t *widget,
                      PtSurface_t *surface );
```

**Library:**

`ph`

**Description:**

This function destroys the control surface specified by `surface` belonging to the given widget.

---

All surfaces of a widget are automatically destroyed when the widget is destroyed.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtCreateActionSurface()`, `PtCreateSurface()`, `PtDestroyAllSurfaces()`, `PtDestroySurfaceById()`, `Control Surfaces chapter of the Photon Programmer’s Guide`
**PtDestroySurfaceById()**

*Destroy the control surface with a given ID*

**Synopsis:**

```c
int PtDestroySurfaceById( PtWidget_t *widget,
                         uchar_t  surface_id );
```

**Library:**

ph

**Description:**

This function destroys the control surface with ID `surface_id` belonging to the given widget.

---

All surfaces of a widget are automatically destroyed when the widget is destroyed.

**Returns:**

- 0  Success.
- -1  The specified surface couldn’t be found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1376  Chapter 13 ● Pt—Widget Toolkit
PtDestroySurfaceById()

See also:

PtCreateActionSurface(), PtCreateSurface(), PtDestroyAllSurfaces(), PtDestroySurface()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtDestroyWidget()**

*Remove a widget from the widget family hierarchy*

**Synopsis:**

```c
int PtDestroyWidget( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This function performs the following on the specified widget:

- unrealizes it, if necessary
- destroys its children
- removes it from the widget family hierarchy
- flags it for destruction by adding it to the destroyed list.

The widget’s resources aren’t freed until the return of `PtEventHandler()`.

---

You might get callbacks from the widget after `PtDestroyWidget()` has returned. To determine if this is happening, check the widget’s `Pt_DESTROYED` flag. For example:

```c
if (PtWidgetFlags(widget) & Pt_DESTROYED)
{
    return( Pt_CONTINUE );
}
```

**Returns:**

- **0** Success.
- **-1** An error occurred.
PtDestroyWidget()
**PtDisableSurface(), PtDisableSurfaceById()**

*Disable a control surface*

**Synopsis:**

```c
void PtDisableSurface( PtWidget_t *widget,
                       PtSurface_t *surface,
                       ulong_t flags );

void PtDisableSurfaceById( PtWidget_t *widget,
                           uchar_t surface_id,
                           ulong_t flags );
```

**Library:**

`ph`

**Description:**

These functions disable a control surface belonging to the given widget.

*PtDisableSurface()*

Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

*PtDisableSurfaceById()*

Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

Disabled surfaces are drawn, but they don’t respond to events. They do or don’t consume events to which they’re sensitive, depending on the setting of their `Pt_SURFACE_CONSUME_EVENTS` flag. If this bit is set, the surface effectively blocks events to which it’s sensitive.

The `flags` argument specifies additional action to take, and may include the following values:

*Pt_DAMAGE_SURFACE*

Damage the surface if its state changes. This is useful if a surface draws differently depending on its enabled/disabled state.
PtDisableSurface(), PtDisableSurfaceById()

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtDisableSurfaceByAction(), PtEnableSurface(),
PtEnableSurfaceByAction(), PtEnableSurfaceById(),
PtSurfaceIsDisabled(), PtSurfaceIsEnabled()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtDisableSurfaceByAction()**

Disable all control surfaces associated with an action

**Synopsis:**

```c
void PtDisableSurfaceByAction(
    PtWidget_t *widget,
    PtWidgetClassRef_t const *cref,
    ushort_t action_id,
    ulong_t flags );
```

**Library:**

ph

**Description:**

This function disables all surfaces belonging to the given widget that are associated with an action. The `cref` and `action_id` specify the class and manifest of the action associated with surfaces to be disabled.

Disabled surfaces are drawn, but they don’t respond to events. They do or don’t consume events to which they’re sensitive, depending on the setting of their Pt_SURFACE_CONSUME_EVENTS flag. If this bit is set, the surface effectively blocks events to which it’s sensitive.

The `flags` argument specifies additional action to take, and may include the following values:

- **Pt_DAMAGE_SURFACE**
  Damage the surface if its state changes. This is useful if a surface draws differently depending on its enabled/disabled state.

**Classification:**

Photon

**Safety**

Interrupt handler  No

*continued...*
PtDisableSurfaceByAction()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtDisableSurface(), PtDisableSurfaceById(), PtEnableSurface(), PtEnableSurfaceByAction(), PtEnableSurfaceById(), PtSurfaceIsDisabled(), PtSurfaceIsEnabled()

Control Surfaces chapter of the Photon Programmer’s Guide
PtDndFetch_t
Structure that defines data types a widget accepts from a drag-and-drop event

Synopsis:

typedef struct ptdndfetch PtDndFetch_t;

struct ptdndfetch {
  char *type_name;
  char *description;
  int unsigned transport;
  int unsigned select_flags;
  void *user_data;
  PtSelectFunc_t *select_cb;
  PtRequestFunc_t *request_cb;
  TransportMalloc_t *transport_malloc;
  void *transport_malloc_cb_data;
};

Description:

The PtDndFetch_t structure is used to define the data types a widget accepts from a drag-and-drop event. It contains at least the following members:

- **type_name**: A string that must match the type string in the data being dropped.
- **description**: A regular expression that’s compared with the description of the data being dropped. A *description* of NULL is a “don’t care” and matches all descriptions.
- **transport**: Indicates the acceptable transport methods to get the data. A value of ~0 means that any method is acceptable. See `<PhTransport.h>` for the defined transport types.
- **select_flags**: Flags that control various aspects of data selection and interaction in a drag-and-drop operation:
  - Pt_DND_SELECT_DATA_DUP or Pt_DND_SELECT_DUP_DATA — the entry is a different representation of the previous fetch array element (e.g. a plain text version of HTML).
PtDndFetch_t

- Pt_DND_SELECT_MOTION — if this entry is selected on a drag-and-drop enter, drag-and-drop motion events are received.
- Pt_DND_SELECT_MULTIPLE — select all data that matches the criteria, not just the first match.

user_data  A convenient place to keep a reference. An index into the PtDndFetch_t array is provided to the Pt_CB_DND callback when invoked due to a drop, which makes finding user_data easy.

select_cb  If provided, this callback is invoked and its return value is used to determine if a piece of the data being dragged should be selected for drop acceptance or not. The return value of the function must be a transport type to select the data, or 0 to prevent that data from being unpacked on a drop event. See <PhTransport.h> for the defined transport types.

request_cb  An optional callback that’s called before asking the source for data. The parameters of the data request can be modified in the callback, or the request can be canceled altogether.

transport_malloc  An optional allocation function that’s called when unpacking drag-and-drop data. It’s useful for placing portions of unpacking data into shared memory areas and so on.

transport_malloc_cb_data  Data that’s passed to the transport_malloc function.

Classification:

Photon
PtDndFetch_t

See also:

PtCreateTransportCtrl(), PtDndSelect(), PtInitDnd(),
PtTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PtDndSelect(
    PtWidget_t *widget,
    PtDndFetch_t *data_array,
    int unsigned array_size,
    PhCursorDescription_t *accept_cursor,
    PhCursorDescription_t *reject_cursor,
    PtCallbackInfo_t *cbinfo );
```

Arguments:

- **widget**: A pointer to the widget involved in the drag-and-drop event.
- **data_array**: An array of data-selection criteria and optional processing functions. For more information, see `PtDndFetch_t`.
- **array_size**: The number of items in `data_array`.
- **accept_cursor**: The cursor that’s displayed if some data from the drag-and-drop event is selected. If this argument is NULL, the default accept cursor is used.
- **reject_cursor**: The cursor that’s displayed if no data from the drag-and-drop event is selected. If this argument is NULL, the default reject cursor is used.
- **cbinfo**: A pointer to the `PtCallbackInfo_t` structure (see the Photon Widget Reference) as received in a widget callback.

Library:

- **ph**
**PtDndSelect()**

**Description:**
This function selects the drag-and-drop data from the event found in the *cbinfo* that matches the selection criteria in *data_array.*

**Returns:**
The number of elements selected from the drag-and-drop event.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- *PtCreateTransportCtrl*, *PtDndFetch_t*, *PtInitDnd*
- *PtTransportType*
- *PtCallbackInfo_t* in the Photon *Widget Reference*
- Drag and Drop chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
PtWidgetClassStyle_t * PtDupClassStyle(
    PtWidgetClassRef_t * const ref,
    char const * const name,
    char const * const new_name);
```

**Library:**

`ph`

**Description:**

This function obtains a copy of the style called `name` in the widget class `ref`, and sets the copy’s name to `new_name`. You can modify this new style and/or add it as a new style to the widget class from which the style was duplicated.

**Returns:**

A pointer to the copy of the specified class style, or NULL if the specified style didn’t exist or there wasn’t enough memory to create the duplicate.

**Examples:**


**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
PtDupClassStyle()

See also:

PtAddClassStyle(), PtCreateClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetStyleMembers(), PtSetWidgetStyle(),

Pt_ARG_STYLE resource of ptBasic in the Photon Widget Reference

**PtEnableSurface(), PtEnableSurfaceById()**

*Enable a control surface*

**Synopsis:**

```c
void PtEnableSurface( PtWidget_t *widget,
                      PtSurface_t *surface,
                      ulong_t flags );

void PtEnableSurfaceById( PtWidget_t *widget,
                           uchar_t surface_id,
                           ulong_t flags );
```

**Library:**

ph

**Description:**

These functions enable a control surface, restoring it from a disabled state. The `widget` argument specifies the widget owning the surface. The functions differ in the way they identify the control surface:

*PtEnableSurface()*

Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

*PtEnableSurfaceById()*

Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

The `flags` argument specifies additional action to take, and may include the following values:

*Pt DAMAGE_SURFACE*

Damage the surface if its state changes. This is useful if a surface draws differently depending on its enabled/disabled state.
**PtEnableSurface(), PtEnableSurfaceById()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PtDisableSurface(), PtDisableSurfaceByAction(), PtDisableSurfaceById(), PtEnableSurfaceByAction(), PtSurfaceIsDisabled(), PtSurfaceIsEnabled()

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtEnableSurfaceByAction()**

Enable all control surfaces associated with an action

**Synopsis:**

```c
void PtEnableSurfaceByAction(
    PtWidget_t *widget,
    PtWidgetClassRef_t const *cref,
    ushort_t action_id,
    ulong_t flags);
```

**Library:**

`ph`

**Description:**

This function enables all surfaces associated with an action, restoring them from a disabled state.

The `widget` argument specifies the widget owning the surfaces, while `cref` and `action_id` specify the class and manifest of the action associated with surfaces to be enabled.

The `flags` argument specifies additional action to take, and may include the following values:

- **Pt_DAMAGE_SURFACE**
  
  Damage the surface if its state changes. This is useful if a surface draws differently depending on its enabled/disabled state.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 13 • Pt—Widget Toolkit  1393
PtEnableSurfaceByAction()

See also:

PtDisableSurface(), PtDisableSurfaceByAction(), PtDisableSurfaceById(), PtEnableSurface(), PtEnableSurfaceById(), PtSurfaceIsDisabled(), PtSurfaceIsEnabled()

Control Surfaces chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
int PtEndFlux( PtWidget_t *container );
```

**Library:**

ph

**Description:**

This function decrements the given container’s flux count. If the container’s flux count is 0 or becomes 0 due to the decrement, the container and its children can be repaired.

**Returns:**

The container widget’s current flux count, or -1 if an error occurred.

When the flux count goes to 0, you must manually call `PtDamageExtent()` and/or `PtDamageWidget()` to damage any widgets or areas that you want to be repaired.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtDamageExtent()`, `PtDamageWidget()`, `PtIsFluxing()`, `PtStartFlux()`

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon *Programmer’s Guide*
**PtEnter()**

*Lock the Photon library for use by a single thread*

**Synopsis:**

```c
int PtEnter( int flags );
```

**Arguments:**

The value of `flags` can be 0 or one of:

- `Pt_EVENT_PROCESS_ALLOW`
  
  Consider the calling thread an event reader.

- `Pt_EVENT_PROCESS_PREVENT`
  
  Consider the calling thread a nonreader.

In most cases, it’s better to set neither of these bits in `flags`, in which case the thread’s status as event reader or nonreader doesn’t change. For more information about changing a thread’s event reader status, see “Threads” in the Parallel Operations chapter of the Photon *Programmer’s Guide*.

You can OR the following into the `flags`:

- `Pt_DELAY_EXIT`
  
  Prevent another thread from terminating the process by calling `PtExit()`.

**Library:**

`ph`

**Description:**

This function gives the calling thread access to the Photon library by “locking the library.” If another thread has already locked the library, `PtEnter()` blocks until the library is unlocked.

Since Photon functions aren’t thread-safe, any thread other than the one that called `PtInit()` must call `PtEnter()` before trying to call any
other Photon functions. After you’re done, call PtLeave() to give other threads access to the library.

PtProcessEvent() unlocks the library before waiting for an event, and locks it back after. If you’re calling PtProcessEvent() or any other function that processes Photon events (like PtBkgdHandlerProcess() or PtModalBlock()), other threads may enter and leave the library while you’re waiting for an event. The same applies to PtCondWait() and PtCondTimedWait(), even though these functions don’t process Photon events.

Mixing threads and work procedures might cause a minor problem; if one of the other threads adds a workproc while another thread is already waiting for an event, the workproc might not be invoked until you receive an event.

As with other Pt functions, you have to make sure that PtInit() has been called (and succeeded) before you can call PtEnter().

The lock implemented by PtEnter() and PtLeave() isn’t recursive (in the way that mutexes can be recursive). If you call PtEnter() twice without calling PtLeave() in between, the second call to PtEnter() fails and returns the negative of EDEADLK.

If another thread has called PtExit() before you call PtEnter(), or calls PtExit() while PtEnter() is waiting for the lock to become available, the PtDELAY_EXIT flag ensures that PtExit() will return and allow the calling thread to keep running. Without this flag, PtEnter() does not return if another thread has called PtExit().

Returns:

0 Success, and the state of the thread didn’t change.

>0 Success, and the state of the thread changed. The return value can be a combination of:

- Pt_EVENT_PROCESS_ALLOW—The thread was a reader, and PtEnter() was called with flags set to Pt_EVENT_PROCESS_PREVENT.
PtEnter()

- Pt_EVENT_PROCESS_PREVENT—The thread was a nonreader, and PtEnter() was called with flags set to Pt_EVENT_PROCESS_ALLOW.
- Pt_DELAY_EXIT—For both the current call to PtEnter() and the previous call to PtLeave(), flags was set to Pt_DELAY_EXIT.

Some functions will reset the Pt_DELAY_EXIT flag bit for the thread to 0, which will affect the return value. These functions are PtCondWait(), PtCondTimedWait(), and any function that reads and processes an event (such as PtProcessEvent() or PtModalBlock()).

< 0  An error occurred; the value is a negative error code.

Errors:

If an error occurs, PtEnter() returns the negative of:

EDEADLK  The library is already locked by the calling thread.
ENOMEM  There wasn’t enough memory to satisfy the request.
EINVAL  The parameter flags is an invalid value.

Examples:

You can test whether you have the Photon Library locked by evaluating PtEnter(). If the Photon Library is already locked, PtEnter() will fail with an error of -EDEADLK. When PtLeave() is called with a negative value, it is guaranteed to fail, so in this example the Photon Library has the same lock state as it had before the call to PtEnter():

```c
int eval;
if ((eval = PtEnter(0)) < 0 && eval != -EDEADLK)
    fprintf(stderr, "Couldn’t enter: %s\n", strerror(-eval));
else
{
    PtSetResource(w, Pt_ARG_WINDOW_TITLE, text, 0);
    PtLeave(eval); // does nothing if eval == -EDEADLK
}
```
In this example, the Photon Library is locked elsewhere, and you want to unlock it to perform some lengthy operation, such as in a widget callback where not unlocking the library would “freeze” the GUI for the duration of the operation:

```c
int my_callback( PtWidget_t * widget, ApInfo_t * apinfo,
                 PtCallbackInfo_t * cbinfo )
{
    int flags;
    if ( ( flags = PtLeave( Pt_EVENT_PROCESS_PREVENT ) ) < 0 )
        fprintf( stderr, "Couldn’t leave: %s\n",
                strerror( -flags ) );
    else {
        do_some_lengthy_stuff();
        /* This will turn your thread back into a reader if it
           was a reader before: */
        if ( ( flags = PtEnter( flags ) ) < 0 )
            fprintf( stderr, "Couldn’t enter: %s\n",
                    strerror( -flags ) );
    }
    ... 
    return( Pt_CONTINUE );
}
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

`PtCondTimedWait()`, `PtCondWait()`, `PtExit()`, `PtInit()`, `PtLeave()`, `PtProcessEvent()`

`pthread_mutex_lock()` in the QNX Neutrino Library Reference
PtEnter()

“Threads” in the Parallel Operations chapter of the Photon
Programmer’s Guide
Synopsis:

```c
int PtEventHandler( PhEvent_t *event );
```

Library:

`ph`

Description:

This function determines which widgets were involved in the given event and invokes the appropriate callback functions. By doing this, the function enables widgets to interact with the user and to repair themselves when they've been exposed.

Returns:

The value returned by the last callback function invoked by the event, or -1 if an error occurred.

Examples:

See `PhEventNext()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhEvent_t, PhEventNext(), PhEventPeek(), PhEventRead(), PtSendEventToWidget()`
**PtExit()**

*Exit a Photon program*

**Synopsis:**

```c
void PtExit( int return_code );
```

**Library:**

```c
ph
```

**Description:**

This function is similar to `exit()`, except that `PtExit()` properly destroys all your application’s widgets before exiting.

In a multithreaded application, a callback can call `PtExit()` in one thread while a second thread might be in the middle of an important operation, such as writing a file. To prevent the second thread from terminating early, call `PtPreventExit()` before starting the operation, and call `PtAllowExit()` when it’s done.

Instead of calling `PtPreventExit()` and `PtAllowExit()` directly, you’re better off calling `PtEnter()` and `PtLeave()` with Pt_DELAY_EXIT set in the flags, because the Pt_DELAY_EXIT flag may not cause your application to lock up in some situations where `PtPreventExit()` would. For a discussion of the difference between these functions and using Pt_DELAY_EXIT, see “Exiting a multithreaded program” in the Parallel Operations chapter of the *Photon Programmer’s Guide*.

Before exiting, `PtExit()`:

1. Invokes all the callbacks in the `Pt_CB_APP_EXIT` list.
2. Destroys all the application’s widgets.
3. Enters the *delayed exit state*. In this state, the function waits for the counter associated with the `PtPreventExit()` function and the Pt_DELAY_EXIT flag to reach zero, if it’s not zero. If its value is nonzero, `PtExit()` unlocks the library using an equivalent of `PtLeave(0)` (notice that this will decrement the counter if the Pt_DELAY_EXIT flag is in effect for the calling thread) and waits until the counter reaches zero. If `PtExit()` is called again
PtExit()

by another thread, that second call also unlocks the library and blocks until the first call terminates the process.

**Returns:**

This function doesn’t return.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation point</td>
<td>No</td>
</tr>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

PtAllowExit(), PtEnter(), PtLeave(), PtPreventExit()

exit() in the QNX Neutrino Library Reference
**PtExtentWidget()**

*Force a widget to calculate its extent*

**Synopsis:**

```c
PtWidget_t *PtExtentWidget( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This function forces the specified widget to calculate its preferred size and apply its resize policy.

**Returns:**

A pointer to the widget.

**Examples:**

See *PtContainerGiveFocus()*.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtExtentWidgetFamily(), PtReRealizeWidget(), PtWidgetExtent()*

“Widget geometry” in the Introduction to the Photon *Programmer’s Guide*
**PtExtentWidgetFamily()**

*Force a widget and its children to calculate their extents*

**Synopsis:**

```c
int PtExtentWidgetFamily( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This function forces the specified widget and all its descendants to calculate their preferred sizes and apply their resize policies.

**Returns:**

- 0 Successful completion.
- -1 An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtExtentWidget(), PtReRealizeWidget(), PtWidgetExtent()`

“Widget geometry” in the Introduction to the Photon *Programmer’s Guide*
**PtFdProcF_t, PtFdProc_t**

*Type for defining a file-descriptor function*

**Synopsis:**

```c
typedef int PtFdProcF_t(int fd,
                         void *data,
                         unsigned mode);
```

```c
typedef PtFdProcF_t *PtFdProc_t;
```

**Description:**

These data types define pointers to file-descriptor functions. The `PtFdProcF_t` type is the function type that the `PtFdProc_t` type points to. This allows you to do something like this:

```c
PtFdProcF_t my_fd_proc;
int my_fd_proc(int fd, void *data, unsigned mode) {
    ...
}
```

The compiler should detect any inconsistencies between the two declarations of `my_fd_proc()` and give you an error (which is better than a “pointer mismatch” warning on the call to `PtAppAddFd()`).

**Classification:**

Photon

**See also:**

- `PtAppAddFd()`
- “Other I/O mechanisms” in the Interprocess Communication chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PtFepCmd( PtWidget_t *widget,
             long cmd,
             long sub_cmd,
             char const *data );
```

**Arguments:**

- **widget**
  A pointer to the widget on which you want any text-input boxes to appear.

- **cmd**
  The command that you want to send to the FEP; one of:
  
  - `Ph_FEP_CHANGE_MODE` — change to the mode specified by the string pointed to by `data`. The `data` string must be a null-terminated UTF-8 string that’s identical to the string that the FEP’s GUI displays in the mode-selection buttons.
  
  - `Ph_FEPHELP` — invoke the FEP’s help routine. This command doesn’t require any extra data, so you should pass NULL for the `data` argument.
  
  - `Ph_FEP_TOGGLE_1` — toggle the FEP’s mode. This is usually the same as pressing Alt-` in the FEP. This command doesn’t require any extra data, so you should pass NULL for the `data` argument.

- **sub_cmd**
  Not currently used; set this argument to 0.

- **data**
  A pointer to any extra data that the specific command needs.

**Library:**

```
ph
```
**PtFepCmd()**

**Description:**

`PtFepCmd()` lets you control a Photon front-end processor (FEP) from an application. This function lets you create your own front-end processor that includes all of the functionality of the GUI versions of the FEPs.

To use this function, start the FEP with the `-h` option to suppress the GUI interface. For more information, see the documentation for `cpim` and `vpim`.

**Returns:**

- 0  Success.
- -1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`cpim`, `vpim`
**PtFileSelection()**

Create a file-selector dialog

**Synopsis:**

```c
int PtFileSelection( PtWidget_t *parent,
                    PhPoint_t const *pos,
                    char const *title,
                    char const *root_dir,
                    char const *file_spec,
                    char const *btn1,
                    char const *btn2,
                    char const *format,
                    PtFileSelectionInfo_t *info,
                    int flags );
```

**Arguments:**

- **parent**  
The dialog’s parent, which can be NULL. It’s used with the *pos* argument to position the dialog.

- **pos**  
A pointer to a `PhPoint_t` structure that’s used with the *parent* to position the dialog.

  If *pos* is NULL, the function centers the dialog on the screen; if *parent* is NULL, the function places the dialog at the absolute coordinates of *pos*; otherwise it places the dialog at the relative offset of *pos* within *parent*.

- **title**  
The dialog’s title. If NULL, the function uses the string `File Selector`.

- **root_dir**  
The current directory for the file-selector widget. The default is `/` if this parameter is NULL.

  You can pass a directory or a full path for a file. *PtFileSelection()* parses the string and uses the longest existing path as the root directory. The function uses rest of the string as a suggested path to be displayed in the Name field.

- **file_spec**  
The file specification to look for. The default is `*` if this argument is NULL.
PtFileSelection()

**btn1** The string for button 1 (the default is Open). This is the button that returns the selected-file information. When activated, it sets info->ret to Pt_FSDIALOG_BTN1.

If you want to have a hotkey for this button, place an ampersand (&) in front of the appropriate character in the string. For example, to have the string Select with s as a hotkey, pass &Select as btn1.

**btn2** The string for button 2 (the default is Cancel). When activated, it sets info->ret to Pt_FSDIALOG_BTN2. If you want to have a hotkey for this button, place an ampersand (&) in front of the appropriate character in the string.

**format** A string to be used with the Pt_ARG_FS_FORMAT resource of the PtFileSel widget. It indicates what information to display and in what order. If you don’t want the divider shown, pass NULL for this argument; the function then displays only the filenames. See the description of PtFileSel for the format of this string.

**info** This is a mandatory parameter. The function returns the selected file in the path portion of this structure. For more information, see “PtFileSelectionInfo_t structure,” below.

**flags** Flags that affect the appearance and behavior of the dialog. The default is 0; you can OR together any of the following bits:

- **Pt_FSR_MULTIPLE**
  Let the user select more than one file or directory at a time.

- **Pt_FSR_NO_FCHECK**
  Permit nonexistent files/directories to be selected. (The path must exist.)

- **Pt_FSR_NO_FSPEC**
  Don’t display the file specification.
PtFileSelection()

Pt_FSR_NO_UP_BUTTON
    Don’t display the up-directory button.
Pt_FSR_NO_NEW
    Disable new directory creation via the Insert key.
Pt_FSR_NO_NEW_BUTTON
    Don’t display the new-directory button.
Pt_FSR_NO_SELECT_FILES
    Files aren’t selectable.
Pt_FSR_SELECT_DIRS
    Directories are selectable.
Pt_FSR_CREATE_PATH
    Create directories as needed when typed in manually.
Pt_FSR_NO_CONFIRM_CREATE_PATH
    Don’t confirm directory creation.
Pt_FSR_NO_DELETE
    Disable deletions via the Delete key.
Pt_FSR_NO_CONFIRM_DELETE
    Don’t confirm deletions.
Pt_FSR_RECURSIVE_DELETE
    Enable the recursive deletion of directories.
Pt_FSR_CONFIRMEXISTING
    Confirm the selection of an existing file with a message.
        The Pt_FSR_CONFIRMEXISTING flag is ignored when you’ve set the Pt_FSR_MULTIPLE flag, however you can supply your own confirm_selection function as described later.

You can also OR in the following bits, which affect the appearance and behavior of the PtFileSel widget in the dialog. Each of these bits corresponds to a flag in the
**PtFileSelection()**

*Pt_ARG_FS_FLAGS* resource of the *PtFileSel* widget:

- **Pt_FSR_DONT_SHOW_DIRS**
  - Don’t display directories.

- **Pt_FSR_DONT_SHOW_FILES**
  - Don’t display files.

- **Pt_FSR_SHOW_HIDDEN**
  - Show hidden directory entries (i.e. those whose names begin with a period).

- **Pt_FSR_SHOW_ERRORS**
  - Display (with a special icon) directory entries that had a read error.

- **Pt_FSR_FREE_ON_COLLAPSE**
  - Free items on every collapse. This means that every time a directory expands, its content is reread from the disk.

- **Pt_FSR_TREE**
  - Display the directory entries in a tree. By default, entries are displayed in a single level.

- **Pt_FSR_NO_SEEK_KEY**
  - Disable keyboard-seek in single-level mode. The default is to allow key-seeks by typing a single character.

- **Pt_FSR_NO_ROOT_DISPLAY**
  - Don’t display a root directory item in tree display mode. By default, when *Pt_FSR_TREE* is set, a root directory item is shown.

- **Pt_FSR_CASE_INSENSITIVE**
  - Make the file-specification pattern-matching insensitive to case.
PtFileSelection()

Pt_FSR_NO_ERROR_POPUP

Don’t pop up an error dialog when unable to open a directory.

Library:

ph

Description:

This function creates a file-selector dialog that lets the user browse files and directories. The dialog allows the selection of a file and/or directory and fills a PtFileSelectionInfo_t structure with information about the selected item and the dialog.
**PtFileSelection()**

An example of the dialog created by PtFileSelection().

Be sure to initialize the `PtFileSelectionInfo_t` structure pointed to by `info` before calling this function. This structure includes some pointers that must be set to NULL if you don’t want to provide callback functions. For more information, see “`PtFileSelectionInfo_t` structure,” below.

You can specify the dimensions of the dialog by setting the `info->dim` field before calling this function.
**PtFileSelection()**

This function can select directories as well as files. Enable directory selection with the Pt_FSR_SELECT_DIRS flag. Existing directories can be selected with *btn1* (the Open button).

*PtFileSelection()* can create and delete directories and delete files. You can create new directories at any time by pressing the New button. When the *PtFileSel* widget has focus, these hotkeys are activated:

- The *Insert* key creates a new directory, just like the New Directory button.
- The *Delete* key removes the currently selected item. If you’ve set Pt_FSR_MULTIPLE in the flags argument, the Delete key tries to delete all the selected items. A separate delete-confirmation/delete-error dialog is presented for each selected item, and the dialog has four buttons: Cancel, Delete, Skip, and Delete All.

*PtFileSelection()* has its own event-processing loop.

**PtFileSelectionInfo_t structure**

The *PtFileSelectionInfo_t* structure includes at least the following members:

- **short ret**
  The return code; either Pt_FSDIALOG_BTN1 or Pt_FSDIALOG_BTN2.

- **char path [PATH_MAX + NAME_MAX + 1]**
  The full path of the selected item. This member isn’t valid if you set Pt_FSR_MULTIPLE in the flags argument to *PtFileSelection()*.

**PtFileSelectorInfo_t *minfo**

If you set Pt_FSR_MULTIPLE in the flags argument to *PtFileSelection()* , this member points to a *PtFileSelectorInfo_t* structure (see below) in which the following members are valid:
PtFileSelection()

- nitems — the number of selected items
- multipath — an array of the full path for each selected item.

If you haven’t set Pt_FSR_MULTIPLE, minfo is NULL, and the selected item’s path is returned in the path member of PtFileSelectionInfo_t.

PhDim_t dim A PhDim_t structure that defines the dimensions of the dialog when the selection was completed. You can specify the size of the dialog by setting this field before calling PtFileSelection().

PhPoint_t pos The position of the dialog when the selection was completed.

cchar format[80] The format string of the dialog when the selection was completed.

cchar fspec[80] The file specification of the dialog when the selection was completed.

void *user_data User data to pass as the data argument to the confirm_display, confirm_selection, and new_directory functions.

int (*confirm_display)(PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo) A function to be called before an item is added to the file selector. If this member isn’t NULL, it must point to a function.

The members of the PtCallbackInfo_t structure are used as follows:

- reason — Pt_CB_FSR_DISPLAY
- cbdata — a pointer to a PtFileChooserInfo_t structure (see below).
PtFileSelection()

This function should return Pt_CONTINUE or Pt_END to indicate whether or not the item should be displayed in the file selector.

```
int (*confirm_selection) ( PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo)
```

A function that’s called when the user has made a final selection of a directory item by double-clicking on an item in the file selector, pressing Enter in the Name box, or clicking on the Open button. If this member isn’t NULL, it must point to a function.

The members of the PtCallbackInfo_t structure are used as follows:

- `reason` — Pt_CB_FSR_SELECTION
- `reason_subtype` — Pt_FSR_MULTIPLE if you’ve permitted multiple selections, 0 otherwise.
- `cbdata` — a pointer to a PtFileChooserInfo_t structure (see below).

If `reason_subtype` is Pt_FSR_MULTIPLE, the following members of the PtFileChooserInfo_t structure are valid:

- `nitems` — the number of items
- `items` — the array items; `items[n]->fullname` is the file specification of the nth selected item.

If `confirm_selection` returns Pt_CONTINUE, PtFileSelection() exits. If `confirm_selection` returns Pt_END, PtFileSelection() doesn’t exit, the selector stays on the screen, and the user must choose another file or directory.

Applications can use this function to screen selections and avoid having to call PtFileSelection() repeatedly.
*PtFileSelection()*

```c
int (*new_directory)( PtWidget_t *widget, void *data,
                      PtCallbackInfo_t *cbinfo)
```

A function that’s called whenever *PtFileSelection()* creates a new directory. If this member isn’t NULL, it must point to a function.

The members of the *PtCallbackInfo_t* structure are used as follows:

- **reason** — *Pt_CB_FSR_DIRECTORY*
- **reason_subtype** — one of:
  - *Pt_CB_FSR_DIR_AUTO*  
    The directory was created automatically.
  - *Pt_CB_FSR_DIR_MANUAL*  
    The directory was created by an explicit command.
- **cbdata** — a pointer to a
  - *PtFileSelectorInfo_t* structure (see below).

The function should return *Pt_CONTINUE*.

*PtArg_t* **args** An array of *PtArg_t* structures that specify resource settings for the dialog’s *PtFileSel* widget. For more information about the resources, see the Photon *Widget Reference*.

You can’t use this field to set the widget’s *Pt_ARG_SELECTION_MODE* resource. If you set *Pt_FSR_MULTIPLE* in the *flags* argument to *PtFileSelection()*, *Pt_ARG_SELECTION_MODE* is set to *Pt_EXTENDED_MODE*.

*PtFileSelection()* defines the following “pseudo resources” for *PtFileSel*:

- *Pt_ARG_FSRLBL_DEL_ALL* — the label of the Delete All button.
- *Pt_ARG_FSRLBL_SKIP* — the label of the Skip button.
When `PtFileSelection()` returns, you have to clean up the `PtFileSelectionInfo_t` structure because it can contain allocated members and strings. You can do the cleanup by calling:

```c
int PtFSFreeInfo( PtFileSelectionInfo_t *fs );
```

If you haven’t set `Pt_FSR_MULTIPLE`, you don’t have to call `PtFSFreeInfo()`.

### `PtFileSelectorInfo_t` structure

`PtFileSelection()` passes the `PtFileSelectorInfo_t` structure as a parameter to the `confirm_display`, `confirm_selection` and `new_directory` functions.

Some of the members of `PtFileSelectorInfo_t` are valid in the `confirm-selection` callback, and others are valid when `PtFileSelection()` returns.

The `PtFileSelectorInfo_t` structure contains at least these members:

- **char *path**
  The full path of the directory item. This member is valid only if you haven’t set `Pt_FSR_MULTIPLE` in the `flags` argument to `PtFileSelection()`.

- **struct stat *statbuf**
  A pointer to the same buffer as `Istatbuf` if `Istat()` succeeded and the file isn’t a symbolic link. If the file is a symbolic link according to `Istat()` (or, perhaps, if `Istat()` failed), `Stat()` is called, and `statbuf` points to its results — or is NULL if `stat()` fails.

- **struct stat *Istatbuf**
  A pointer to the `stat` structure returned by `Istat()`, or NULL if `Istat()` failed. - when
PtFileSelection()

Pt_FSR_MULTIPLE is set, all selected items will be returned via three new members in the existing PtFileSelectorInfo_t structure:

`int nitems`  The number of selected items if you've set Pt_FSR_MULTIPLE in the flags argument to PtFileSelection().

`char **multipath`  The full path of each selected item if you've set Pt_FSR_MULTIPLE.

`FileSelItem_t **items`  An array of the selected items if you've set Pt_FSR_MULTIPLE.

**Returns:**

0  Success.

-1  An error occurred.

**Examples:**

```c
/***********************
 * fsel.c
 *
 * Sample program that illustrates usage of
 * the PtFileSelection() convenience function.
 *
 * Compile as follows:
 * $ gcc -lph -o fsel fsel.c
 *
 * Run as follows:
 * $ ./fsel
 *
 ***********************/

#include <stdio.h>
#include <stdlib.h>
#include <Ph.h>
#include <Pt.h>

int main(int argc, char **argv)
```
PtFileSelection()

{
    PtFileSelectionInfo_t info;
    PtArg_t args[1];
    int k;

    /* Initialize the widget library and connect to Photon. */
    PtInit( NULL );

    /* Initialize the file-select info structure */
    memset( &info, 0x0, sizeof(PtFileSelectionInfo_t) );

    /* Change the name-column label of the PtFileSel widget in the filesel dialog from the default "Name" to "Nom" */
    PtSetArg( args, Pt_ARG_FS_LBL_NAME, "Nom:", 0 );
    info.args = args;
    info.num_args = 1;

    /* Invoke the convenience function. */
    k = PtFileSelection( NULL, /* parent */
        NULL, /* pos */
        "PtFileSelection Example", /* title */
        "˜", /* root_dir, tilde is the home directory specified by $HOME */
        NULL, /* file_spec filter */
        NULL, /* label of btn1, the Open button, default is "Open" */
        NULL, /* label of btn2, the Cancel button, default is "Cancel" */
        NULL, /* Pt ARG_FS_FORMAT resource of the PtFileSel widget, default is "nsd" */
        &info, /* PtFileSelectionInfo_t *info structure, must be specified */
        Pt_FSR_CONFIRM_EXISTING | 
        Pt_FSR_SHOW_HIDDEN | 
        Pt_FSR_NO_FCHECK /* PtFileSelection flags */
    );

    if ( k ) {
        fprintf( stderr, "\nPtFileSelection failed." );
        PtExit( -1 );
    }

    if ( info.ret == Pt_FSDIALOG_BTN1 )
        fprintf( stderr, "\nOpen button was pressed. The selected file is:\n\n%s\n\n", info.path );
    else
        fprintf( stderr, "\nCancel button was pressed.\n" );

    PtExit( 0 );
}
PtFileSelection()

        return EXIT_SUCCESS;
    }

Classification:

    Photon

    Safety
    Interrupt handler   No
    Signal handler      No
    Thread              No

See also:

    PhDim_t, PtArg_t
    PtFileSel in the Photon Widget Reference
    “Dialog modules” in the Working with Modules chapter of the Photon
    Programmer’s Guide
**PtFindChildClass()**

*Find the first descendant that matches the specified class*

**Synopsis:**

```c
PtWidget_t *PtFindChildClass(
    PtWidgetClassRef_t *class,
    PtWidget_t *widget);
```

**Library:**

`ph`

**Description:**

This function searches the widget family hierarchy of the given container, `widget`, for a descendant widget that matches the specified class.

Some container widgets, including `PtDivider`, `PtMenuBar`, `PtMultiText`, and `PtScrollArea` redirect children to an alternate parent. For all container widgets, it’s best to call `PtValidParent()` to determine the “real” parent of the children. For example, to find a `PtButton` in a `PtScrollArea`:

```c
child = PtFindChildClass( PtButton,
    PtValidParent( my_scrollarea, PtButton ));
```

**Returns:**

A pointer to a `PtWidget_t` structure, or `NULL` if an error occurs.

**Examples:**

```c
#include <stdlib.h>
#include <Pt.h>

int main()
{
    PtArg_t argt;
    PtWidget_t *window, *pane, *button;

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    // Create a window that contains a pane, which in turn
```
PtFindChildClass()

// contains a button.

if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL)) == NULL)
    PtExit(EXIT_FAILURE);

PtSetArg( &argt, Pt_ARG_RESIZE_FLAGS,
    Pt_TRUE, Pt_RESIZE_XY_ALWAYS);
pane = PtCreateWidget(PtPane, Pt_DEFAULT_PARENT, 1, &argt);

PtSetArg( &argt, Pt_ARG_TEXT_STRING, "Sample", 0);
PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, &argt );

// The following call finds the button because PtButton
// is a subclass of PtLabel.

button = PtFindChildClassMember( PtLabel, pane );
if (button != NULL) {
    printf ("The widget is a subclass of PtLabel.\n");
} else {
    printf ("The widget isn't a subclass of PtLabel.\n");
}

// The following call does not find the button because
// PtButton is not equivalent to the PtLabel class.

button = PtFindChildClass( PtLabel, pane );
if (button != NULL) {
    printf ("The widget is a PtLabel.\n");
} else {
    printf ("The widget isn't a PtLabel.\n");
}

// The following call finds the button because PtButton
// is in the class PtButton.

button = PtFindChildClass( PtButton, pane );
if (button != NULL) {
    printf ("The widget is a PtButton.\n");
} else {
    printf ("The widget isn't a PtButton.\n");
}

return EXIT_SUCCESS;
}
**PtFindChildClass()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtFindChildClassMember(), PtValidParent(), PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetFamily(), PtWidgetParent()*
**PtFindChildClassMember()**

*Find the first descendant that’s a subclass of the specified class*

**Synopsis:**

```c
PtWidget_t *PtFindChildClassMember(
    PtWidgetClassRef_t *class,
    PtWidget_t *widget);
```

**Library:**

`ph`

**Description:**

This function searches the widget family hierarchy of the given container, `widget`, for a descendant widget that’s a subclass of the specified `class`. For example, `PtMultiText`, `PtText`, `PtButton`, and `PtLabel` are all subclasses of `PtLabel`.

Some container widgets, including `PtDivider`, `PtMenuBar`, `PtMultiText`, and `PtScrollArea` redirect children to an alternate parent. For all container widgets, it’s best to call `PtValidParent()` to determine the “real” parent of the children. For example, to find a subclass of `PtLabel` in a `PtScrollArea`:

```c
child = PtFindChildClassMember( PtLabel,
    PtValidParent( my_scrollarea, PtLabel ));
```

**Returns:**

A pointer to a `PtWidget_t` structure, or NULL if an error occurs.

**Examples:**

See `PtFindChildClass()`.

**Classification:**

Photon
**PtFindChildClassMember()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetFamily(), PtWidgetParent()*
**PtFindClassStyle()**

**Find the style with a given name**

**Synopsis:**

```c
PtWidgetClassStyle_t *PtFindClassStyle(
    PtWidgetClassRef_t *ref,
    char *name);
```

**Library:**

ph

**Description:**

This function returns a pointer to the style called `name` from the widget class `ref`. If the name is NULL, `PtFindClassStyle()` returns a pointer to the default style for the provided class.

**Returns:**

A pointer to the style, or NULL if it wasn’t found.

**Examples:**

Return a pointer to the default style for `PtButton`:

```c
PtWidgetClassStyle_t *stylep;
stylep = PtFindClassStyle (PtButton, NULL);
```

Return a pointer to the `blue` style for `PtButton`:

```c
PtWidgetClassStyle_t *stylep;
stylep = PtFindClassStyle (PtButton, "blue");
```

**Classification:**

Photon
**PtFindClassStyle()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetStyleMembers(), PtSetWidgetStyle()*

*Pt_ARG_STYLE* resource of **PtBasic** in the Photon Widget Reference

**PtFindContainer()**

*Return the nearest container parent*

**Synopsis:**

PtWidget_t *PtFindContainer( PtWidget_t *widget );

**Library:**

ph

**Description:**

This function returns the nearest container parent (which could be
widget itself).

Some container widgets, including PtDivider, PtMenuBar,
PtMultiText, and PtScrollArea redirect children to an alternate
parent. For all container widgets, it’s best to call PtValidParent() to
determine the “real” parent of the children.

**Returns:**

A pointer to the nearest container parent of widget, or NULL if no
container parent was found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtFindData()
Find the first data block of a given type and subtype

Synopsis:

```c
void * PtFindData( PtDataHdr **ptr,
    long type,
    long subtype,
    long *len,
    PtDataHdr_t **node );
```

Library:

ph

Description:

This function finds the first data block that matches the type and subtype provided. If type is 0, any type will match. If subtype is -1, any subtype will match.

If _node is provided, it's set to point to the data_node that contained the returned data in order to be able to continue the search from that node. If len is provided, it's set to the length of the data item as set when the data was originally added to the chain.

Returns:

A pointer to the data, or NULL if no matching data was found.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtFindData()

See also:

PtAddData(), PtFindNextData(), PtRemoveData(), PtUnlinkData()
PtFindDisjoint()

Return the nearest disjoint parent widget

Synopsis:

\[
\text{PtWidget}_t \ast \text{PtFindDisjoint}( \text{PtWidget}_t \ast \text{widget} );
\]

Library:

\text{ph}

Description:

This function returns the nearest disjoint parent widget (which could be \text{widget} itself). A disjoint widget owns regions that aren’t children of its parent’s regions. Any clipping set by the parent of a disjoint widget isn’t applied to the disjoint widget. The regions of disjoint widgets are sensitive and opaque to expose events. A disjoint widget’s \text{class}_rec has the Pt_DISJOINT flag set.

Examples of widgets that are disjoint include:

- \text{PtWindow}
- \text{PtMenu}
- \text{PtRegion}

Examples of widgets that aren’t disjoint include:

- \text{PtButton}
- \text{PtBkgd}
- \text{PtRect}

Returns:

A widget pointer to the nearest disjoint parent of \text{widget}, or NULL if no disjoint container parent was found.

Classification:

Photon
## PtFindDisjoint()

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**Synopsis:**

```
PtWidget_t *PtFindFocusChild( PtWidget_t *widget );
```

**Library:**

ph

**Description:**

This function finds the closest focusable child widget of `widget`. If no focusable children are found, `widget` is returned.

This function doesn’t give focus to the child widget.

**Returns:**

The pointer passed in `widget` if no focusable children are found, or a pointer to the first focusable child of `widget`.

**Classification:**

Photon

- **Safety**
  - Interrupt handler: No
  - Signal handler: No
  - Thread: No

**See also:**

`PtContainerFindFocus()`, `PtFindFocusNextFrom()`, `PtFindFocusPrevFrom()`
**PtFindFocusNextFrom()**

*Find the next widget that can get focus*

**Synopsis:**

```c
PtWidget_t *PtFindFocusNextFrom(
    PtWidget_t *widget
);
```

**Library:**

`ph`

**Description:**

This function returns a pointer to the next widget that can get focus above the provided widget. If there are no such widgets above the provided widget, the search wraps around to the backmost widget that can get focus.

This function doesn’t give focus to the widget found.

**Returns:**

A pointer to the next widget that can get focus.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtFindFocusChild()`, `PtFindFocusPrevFrom()`
**Synopsis:**

```c
PtWidget_t *PtFindFocusPrevFrom(
    PtWidget_t *widget);
```

**Library:**

`ph`

**Description:**

This function returns a pointer to the previous widget that can get focus behind the provided widget. If there are no such widgets below the provided widget, the search wraps around to the frontmost widget that can get focus.

---

This function doesn’t give focus to the widget found.

**Returns:**

A pointer to the previous widget that can get focus.

**Classification:**

Photon

---

### Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtFindFocusChild()`, `PtFindFocusNextFrom()`
**PtFindGuardian()**

*Find the widget responsible for another widget’s actions*

**Synopsis:**

```c
PtWidget_t *PtFindGuardian( PtWidget_t *child,
                          int superior_only );
```

**Library:**

*ph*

**Description:**

This function returns the widget that’s responsible for the `child`’s actions. This is either the child’s natural parent or, if the child is `Pt_PROCREATED`, the widget that the child is a subordinate of (its superior widget).

If the `superior_only` value is nonzero, this function returns only a pointer to a superior widget as a guardian. If the child hasn’t been procreated, the function returns NULL. (Only procreated widgets have superiors.)

**Returns:**

A pointer to the child widget’s legal guardian, or NULL if the child widget has no guardian.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtFindGuardian()

See also:

PtGetParent(), PtValidParent(), PtWidgetParent()

**PtFindNextData()**

Find the next data block of a given type and subtype

**Synopsis:**

```c
void * PtFindNextData( PtDataHdr_t **ptr,
                      PtDataHdr_t *data,
                      long type,
                      long subtype,
                      long *len,
                      PtDataHdr_t **node);
```

**Library:**

ph

**Description:**

This function finds the next data block, from `data_node`, that matches the type and subtype provided. If `data_node` is NULL, the first instance of data that matches the type and subtype provided is found. If `type` is 0, any type will match. If `subtype` is -1, any subtype will match.

If `_node` is provided, it’s set to point to the `data_node` that contained the returned data in order to be able to continue the search from that node. If `len` is provided, it’s set to the length of the data item as set when the data was originally added to the chain.

**Returns:**

A pointer to the data, or NULL if no matching data was found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 13 • Pt—Widget Toolkit
See also:

PtAddData(), PtFindData(), PtRemoveData(), PtUnlinkData()
**PtFindSurface()**

*Find the control surface with a given ID*

**Synopsis:**

```c
PtSurface_t *PtFindSurface( PtWidget_t *widget, uchar_t surface_id );
```

**Library:**

ph

**Description:**

This function locates a control surface belonging to a given widget, using the surface’s numerical ID, `surface_id`, as a search key.

**Returns:**

A pointer to the structure representing the control surface, or `NULL` if the specified surface couldn’t be found.

---

Since control surfaces are maintained internally as an array, it’s not uncommon for them to shift around in memory as surfaces are added and removed, thereby possibly invalidating a pointer returned by this function. As such, all surface pointers should be regarded as transient, and you should retrieve an updated copy whenever there’s a chance that the widget’s surface configuration might have changed.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtFindSurface()

See also:

PtFindSurfaceByAction(), PtWidgetActiveSurface()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtFindSurfaceByAction()**

*Find the control surface associated with a given action*

**Synopsis:**

```c
PtSurface_t *PtFindSurfaceByAction(
    PtWidget_t *widget,
    PtWidgetClassRef_t const *cref,
    ushort_t action_id,
    PtSurface_t const *surface);
```

**Library:**

`ph`

**Description:**

This function locates a control surface using its associated action as a search key.

The `widget` argument specifies the widget to search, while `cref` and `action_id` specify the class and manifest of the action to seek. The `surface` argument specifies the surface at which to start the search, letting you find all the surfaces in a widget associated with a particular action. If this argument is NULL, the search begins at the first surface within the widget.

This function ignores regular surfaces (i.e. those created using `PtCreateSurface()` as opposed to `PtCreateActionSurface()`).

**Returns:**

A pointer to the structure representing the control surface, or NULL if no more surfaces associated with the specified action could be found.
PtFindSurfaceByAction()

Since control surfaces are maintained internally as an array, it’s not uncommon for them to shift around in memory as surfaces are added and removed, thereby possibly invalidating a pointer returned by this function. As such, all surface pointers should be regarded as transient, and you should retrieve an updated copy whenever there’s a chance that the widget’s surface configuration might have changed.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtFindSurface(), PtWidgetActiveSurface()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtFlush()**

*Immediately repair widget damage*

**Synopsis:**

```c
int PtFlush( void );
```

**Library:**

```c
ph
```

**Description:**

This function ensures that any widget damage is repaired, and then calls `PgFlush()` to flush buffered draw commands from your application to the graphics driver.

You’ll need to call this function explicitly if you’re drawing somewhere “outside” the standard Photon event loop (for example, in an input procedure) or if you want changes to the widgets to be made immediately.

*PtFlush() ignores and doesn’t affect the application’s hold count. For more information, see “Delaying and forcing updates to the display” in the Working with Code chapter of the Photon Programmer’s Guide.*

**Returns:**

The new value of the global hold count.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1446 Chapter 13 • Pt—Widget Toolkit
See also:

`PgFlush()`, `PtHold()`, `PtRelease()`

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon *Programmer’s Guide*
**PtFontSelection()**

Display a modal dialog for selecting a font

**Synopsis:**

```c
char *PtFontSelection( PtWidget_t *parent,
                       const PhPoint_t *pos,
                       const char *title,
                       const char *font,
                       long symbol,
                       unsigned flags,
                       const char *sample);
```

**Library:**

ph

**Description:**

This function displays a modal dialog that lets you select a text font.

![A font-selection dialog.](image)

The dialog is parented off the `parent` widget, which may be NULL; if
non-NULL, that parent widget is blocked and its cursor is changed to reflect this.
The dialog is positioned according to the pos parameter: if NULL, the
dialog is centered on the screen; if parent is NULL, the dialog is placed at the absolute coordinates of pos; otherwise it’s placed at the relative offset of pos within parent.

The title of the dialog is given by title; if this is NULL a default title of “Select Text Font” is used.

The initial font selected in the dialog is given by font. If this argument is NULL, the initial font is TextFont09. Otherwise, it should point to a font name created by PfGenerateFontName().

The symbol parameter specifies which character will be used to construct the list of available font families (refer to PfQueryFonts() in the Library Reference); if -1L is specified a default symbol ‘A’ (standard Latin fonts) is used.

The flags parameter is used to limit the inclusion of fonts based on certain characteristics:

<table>
<thead>
<tr>
<th>To include:</th>
<th>Use this flag:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalable fonts</td>
<td>PHFONT_SCALABLE</td>
</tr>
<tr>
<td>Bitmapped fonts</td>
<td>PHFONT_BITMAP</td>
</tr>
<tr>
<td>Proportional fonts</td>
<td>PHFONT_PROP</td>
</tr>
<tr>
<td>Fixed-width fonts</td>
<td>PHFONT_FIXED</td>
</tr>
</tbody>
</table>

You can OR the flags together to obtain a suitable mask. The default is PHFONT_ALL_FONTS (which is every option set).

The sample parameter sets the text string used to display a sample of the selected font; if NULL the default of AaBbCcXxYyZz is used.

The modal dialog is constructed and the user may select a text font using comboboxes for the family and size, and buttons for the style. A sample of text using the font is dynamically updated.
PtFontSelection()

The user may click a Cancel button or close the window to cancel the selection, or click an OK button to accept it.

Returns:

A pointer to the new font string if one selected, NULL otherwise. This string is obtained using the *strdup()* function, so the application should *free()* it once finished with the font name.

Examples:

```c
const char *fontname;
PhPoint_t pos= {10, 23};

fontname = PtFontSelection (widget, &pos,
   "Select a font!", "TextFont12", -1,
   PHFONT_ALL_FONTS,
   "Who is Sylvia? what is she, that all our\n   swains commend her?");
```

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PfGenerateFontName()* ,  *PhPoint_t*

PtFontSel in the Photon Widget Reference

“Dialog modules” in the Working with Modules chapter, and the Fonts chapter of the Photon Programmer’s Guide
**PtForwardWindowEvent()**

*Forward a window event to the window with a given region ID*

**Synopsis:**

```c
int PtForwardWindowEvent(
    PhWindowEvent_t const *window_event);
```

**Library:**

ph

**Description:**

This function forwards the provided window event to the window manager, which forwards it to the window whose region ID is specified in the event.

**Returns:**

0 Success.

-1 The message couldn’t be forwarded, possibly because either the Photon Manager or pwm wasn’t running.

**Examples:**

```c
int give_a_window_focus( PtWidget_t *widget )
{
    PhWindowEvent_t WE;
    if( !widget || !PtWidgetIsClassMember( widget, PtWindow ))
        return -1;
    memset( &WE, 0, sizeof (WE));
    WE.event_f = Ph_WM_FOCUS;
    WE.rid = PtWidgetRid( widget );
    WE.event_state = Ph_WM_EVSTATE_FOCUS;
    return PtForwardWindowEvent( &WE );
}
```

**Classification:**

Photon
**PtForwardWindowEvent()**

---

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhWindowEvent_t, PtForwardWindowTaskEvent()

PtWindowFocus(), PtWindowToBack(), PtWindowToFront() in the Photon Widget Reference

Window Management chapter of the Photon Programmer’s Guide.
**PtForwardWindowTaskEvent()**

Forward a window event to the task with a given Photon connection ID

**Synopsis:**

```c
int PtForwardWindowTaskEvent(
    PhConnectId_t task,
    PhWindowEvent_t const *evt);
```

**Library:**

`ph`

**Description:**

This function forwards a window event to the task with a given Photon connection ID. It’s similar to `PtForwardWindowEvent()`. You can get the Photon connection ID from the information that you get from `PhGetConnectInfo()`. `PtForwardWindowTaskEvent()` forwards the event to the main window for the task.

In addition to the window-event types described for the `PhWindowEvent_t` structure, you can specify a special pseudo-event, `Ph_WM_SUPERSELECT`. If you forward this type of event to a task, the window manager moves the window and any child windows to the current console, puts them in front of any other windows, and gives focus to the first nonblocked window.

**Returns:**

- 0 Success.
- -1 An error occurred.

**Examples:**

See “Getting and setting the window state” in the Window Management chapter of the Photon *Programmer’s Guide.*
**PtForwardWindowTaskEvent()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PhGetConnectId()*, *PhGetConnectInfo()*, *PhWindowEvent_t*, *PtForwardWindowEvent()*, *PtWindowFocus()*, *PtWindowToBack()*, *PtWindowToFront()* in the Photon Widget Reference

“Getting and setting the window state” in the Window Management chapter of the Photon Programmer’s Guide.
**PtGetAbsPosition()**

Get the absolute position of a widget

**Synopsis:**

```c
void PtGetAbsPosition( PtWidget_t *widget,
                      short *x,
                      short *y );
```

**Library:**

ph

**Description:**

This function gets the absolute position of a widget (i.e. the coordinates of the upper left corner of the widget’s border). The coordinates are returned in \( x \) and \( y \).

For a window, the \( x \) and \( y \) coordinates don’t include the frame that’s added by the window manager. Windows don’t usually have borders.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtWidgetOffset()*
PtGetControlFlags()
Get the flags from the _Pt_ control structure

Synopsis:

```c
int PtGetControlFlags( void );
```

Library:

`ph`

Description:

This function returns the flags of the _Pt_ control structure. The valid _Pt_ flag bits are:

- **Pt_FEP_PRESENT**
  - An FEP is present.
- **Pt_FEP_QUERIED**
  - A search for any existing FEPs has been done.
- **Pt_IN_EXPOSE**
  - The widget library is currently processing an expose event.

Returns:

The control flags.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtGetDndFetchIndex()**

Search for an entry in the data_array for an incoming drag-and-drop event.

**Synopsis:**

```c
int PtGetDndFetchIndex(
    const PtDndCallbackInfo_t * cbinfo,
    const PtDndFetch_t * data_array,
    unsigned array_size);
```

**Arguments:**

- `dndcb` A pointer to the drag-and-drop callback data.
- `data_array` A pointer to an array of transport types that the widget accepts in drag-and-drop events. For more information see `PtDndFetch_t`.
- `array_size` The number of items in `data_array`.

**Library:**

`ph`

**Description:**

This function determines if the drag-and-drop data from the event found in the `PtDndCallbackInfo_t` matches any of the transport types in the `data_array`.

**Returns:**

- The index in the data_array for drag-and-drop event
  - A match was found.
  - `-1` No entry is found.

**Examples:**

In this example, we determine the index in the data_array for the incoming drag-and-drop event:

```c
PtDndCallbackInfo_t *dndcb = cbinfo->cbdata;
static PtDndFetch_t FetchTypes[] = {
    {"PhTransfiles", NULL, Ph_TRANSPORT_INLINE, },
```
/*
 * ARRAY_SIZE is defined as follows.
 */
#define ARRAY_SIZE ( m_array ) ( sizeof( m_array ) / sizeof( m_array[0] ) )

int dnd_callback( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
    switch( cbinfo->reason_subtype )
    {
        case Ph_EV_DND_ENTER:
            num_matches = PtDndSelect( widget, FetchTypes, ARRAY_SIZE( FetchTypes ) )
            break;
        case Ph_EV_DND_DROP:
            switch( PtGetDndFetchIndex( dndcb, FetchTypes, ARRAY_SIZE( FetchTypes ) ) )
            {
                case 0: //file
                    
                    
                    
                    break;
            }
            return( Pt_CONTINUE );
    }
}

Classification:

Photon

Safety

Interrupt handler No
Signal handler No
Thread No

See also:

PtDndFetch_t, PtDndSelect(), PtInitDnd(),
PtReleaseTransportCtrl() PtTransportCtrl_t, PtTransportType()
PtGetDndFetchIndex()

Drag and Drop chapter of the Photon Programmer’s Guide
\textbf{PtGetParent()}

Find the nearest parent widget that matches the specified class

\textbf{Synopsis:}

\begin{verbatim}
PtWidget_t *PtGetParent(
    PtWidget_t *widget,
    PtWidgetClassRef_t *parent_class )
\end{verbatim}

\textbf{Library:}

\textbf{ph}

\textbf{Description:}

This function examines the specified widget's hierarchy, and tries to find the nearest widget in the hierarchy (including the specified widget itself) that matches the specified parent class.

\textbf{Returns:}

A pointer to the matching widget, or NULL if no parent was found.

\textbf{Examples:}

\begin{verbatim}
PtWidget_t *window;

// Get main window widget and make it a parent for drawing.
window = PtGetParent( widget, PtWindow );
PtSetParentWidget( window );
PgSetRegion( PtWidgetRid(window) );
\end{verbatim}

\textbf{Classification:}

Photon

\begin{tabular}{l l}
\textbf{Safety} & \\
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}
See also:

*PtSetParentWidget(), PtWidgetParent()*

**PtGetParentWidget()**

*Return the current default widget parent*

**Synopsis:**

```c
PtWidget_t *PtGetParentWidget( void );
```

**Library:**

`ph`

**Description:**

`PtGetParentWidget()` returns the current default widget parent.

**Returns:**

A pointer to the current default parent.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtCreateWidget()`, `PtGetParent()`, `PtReparentWidget()`, `PtSetParentWidget()`, `PtWidgetParent()`,

"Ordering widgets" in the Managing Widgets in Application Code chapter of the Photon *Programmer’s Guide*
PtGetRcvidPid()
Get the PID from a RCVID

Synopsis:

```c
pid_t PtGetRcvidPid( int rcvid );
```

Library:

ph

Description:

This function should be used to obtain the process ID (PID) from the receive ID (RCVID). It might be needed if a nonspecific input procedure attaches a specific input procedure.

Returns:

The PID, or -1 on error.

Examples:

```c
int general_input_proc( void *data, pid_t rcvid,
                      void *msg, size_t size)
{
    PtAppAddInput( NULL, PtGetRcvidPid(rcvid),
                   function, data );
#if defined(__QNXNTO__)
    MsgReplyv( rcvid, ... );
#else
    Reply( rcvid, ... );
#endif
    return Pt_END;
}
```

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
</tbody>
</table>

continued...
PtGetRcvidPid()

Safety

Thread No

See also:

PtAppAddInput(), PtAppAddInputRemote(), PtGetRcvidPidNd()
**PtGetRcvidPidNd()**

Get the PID and ND from a RCVID

**Synopsis:**

```c
pid_t PtGetRcvidPid( int rcvid,
                    int *nd );
```

**Arguments:**

- `rcvid` The receive ID you want to get the PID and node descriptor from
- `nd` The address where the function stores the node descriptor associated with the `rcvid`. You can pass NULL, in which case the function will fail with EREMOTE if the `rcvid` is from a remote process.

**Library:**

`ph`

**Description:**

This function should be used to obtain the process ID (PID) and node descriptor (ND) from the receive ID (RCVID). It might be needed if a nonspecific input procedure attaches a specific input procedure.

**Returns:**

The PID on success or:

- `EREMOTE` The `rcvid` is from a remote process, but you passed `NULL` for `nd`.
- `-1` an error occurred.

**Classification:**

Photon
PtGetRcvidPidNd()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAppAddInput()
**PtGetResource()**

Retrieve one resource value for a widget

Synopsis:

```c
#define PtGetResource( widget, type, value, len ) ...
```

Arguments:

- **widget**: A pointer to the widget whose resource you want to set.
- **type**: The resource manifest (e.g. `Pt_ARG_COLOR`).
- **value**: The address of a pointer to the appropriate data type (see the “New resources” tables in the Photon Widget Reference).
- **len**: Depends on the resource type.

Library:

```
ph
```

Description:

This macro sets a pointer to a resource value within the specified widget.

**PtGetResource()** doesn’t support the nonpointer method of getting resources. For information on getting and setting resources, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.
PtGetResource()

CAUTION: Because PtGetResource() returns a pointer directly into the internals of the widget, don’t modify the resource value directly. If you wish to retrieve the value of a given resource and then modify that value:

1. Get the resource.
2. Copy the resource to a temporary variable.
3. Modify the temporary variable.
4. Using the modified copy, set the resource.

Returns:

0   Success.

-1  An error occurred.

Examples:

Determine whether or not a widget is highlighted:

```
unsigned long *flags;
PtWidget_t *widget;

PtGetResource( widget, Pt_ARG_FLAGS, &flags, 0 );
printf( "Highlighted: %s\n",
    (*flags & Pt_HIGHLIGHTED) ? "Yes":"No" );
```

Classification:

Photon

Safety

Interrupt handler  No

continued...
**PtGetResource()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtArg_t, Pt_ARG(), PtGetResources(), PtSetArg(), PtSetResource(), PtSetResources()

**PtGetResources()**

Retrieve one or more resource values for a widget

**Synopsis:**

```c
int PtGetResources( PtWidget_t *widget,
                   int n_args,
                   PtArg_t *args );
```

**Library:**

ph

**Description:**

This function sets pointers to resource values within the specified widget. The `args` array indicates which resources are desired, and `n_args` indicates the number of items in the `args` array.

If you’re getting only one resource, it’s easier to call `PtGetResource()`.

You must initialize the `args` array with `PtSetArg()` or `Pt_ARG()` before calling `PtGetResources()`. The Pt type of a resource determines how that resource should be set or queried. You use the Pt type when setting a resource entry with `PtSetArg()`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.
CAUTION: Because PtGetResources() returns pointers directly into the internals of the widget, don’t modify those values directly. If you wish to retrieve the value of a given resource and then modify that value:

1. Get the resource.
2. Copy the resource to a temporary variable.
3. Modify the temporary variable.
4. Using the modified copy, set the resource.

Returns:

0  Success.
-1  An error occurred.

Examples:

Determine a widget’s color, and determine whether or not the widget is highlighted:

```c
PtArg_t     args[2];
PgColor_t   *color;
unsigned long  *flags;
PtWidget_t  *widget;

PtSetArg( &args[0], Pt_ARG_FILL_COLOR, &color, 0 );
PtSetArg( &args[1], Pt_ARG_FLAGS, &flags, 0 );
PtGetResources( widget, 2, args );

printf( "Color: %08lx Highlighted: %s\n", *color,
    *flags & Pt_HIGHLIGHTED ? "Yes":"No" );
```
**PtGetResources()**

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtArg_t, Pt_ARG(), PtGetResource(), PtSetArg(), PtSetResource(), PtSetResources()`

Synopsis:

```c
void *PtGetStyleMember(
    PtWidgetClassStyle_t *style,
    int manifest);
```

Library:

```
ph
```

Description:

This function retrieves the value of the given `style` associated with the given `manifest`.

For a description of the possible manifests, see `PtSetStyleMember()`.  

Returns:

The provided manifest determines the type of the pointer returned:

**Pt STYLE DRAW**

A pointer to the style’s draw function:

```c
void (*draw_f)( PtWidget_t *widget,
    PhTile_t const *damage );
```

**Pt STYLE EXTENT or Pt STYLE SIZING**

A pointer to the style’s extent function:

```c
void (*sizing_f)( PtWidget_t *widget );
```

**Pt STYLE ACTIVATE**

A pointer to the style’s activate function:

```c
void (*activate_f)( PtWidget_t *widget,
    PtWidgetClassStyle_t *old_style );
```
This isn’t the same as the widget’s `Pt_CB_ACTIVATE` callback.

**Pt_STYLE_CALC_BORDER**

A pointer to the function that calculates the style’s borders:

```c
void (*calc_border_f)( PtWidget_t const *widget,
                      PhRect_t *border );
```

The style’s borders are the distances from the widget’s extent (i.e. the widget’s outermost borders) to the widget’s canvas (i.e. the content area of the widget). For example, if the style’s `border.ul.x` is 5, there are five pixels between the widget’s left extent and the left side of the widget’s canvas. This space is typically used to render outlines, bevels, and so on.

**Pt_STYLE_CALC_OPAQUE**

A pointer to the function that calculates the style’s opacity:

```c
void (*calc_opaque_f)( PtWidget_t *widget );
```

**Pt_STYLE_DEACTIVATE**

A pointer to the style’s deactivation function:

```c
void (*deactivate_f)( PtWidget_t *widget,
                      PtWidgetClassStyle_t *new_style );
```

**Pt_STYLE_NAME**

A pointer to a string.

**Pt_STYLE_DATA**

A pointer to a block of data associated with the style.

`PtGetStyleMember()` returns NULL if the member specified isn’t set or the manifest is invalid.
PtGetStyleMember()

Examples:

```c
void display_style_name( PtWidgetClassStyle_t *style )
{
    char *name = PtGetStyleMember( style, Pt_STYLE_NAME );

    printf( "Style: %s\n", name ? name : "Default" );
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetStyleMembers(), PtSetWidgetStyle()

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**PtGetWidgetStyle()**

Get the style that a widget is currently using

**Synopsis:**

```c
PtWidgetClassStyle_t *PtGetWidgetStyle(
    PtWidget_t *widget);
```

**Library:**

`ph`

**Description:**

This function returns a pointer to the style that the provided widget is currently using.

**Returns:**

A pointer to the style.

**Examples:**

```c
void display_style( PtWidget_t *widget )
{
    PtWidgetClassStyle_t *style =
        PtGetWidgetStyle( widget );

    display_style_name ( style );
}
```

For details about `display_style_name()`, see `PtGetStyleMember()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetClassStyleMember(), PtSetClassStyleMethods(), PtSetClassStyleMember(), PtSetClassStyleMembers(), PtSetWidgetStyle()

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**PtGiveFocus()**

*Give focus to a widget*

**Synopsis:**

```c
PtWidget_t *PtGiveFocus( PtWidget_t *widget, PhEvent_t *event );
```

**Library:**

```
ph
```

**Description:**

This function gives focus to the specified widget, even if the widget's Pt_GETS_FOCUS flag isn’t set. `PtGiveFocus()` is the same as `PtContainerGiveFocus()`.

If the widget is a `PtWindow`, use `PtWindowFocus()` instead of this function — see the Photon Widget Reference.

The `event` argument is a pointer to a `PhEvent_t` structure that describes the event that will be passed to the lost-focus callback of the widget losing focus and to the got-focus callback of the widget getting focus. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

A pointer to the newly focused widget. This is usually the same as the `widget` argument, but it could be NULL if one of the following is true:

- The `widget` argument is NULL.
- The given widget is disjoint (e.g. a window).
- The widget is blocked; that is, it has Pt_BLOCKED set in its `Pt_ARG_FLAGS` resource (see `PtWidget` in the Photon Widget Reference).
- The widget has been destroyed before the attempt to give it focus.
- The event passed (if not NULL) has already caused focus to change, as indicated by:
PtGiveFocus()

\[ event->\text{processing\_flags} \& \text{Ph\_DIRECTED\_FOCUS} \]

This function could also return a pointer to a different widget if that widget for some reason refused to relinquish focus (i.e. its \text{Pt\_CB\_LOST\_FOCUS} callback returned Pt\_END — see \text{PtBasic} in the Photon Widget Reference). This usually happens if the requirements of an entry field haven’t been met and must be met before any other action can be taken.

The widget library never refuses to relinquish focus. If a widget does this, it’s because of a \text{Pt\_CB\_LOST\_FOCUS} callback in your application.

Examples:

See \text{PtContainerGiveFocus()}.

Classification:

Photon

\begin{center}
\begin{tabular}{l|c}
\textbf{Safety} & \\
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}
\end{center}

See also:

\text{PhEvent\_t, PtContainerFocusNext(), PtContainerFocusPrev(), PtContainerGiveFocus(), PtContainerNullFocus(), PtWindowFocus()} in the Photon Widget Reference
PtGlobalFocusNext()
Give focus to next widget

Synopsis:

```
PtWidget_t *PtGlobalFocusNext( PtWidget_t *widget,
                             PhEvent_t *event );
```

Library:

```
ph
```

Description:

This function finds the currently focused widget in the same disjoint widget (window, region, menu) as `widget` and moves the focus to the next focusable widget.

The widget that’s given focus receives the event described in the given `PhEvent_t` structure as the reason. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

Returns:

The widget that was given focus, or NULL if no focusable widgets are found.

Classification:

```
Photon
```

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

```
PhEvent_t, PtContainerGiveFocus(), PtContainerNullFocus(),
PtContainerFocusNext(), PtContainerFocusPrev(),
```

1480  Chapter 13  •  Pt—Widget Toolkit
PtGlobalFocusNext()
**PtGlobalFocusNextContainer()**

*Give focus to another container’s widget*

**Synopsis:**

```c
PtWidget_t *PtGlobalFocusNextContainer(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

This function finds the currently focused widget in the same disjoint widget (window, region, menu) as `widget` and moves the focus to the next focusable widget in a different container.

The widget that’s given focus receives the given event as the reason. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

The widget that was given focus, or NULL if no focusable widgets are found.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
PtGlobalFocusNextContainer()

See also:

PhEvent_t, PtContainerGiveFocus(), PtContainerNullFocus(),
PtContainerFocusNext(), PtContainerFocusPrev(),
PtGlobalFocusNext(), PtGlobalFocusNextFrom(),
PtGlobalFocusPrev(), PtGlobalFocusPrevFrom(),
PtGlobalFocusPrevContainer(), PtIsFocused()
**PtGlobalFocusNextFrom()**

*Give focus to the next widget behind the specified widget*

**Synopsis:**

```c
PtWidget_t *PtGlobalFocusNextFrom(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

Unlike `PtGlobalFocusNext()`, this function doesn’t find the currently focused widget first. `PtGlobalFocusNextFrom()` moves the focus to the next focusable widget after `widget` in the same disjoint widget (window, region, menu) as `widget`.

The widget that’s given focus receives `event` as the reason. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

The widget that was given focus, or NULL if no focusable widgets are found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

1484 Chapter 13 • Pt—Widget Toolkit
PtGlobalFocusNextFrom()

See also:

PhEvent_t, PtContainerGiveFocus(), PtContainerNullFocus(), PtContainerFocusNext(), PtContainerFocusPrev(), PtGlobalFocusNext(), PtGlobalFocusNextContainer(), PtGlobalFocusPrev(), PtGlobalFocusPrevFrom(), PtGlobalFocusPrevContainer(), PtIsFocused()
**PtGlobalFocusPrev()**

*Give focus to the previous widget*

**Synopsis:**

```c
PtWidget_t *PtGlobalFocusPrev( PtWidget_t *widget,
                              PhEvent_t *event );
```

**Library:**

`ph`

**Description:**

This function finds the currently focused widget in the same disjoint widget (window, region, menu) as `widget` and moves the focus to the previous focusable widget.

The widget that’s given focus receives `event` as the reason. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

The widget that was given focus, or NULL if no focusable widgets are found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhEvent_t`, `PtContainerGiveFocus()`, `PtContainerNullFocus()`, `PtContainerFocusNext()`, `PtContainerFocusPrev()`
**PtGlobalFocusPrev()**

`PtGlobalFocusNext()`, `PtGlobalFocusNextContainer()`,
`PtGlobalFocusNextFrom()`, `PtGlobalFocusPrevFrom()`,
`PtGlobalFocusPrevContainer()`, `PtIsFocused()`
**Synopsis:**

```c
PtWidget_t *PtGlobalFocusPrevContainer(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

This function finds the currently focused widget in the same disjoint widget (window, region, menu) as `widget` and moves the focus to the previous focusable widget in another container.

The widget that’s given focus receives the given event as the reason. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

The widget that was given focus, or NULL if no focusable widgets are found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtGlobalFocusPrevContainer()

See also:

PhEvent_t, PtContainerGiveFocus(), PtContainerNullFocus(),
PtContainerFocusNext(), PtContainerFocusPrev(),
PtGlobalFocusNext(), PtGlobalFocusNextContainer(),
PtGlobalFocusNextFrom(), PtGlobalFocusPrev(),
PtGlobalFocusPrevFrom(), PtIsFocused()
**PtGlobalFocusPrevFrom()**

Give focus to widget previous to the specified widget

**Synopsis:**

```c
PtWidget_t *PtGlobalFocusPrevFrom(
    PtWidget_t *widget,
    PhEvent_t *event);
```

**Library:**

`ph`

**Description:**

Unlike `PtGlobalFocusPrev()`, this function doesn’t find the currently focused widget first. `PtGlobalFocusNextFrom()` will move the focus to the first focusable widget previous to `widget` in the same disjoint widget (window, region, menu) as `widget`.

The widget that’s given focus receives the event described in the given `PhEvent_t` structure as the reason. If `event` is NULL, this function generates a `PhEvent_t` structure filled with zeros for you.

**Returns:**

The widget that was given focus, or NULL if no focusable widgets are found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PhEvent_t, PtContainerGiveFocus(), PtContainerNullFocus(),
PtContainerFocusNext(), PtContainerFocusPrev(),
PtGlobalFocusNext(), PtGlobalFocusNextContainer(),
PtGlobalFocusNextFrom(), PtGlobalFocusPrev(),
PtGlobalFocusPrevContainer(), PtIsFocused()
**PtHelpQuit()**  
*Exit the Helpviewer*

**Synopsis:**

```c
void PtHelpQuit( void );
```

**Library:**

`ph`

**Description:**

`PtHelpQuit()` tells the Helpviewer to exit.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtHelpTopic()`, `PtHelpTopicRoot()`, `PtHelpTopicTree()`,  
`PtHelpSearch()`, `PtHelpUrl()`, `PtHelpUrlRoot()`  
Context-Sensitive Help chapter of the Photon *Programmer’s Guide*  
`helpviewer` in the QNX Neutrino *Utilities Reference*.
Synopsis:

```c
int PtHelpSearch( char *string,
                  int mode,
                  int scope,
                  int method);
```

Arguments:

- **string**: The string you’re looking for.
- **mode**: One of the following:
  - HELP_SEARCH_MODE_TITLE—search for `string` in the titles of all the help topics in the given scope.
  - HELP_SEARCH_MODE_TEXT—search the text of all the help topics in the current scope.
  - HELP_SEARCH_MODE_DISPLAYED—search the text of the displayed topic only.
- **scope**: One of the following:
  - HELP_SEARCH_SCOPE_ALL—search for the text in all the online help information.
  - HELP_SEARCH_SCOPE_SELECTED—search for text in the selected topic only (for example, in a single book or bookset).
- **method**: One of the following:
  - HELP_SEARCH_METHOD_EXACT—search for an exact match (excluding case). For example, if you’re searching topic titles for `Help`, match a title `Help`, but not `Help files`.
  - HELP_SEARCH_METHOD_WORD—search for the string as a distinct word or words, ignoring case. For example, if you’re searching for `Help`, match `Help` and `help files` but not `Helpviewer`.

---

Chapter 13 • Pt—Widget Toolkit  1493
PtHelpSearch()

- HELP_SEARCH_METHOD_SUBSTRING—search for the string as distinct words or as substrings, ignoring case. For example, if you’re searching for Help, match Help, help files, and Helpviewer.
- HELP_SEARCH_METHOD_SUBSTRING_CASE—search for the string as distinct words or as substrings, paying attention to the case. For example, if you’re searching for Help, match Help and Helpviewer, but not help files.

Library:

ph

Description:

Use PtHelpSearch() to search for a string in the online help information. PtHelpSearch() spawns the Helpviewer if it isn’t running, or sends a message to the Helpviewer if it is.

Returns:

0 on success, or -1 if the Helpviewer couldn’t be found or spawned.

\[\text{PtHelpSearch()}\text{ returns immediately, before the search is complete.}\]

Examples:

\[
\text{PtHelpSearch( "console", HELP_SEARCH_MODE_TITLE, }
\text{ HELP_SEARCH_SCOPE_ALL,}
\text{ HELP_SEARCH_METHOD_SUBSTRING );}
\]

Classification:

Photon
PtHelpSearch()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtHelpQuit(), PtHelpTopic(), PtHelpTopicRoot(), PtHelpTopicTree(), PtHelpUrl(), PtHelpUrlRoot()

Context-Sensitive Help chapter of the Photon Programmer’s Guide

helpviewer in the QNX Neutrino Utilities Reference.
**PtHelpTopic()**

*Display help information identified by a topic path*

**Synopsis:**

```c
int PtHelpTopic( char *topic );
```

**Library:**

`ph`

**Description:**

*PtHelpTopic()* tells the Helpviewer to display the help text located by the given topic path.

*PtHelpTopic()* spawns the Helpviewer if it isn’t running, or sends a message to the Helpviewer if it is. Using this function, a Photon application can respond to a help request from the user by telling the Helpviewer to display the relevant help information.

If the topic path is relative (i.e. it doesn’t start with a `/`), it’s appended to the root topic path specified in an earlier call to *PtHelpTopicRoot()*.

**Returns:**

0 on success, or -1 if the Helpviewer couldn’t be found or spawned.

---

*PtHelpTopic()* returns immediately, before the help topic has been displayed.

**Examples:**

The following example shows a fragment from a hypothetical application built using PhAB. This application first sets up a topic root for all help requests; it then displays help for a particular part of the application in response to a callback (which could be attached to a hotkey, for example).

```c
#include <Pt.h>

int main( void )
{
    :
```
int HelpCallback( PtWidget_t * widget, ApInfo_t * apinfo,
                 PtCallbackInfo_t * cbinfo )
{
    :
    if( widget == ABW_Concepts )
        PtHelpTopic( "Widget concepts" );
    :
    return( Pt_CONTINUE );
}

Classification:
Photon

Safety
Interrupt handler   No
Signal handler      No
Thread              No

See also:
PtHelpQuit(), PtHelpSearch(), PtHelpTopicRoot(),
PtHelpTopicTree(), PtHelpUrl(), PtHelpUrlRoot()
Context-Sensitive Help chapter of the Photon Programmer’s Guide
helpviewer in the QNX Neutrino Utilities Reference.
PtHelpTopicRoot()
Specify the root of help topic paths

Synopsis:

```c
void PtHelpTopicRoot( char *topic );
```

Library:

ph

Description:

PtHelpTopicRoot() lets you specify a partial root topic path that’s prefixed to any relative topic paths in subsequent PtHelpTopic() calls. (Any path that doesn’t start with a / is considered relative.)

PtHelpTopicRoot() doesn’t copy the topic root. Don’t free the string until you’ve finished using the root topic path.

Examples:

See PtHelpTopic().

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtHelpQuit(), PtHelpSearch(), PtHelpTopic(), PtHelpTopicTree(), PtHelpUrl(), PtHelpUrlRoot()

Context-Sensitive Help chapter of the Photon Programmer’s Guide
helpviewer in the QNX Neutrino Utilities Reference.
PtHelpTopicTree()  
Load a new help topic tree

Synopsis:

```c
int PtHelpTopicTree( char *file );
```

Library:

```
ph
```

Description:

`PtHelpTopicTree()` tells the Helpviewer to load a new topic tree. The argument must be a top-level topic file with the `.toc` extension. The format of the topic file is defined in “Creating topic files” in the Helpviewer documentation in the QNX Neutrino Utilities Reference.

`PtHelpTopicTree()` spawns the Helpviewer if it isn’t running, or sends a message to the Helpviewer if it is.

Returns:

0 on success, or -1 if the Helpviewer couldn’t be found or spawned.

---

`PtHelpTopicTree()` returns immediately, before the topic tree has been displayed.

Examples:

```c
PtHelpTopicTree( root/path/to/product/photon.toc );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

*PtHelpQuit(), PtHelpSearch(), PtHelpTopic(), PtHelpTopicRoot(), PtHelpUrl(), PtHelpUrlRoot()

Context-Sensitive Help chapter of the Photon *Programmer’s Guide* [helpviewer](#) in the QNX Neutrino *Utilities Reference*.
**PtHelpUrl()**  
Display help information identified by a URL

**Synopsis:**

```c
int PtHelpUrl( char *url );
```

**Library:**

ph

**Description:**

`PtHelpUrl()` tells the Helpviewer to display the help text located by the given Universal Resource Locator. If the URL is relative (i.e. it doesn’t start with a `/`), it’s appended to any partial URL root specified in a previous call to `PtHelpUrlRoot()`.

`PtHelpUrl()` spawns the Helpviewer if it isn’t running, or sends a message to the Helpviewer if it is. Using this function, a Photon application can respond to a help request from the user by telling the Helpviewer to display the relevant help information.

**Returns:**

0 on success, or -1 if the Helpviewer couldn’t be found or spawned.

---

*PtHelpUrl() returns immediately, before the help topic has been displayed.*

**Examples:**

The following example shows a fragment from a hypothetical application built using PhAB. This application first sets up a URL root for all help requests; it then displays help for a particular part of the application in response to a callback (which could be attached to a hotkey, for example).

```c
#include <Pt.h>

int main( void )
{
  
  :
```
PtHelpUrl() function:

```c
PtHelpUrlRoot(
    "*/usr/help/product/photon/prog_guide/intro.html" );
    
int HelpCallback( PtWidget_t * widget, ApInfo_t * apinfo,
    PtCallbackInfo_t * cbinfo )
{
    
    if( widget == ABW_Libraries )
        PtHelpUrl( "#PhotonLibraries" );
    
    return( Pt_CONTINUE );
}
```

Classification:
Photon

Safety
- Interrupt handler: No
- Signal handler: No
- Thread: No

See also:
- PtHelpQuit(), PtHelpSearch(), PtHelpTopic(), PtHelpTopicRoot(), PtHelpTopicTree(), PtHelpUrlRoot()

**PtHelpUrlRoot()**

Specify a partial URL for help information

**Synopsis:**

```c
void PtHelpUrlRoot( char *url );
```

**Library:**

ph

**Description:**

`PtHelpUrlRoot()` lets you specify a partial root URL that’s prefixed to any relative URLs in subsequent `PtHelpUrl()` calls. (A relative URL is one that doesn’t start with a `/`.)

---

`PtHelpUrlRoot()` doesn’t copy the given URL. Don’t free the string until you’ve finished using the root URL.

**Examples:**

See `PtHelpUrl()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtHelpQuit()`, `PtHelpSearch()`, `PtHelpTopic()`, `PtHelpTopicRoot()`, `PtHelpTopicTree()`, `PtHelpUrl()`

Context-Sensitive Help chapter of the Photon *Programmer’s Guide*
helpviewer in the QNX Neutrino Utilities Reference.
Hide a control surface

**Synopsis:**

```c
int PtHideSurface( PtWidget_t *widget,
                  PtSurface_t *surface );

int PtHideSurfaceById( PtWidget_t *widget,
                       uchar_t surface_id );
```

**Library:**

```
ph
```

**Description:**

These functions hide a control surface belonging to the given widget. They differ in how they identify the control surface:

- **PtHideSurface()**
  - Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

- **PtHideSurfaceById()**
  - Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

Hidden surfaces don’t draw and aren’t included in event processing.

**Returns:**

- 0  Success.
- -1  The specified surface couldn’t be found or was already hidden.

**Classification:**

Photon
**PtHideSurface(), PtHideSurfaceById()**

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

- `PtHideSurfaceByAction()`, `PtShowSurface()`, `PtShowSurfaceByAction()`, `PtShowSurfaceById()`, `PtSurfaceIsHidden()`, `PtSurfaceIsShown()`

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtHideSurfaceByAction()**

*Hide all control surfaces associated with an action*

**Synopsis:**

```c
int PtHideSurfaceByAction(
    PtWidget_t *widget,
    PtWidgetItemClassRef_t const *cref,
    ushort_t action_id);
```

**Library:**

`ph`

**Description:**

This function hides all control surfaces associated with an action. The `widget` argument specifies the widget owning the surfaces, while `cref` and `action_id` specify the class and manifest of the action associated with surfaces to be hidden.

**Returns:**

0 if any surfaces were hidden, or -1 if no surfaces were affected.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtHideSurface(), PtHideSurfaceById(), PtShowSurface(), PtShowSurfaceById(), PtSurfaceIsHidden(), PtSurfaceIsShown()`

Control Surfaces chapter of the Photon *Programmer’s Guide*
PtHit()
Identify a widget in the specified container

Synopsis:

```c
PtWidget_t *PtHit( PtWidget_t *container,
                   unsigned n,
                   PhRect_t const *rect );
```

Library:

ph

Description:

This function returns container’s nth child widget whose extent intersects with the rectangle defined in the PhRect_t pointed to by rect. The rectangle’s coordinates must be relative to container’s canvas. PtHit() ignores unrealized or procreated widgets.

Returns:

A pointer to the container’s nth widget who’s extent intersects with rect, or NULL if there’s no such child widget.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t
**PtHold()**

Increment the hold count to prevent the visible repair of all widgets

**Synopsis:**

```c
int PtHold( void );
```

**Library:**

```c
ph
```

**Description:**

This function prevents visible repair of all widgets until your application makes a corresponding call to `PtRelease()`.

The application’s hold count is incremented when you call `PtHold()`, and decremented when you call `PtRelease()`. When the hold count reaches 0, the widgets are repaired.

A hold count of 0 simply means that the application doesn’t want to prevent widgets from repairing themselves. The widgets manage their own damage repair and may not take immediate action even if the count is 0.

**Returns:**

The new value of the hold count.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1510  Chapter 13  ●  Pt—Widget Toolkit
PtHold()

See also:

*PtFlush(), PtRelease()*

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon *Programmer’s Guide*
PtInflateBalloon()
Create a balloon widget

Synopsis:

PtWidget_t *PtInflateBalloon( PtWidget_t *win,
                               PtWidget_t *me,
                               int position,
                               char const *string,
                               char const *font,
                               PgColor_t fill,
                               PgColor_t text_color );

Library:

ph

Description:

This function creates a label widget as a child of *win*. The widget is placed according to *position*, relative to *me*. The *string* is rendered inside the label widget using a text color of *text_color*, and the font specified in *font* (which you should create by calling *PfGenerateFontName()*). The widget itself is filled with the color specified in *fill*.

Valid values for *position* include:

- Pt_BALLOON_RIGHT
- Pt_BALLOON_LEFT
- Pt_BALLOON_TOP
- Pt_BALLOON_BOTTOM
- Pt_BALLOON_INPLACE

Returns:

A pointer to the newly created balloon widget, or NULL if *win* isn’t a disjoint widget (e.g. *PtWindow*, *PtRegion*, etc.) or *me* is NULL.
**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PfGenerateFontName*, *PgColor_t*
**PtInit()**

*Initialize the widget library*

**Synopsis:**

\[
\text{int PtInit( char const *name );}
\]

**Library:**

`ph`

**Description:**

This function initializes the widget library. If no Photon channel is currently attached, this function calls `PhAttach()` with the given Photon server name (`/net/darrin/dev/photon`, for example). If there is a current channel, `PhAttach()` isn’t called.

Once a channel is attached, `PtInit()` installs all the widgets supplied by QNX Software Systems.

**Returns:**

- `0` Successful completion.
- `-1` An error occurred.

**Examples:**

See `PtClearWidget()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1514  Chapter 13  •  Pt—Widget Toolkit
See also:

PhAttach()
**PtInitDnd()**

*Initiate a drag-and-drop operation*

**Synopsis:**

```c
int PtInitDnd( PtTransportCtrl_t *ctrl,
              PtWidget_t *widget,
              PhEvent_t *event,
              PhDndCursors_t *cursors,
              int unsigned flags );
```

**Arguments:**

- `ctrl` A pointer to a `PtTransportCtrl_t` structure which contains the data that the function uses to initiate a drag-and-drop operation.

- `widget` A pointer to a `PtWidget_t` structure which specifies the widget on whose behalf the drag-and-drop operation is being started. It’s this widget that receives all drag-and-drop progress-notification events and callbacks.

- `event` A pointer to a `PhEvent_t` structure that describes the event that was used to precipitate the operation.

- `cursors` A pointer to a `PhDndCursors_t` structure which, if provided, is used instead of the default drag-and-drop cursors (accept, reject, unknown... app not responding) if the destination doesn’t override them.

  `PhDndCursors_t` contains three members, all pointers to `PhCursorDescription_t`, which describe the cursor’s appearance when performing a drag-and-drop operation:

  ```c
typedef struct Ph_event_dndrop_cursors {
    const PhCursorDescription_t *active, *inactive, *unknown;
  } PhDndCursors_t;
```

- `active` — NULL or a cursor to display over a region that has declared that it wants the drop (and hasn’t provided a different cursor).

- `inactive` — NULL or a cursor to display over a region that has declared that it doesn’t want the drop (and hasn’t provided a different cursor).
**PtInitDnd()**

- *unknown* — NULL or a cursor to display over a region while we’re still waiting for a response from it.

**flags**

A combination of the following bits:

- `Pt_DND_SILENT` Don’t notify the initiator of drag-and-drop progress (valid only if no requestable data types were added to the `PtTransportCtrl_t` structure).
- `Pt_DND_LOCAL` Restrict the drop so that it can occur only within the context of the application that initiated the drag-and-drop. That is to say that the user can’t drop the data on any other application. This is very useful for allowing the dragging and dropping of private data or pointer references that are meaningful only within a single application’s context.

**Library:**

ph

**Description:**

This function initiates a drag-and-drop operation with the data previously added to the `PtTransportCtrl_t` structure pointed to by `ctrl`. Before calling `PtInitDnd()`, your application must create this structure by calling `PtCreateTransportCtrl()` and populate it via calls to `PtTransportType()` and `PtTransportRequestable()`.

The structure pointed to by `ctrl` is automatically released at the end of the drag-and-drop operation.
PtInitDnd()

Returns:

0  Success.
-1  An error occurred.

Examples:

```c
int cb_outbound (PtWidget_t *widget, void *data,
                   PtCallbackInfo_t *cbinfo);
{
    PtTransportCtrl_t *tctrl = PtCreateTransportCtrl();
    PtTransportType( tctrl, "text", "simple sentence",
                   0, Ph_TRANSPORT_INLINE, "string",
                   "This is my inlined data.", 0, 0 );
    PtInitDnd( tctrl, widget, cbinfo->event, NULL,
               Pt_DND_LOCAL | Pt_DND_SILENT );
    return Pt_CONTINUE;
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

- `PhCursorDescription_t`
- `PhEvent_t`
- `PtCancelDnd()`
- `PtCreateTransportCtrl()`
- `PtDndFetch_t`
- `PtDndSelect()`
- `PtTransportCtrl_t`
- `PtTransportType()`

Drag and Drop chapter of the Photon Programmer's Guide
**PtInputCallbackProcF_t, PtInputCallbackProc_t**  
Type for defining an input callback function

**Synopsis:**

```c
typedef int PtInputCallbackProcF_t( void *, int,  
                                 void *,  
                                 size_t);

typedef PtInputCallbackProcF_t *  
PtInputCallbackProc_t;
```

**Description:**

These data types define pointers to input callback functions. The  
PtInputCallbackProcF_t type is the function type that the  
PtInputCallbackProc_t type points to. This allows you to do  
something like this:

```c
PtInputCallbackProcF_t my_input_callback;

int my_input_callback( void *, int, void *, size_t ) {  
    ...  
}
```

The compiler should detect any inconsistencies between the two  
declarations of `my_input_callback()` and give you an error (which is  
better than a “pointer mismatch” warning on the call to  
PtAppAddInput()).

**Classification:**

Photon

**See also:**

`PtAppAddInput()`

Interprocess Communication chapter of the Photon Programmer’s  
Guide
PtInsertSurface(), PtInsertSurfaceById()

Insert a control surface in front of or behind another

Synopsis:

```c
int PtInsertSurface( PtWidget_t *widget,
                     PtSurface_t *surface,
                     uchar_t brother_id,
                     int behind );

int PtInsertSurfaceById( PtWidget_t *widget,
                          uchar_t surface_id,
                          uchar_t brother_id,
                          int behind );
```

Library:

`ph`

Description:

These functions change a surface’s z coordinate by inserting the
surface in front of or behind another surface. They differ in how they
identify the control surface:

`PtInsertSurface()`

Uses the `surface` argument, which points to a `PtSurface_t`
structure that describes the control surface. This pointer must
not be NULL.

`PtInsertSurfaceById()`

Searches the control surfaces belonging to the given widget for
the one with an ID of `surface_id`.

The `widget` argument specifies the widget owning the specified
surfaces. The `brother_id` argument specifies the numerical ID of the
surface to position relative to, and the `behind` argument specifies how
to position the surface:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The surface is positioned in front of its new brother.</td>
</tr>
<tr>
<td>Nonzero</td>
<td>The surface is positioned behind.</td>
</tr>
</tbody>
</table>
Returns:

0  Success.
-1  One or both of the specified surfaces couldn’t be found.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtSurfaceBrotherBehind(), PtSurfaceBrotherInFront(),
PtSurfaceInBack(), PtSurfaceInFront(), PtSurfaceToBack(),
PtSurfaceToBackById(), PtSurfaceToFront(), PtSurfaceToFrontById()

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtIsFluxing()**

*Determine whether a container or its family is in flux*

**Synopsis:**

```c
int PtIsFluxing( PtWidget_t *container );
```

**Library:**

ph

**Description:**

This function determines whether `container` or any parent widget of `container` is currently in flux (i.e. your application is delaying any updates to the display for the container).

**Returns:**

- 1  The container’s family is in flux.
- 0  The container’s family isn’t in flux.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtContainerHold(), PtContainerRelease(), PtEndFlux(), PtStartFlux()*

“Delaying updates to the display” in the Working with Code chapter of the Photon *Programmer’s Guide*
**Synopsis:**

```c
int PtIsFocused( PtWidget_t *widget );
```

**Arguments:**

- `widget` A pointer to the widget whose focus you want to determine.

**Library:**

`ph`

**Description:**

This function returns a value indicating to what degree a widget is focused.

The widget family hierarchy is a set of trees. The widget that you would normally say has focus is called the *focus leaf*. Its parent, grandparent, and so on, up to the root of the tree (typically a window), form the *focus branch*. In other words, a widget is on the focus branch if the focus leaf is somewhere inside it.

**Returns:**

- **0** The widget isn’t focused.
- **1** The widget is on the focus branch.
- **2** The widget is the focus leaf.

**Classification:**

Photon

**Safety**

- Interrupt handler: No

*continued...*
PtIsFocused()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtContainerFindFocus(), PtContainerFocusNext(), PtContainerFocusPrev(), PtGlobalFocusNext(), PtGlobalFocusNextFrom(), PtGlobalFocusPrev(), PtGlobalFocusPrevFrom()
**Synopsis:**

```c
int PtLeave( int flags );
```

**Arguments:**

The value of `flags` can be one of:

- `Pt_EVENT_PROCESS_ALLOW`
  
  Turn the calling thread into an event reader.

- `Pt_EVENT_PROCESS_PREVENT`
  
  Turn the calling thread into a nonreader.

In most cases, it’s better to set neither of these bits in `flags`, in which case the thread’s status as event reader or nonreader doesn’t change. For more information about changing a thread’s event reader status, see “Threads” in the Parallel Operations chapter of the Photon Programmer’s Guide.

You can OR the following into the flags:

- `Pt_DELAY_EXIT`
  
  Prevent another thread from terminating the process by calling `PtExit()`.

  If another thread calls `PtExit()` after you unlock the library, or has called `PtExit()` before you last called `PtEnter()`, the `Pt_DELAY_EXIT` flag ensures that the process will not exit at least until this thread terminates or calls `PtEnter()`.

**Library:**

`ph`
**PtLeave()**

**Description:**

This function is the opposite to *PtEnter()*; it “unlocks” the library and lets other threads use Photon functions.

Don’t call *PtLeave()* if your thread hasn’t locked the library by successfully calling *PtEnter()* If you do, your application will crash if you’re lucky, or just randomly corrupt some data if you’re less lucky.

*PtLeave()* doesn’t atomically give the library lock to another thread blocked inside *PtEnter()*; the other thread gets unblocked, but then it must compete with any other threads as if it just called *PtEnter()*.

**Returns:**

0 Success. and the state of the thread didn’t change.

> 0 Success, and the state of the thread changed. The return value can be a combination of:

- *Pt_EVENT_PROCESS_ALLOW*—The thread was a reader, and *PtLeave()* was called with *flags* set to *Pt_EVENT_PROCESS_PREVENT*.

- *Pt_EVENT_PROCESS_PREVENT*—The thread was a nonreader, and *PtLeave()* was called with *flags* set to *Pt_EVENT_PROCESS_ALLOW*.

- *Pt_DELAY_EXIT*—For both the current call to *PtLeave()* and the previous call to *PtEnter()*,*flags* was set to *Pt_DELAY_EXIT*.

Some functions will reset the *Pt_DELAY_EXIT* flag bit for the thread to 0, which will affect the return value. These functions are *PtCondWait()*,**PtCondTimedWait()**, and any function that reads and processes an event (such as *PtProcessEvent()* or *PtModalBlock()*).

< 0 An error occurred; the value is a negative error code.
PtLeave()

Errors:

EINV AL The parameter flags is an incorrect value.

Examples:

You can test whether you have the Photon Library locked by evaluating PtEnter(). If the Photon Library is already locked, PtEnter() will fail with an error of -EDEADLK. When PtLeave() is called with a negative value, it is guaranteed to fail, so in this example the Photon Library has the same lock state as it had before the call to PtEnter():

```c
int eval;
if ((eval = PtEnter(0)) < 0 && eval != -EDEADLK)
    fprintf(stderr, "Couldn't enter: %s\n",
            strerror(-eval));
else
{
    PtSetResource(w, Pt_ARG_WINDOW_TITLE, text, 0);
    PtLeave(eval); // does nothing if eval == -EDEADLK
}
```

In this example, the Photon Library is locked elsewhere, and you want to unlock it to perform some lengthy operation, such as in a widget callback where not unlocking the library would “freeze” the GUI for the duration of the operation:

```c
int my_callback( PtWidget_t * widget, ApInfo_t * apinfo,
                 PtCallbackInfo_t * cbinfo )
{
    int flags;
    if ((flags = PtLeave( Pt_EVENT_PROCESS_PREVENT )) < 0)
        fprintf(stderr, "Couldn’t leave: %s\n",
                strerror(-flags));
    else {
        do_some_lengthy_stuff();
        /* This will turn your thread back into a reader if it
           was a reader before: */
        if ((flags = PtEnter( flags )) < 0)
            fprintf(stderr, "Couldn’t enter: %s\n",
                    strerror(-flags));
    }
    ...
    return( Pt_CONTINUE );
}
```
**PtLeave()**

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

- *PtEnter(), PtExit()*
- *pthread_mutex_unlock()* in the QNX Neutrino *Library Reference*
- “Threads” in the Parallel Operations chapter of the Photon *Programmer’s Guide*
PtMainLoop()
Implement an application main loop

Synopsis:

```c
void PtMainLoop( void );
```

Library:

```c
ph
```

Description:

This is a convenience function that implements an application main loop using `PhEventNext()` and `PtEventHandler()`. `PtMainLoop()` also supports background processing (WorkProcs), signals, threads, and the handling of non-Photon messages (inputs).

`PtMainLoop()` allocates an event buffer and resizes it as necessary. You can set the size yourself by calling `PtResizeEventMsg()`.

To terminate normally, your applications should call `PtExit()` within a callback function. To terminate a thread that’s running `PtMainLoop()` without terminating the entire application, call `PtQuitMainLoop()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PhEventNext()`, `PtAppAddInput()`, `PtAppAddWorkProc()`, `PtEventHandler()`, `PtExit()`, `PtQuitMainLoop()`, `PtResizeEventMsg()`

“Receiving QNX messages” in the Interprocess Communications chapter of the Photon Programmer’s Guide
PtMakeModal()

*Block all of an application’s windows, except the one containing a given widget*

**Synopsis:**

```c
void PtMakeModal( PtWidget_t *widget,
                 unsigned short cursor,
                 PgColor_t cursor_color );
```

**Library:**

`ph`

**Description:**

`PtMakeModal()` blocks all windows except the one that contains `widget` (using `PtBlockAllWindows()`), and attaches a destroyed callback that will unblock them (using `PtUnblockWindows()`).

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`ApModalWait()`, `PgColor_t`, `PtBlockAllWindows()`, `PtBlockWindow()`, `PtUnblockWindows()`

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
**PtMessageBox()**

*Pop up a message box*

**Synopsis:**

```c
int PtMessageBox( PtWidget_t *parent,  
                 char const *title,  
                 char const *question,  
                 char const *font,  
                 char const *btn );
```

**Library:**

`ph`

**Description:**

This function displays the message specified by `question`, blocking input to the specified `parent` widget until the message has been acknowledged. If no `parent` is specified, blocking won’t occur, but the message will persist until it’s acknowledged. If specified, the parent widget must be a window.

The other arguments are:

- `title`  
  The title of the dialog. If this argument is NULL, the dialog has no title bar.

- `font`  
  The name of the font to use for the message, as created by `PfGenerateFontName()`. The default is `TextFont09`.

- `btn`  
  The label to display in the dialog’s button. The default is `Ok`.

This function returns immediately.

**Returns:**

- `0`  
  Success.

- `-1`  
  Failure occurred due to lack of memory.
**PtMessageBox()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`ApError()`, `PfGenerateFontName()`, `PtAlert()`, `PtNotice()`, `PtPrompt()`

“Dialog modules” in the Working with Modules chapter of the Photon

*Programmer’s Guide*
Synopsis:

```c
void *PtModalBlock( PtModalCtrl_t *ctrl,
                    unsigned flags );
```

Library:

`ph`

Description:

`PtModalBlock()` implements a modal loop.

We recommend using `PtModalBlock()` as a replacement for the `PtModalStart() / PtProcessEvent() / PtModalEnd()` loop. This function makes it easier for you to write modal code that’s compatible with multithreaded applications.

The `PtModalCtrl_t` structure is a replacement for the “done” flag typically used by the modal loop. `PtModalUnblock()` is a replacement for setting that flag.

`PtModalBlock()` doesn’t return until `PtModalUnblock()` is called with the same value of its `ctrl` argument. The structure pointed to by `ctrl` doesn’t need to be initialized in any special way.

While `PtModalBlock()` is running, Photon events are processed by either the same thread, other threads, or both. `PtModalUnblock()` causes the corresponding `PtModalBlock()` call to return the value passed to the `result` argument.

The `flags` argument can be one of:

- **Pt(EVENT_PROCESS_PREVENT**
  Temporarily turn your thread into a nonreader: `PtModalBlock()` blocks on a condvar rather than processing events.

- **Pt(EVENT_PROCESS_ALLOW**
  Make sure that `PtModalBlock()` processes events rather than blocking on a condvar.
PtModalBlock()

If you pass 0 for flags, PtModalBlock() tries to guess the best possible behavior:

- If the thread is a nonreader, it won’t process any events.
- If the thread is an event reader, PtModalBlock() either processes events or blocks on a condvar, depending on whether there are any other event readers. If other threads turn from readers into nonreaders and back, your thread may switch between processing events and blocking on the condvar. This prevents you from going into another callback that might do another modal operation and prevent this one from completing, provided that you have enough reader threads.

Either way, the previous status of the thread as a reader or nonreader is restored before PtModalBlock() returns, even if the status was changed by a callback invoked from within PtModalBlock() rather than by PtModalBlock() itself.

If another thread calls PtExit() while this function is blocked, the function does not return, even if a third thread calls PtModalUnblock().

Returns:

NULL on error, or the value passed as the second argument to PtModalUnblock() (don’t use NULL or you won’t be able to recognize a failure).

Examples:

/* callbacks.c */
/* */
/* This application demonstrates how to obtain set */
/* up a modal dialog in an application in order to */
/* obtain information from the user before continuing. */
/* */
/* This file contains: */
/* */
/* modal_btn_done_activateCB */
/* Callback type: Pt_CB_ACTIVATE */
/* Widget: modal_btn_done */
/* */
/* This function displays how to obtain and pass back */
/* some data input by the user, and then unblock from */
/* a modal situation */
/*-------------------------------------------------------*/
/**
* nonmodal_btn_launchmodaldlg_activateCB
*/
/*
* Callback type: Pt_CB_ACTIVATE
* Widget: modal_btn_done
*/
/**
* This function launches our modal dialog, blocks other windows in the application, changes their cursors (and overrides those of their children) to be the noninput cursor, and then calls PtModalBlock() to wait for the user to return from the dialog (via modal_btn_done_activateCB()).
*/
/**
* Information obtained from the user in the modal dialog is used, and then the other windows are restored to normal.
*/
/**
*-------------------------------------------------------*/
/**
* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
/* Toolkit headers */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>
/* Local headers */
#include "globals.h"
#include "abimport.h"
#include "proto.h"

/*
* global variables
*/
PtModalCtrl_t modalctrl;
int *choice;
PtModalBlock()

/******************************************************************************
* modal_btn_done_activateCB
*******************************************************************************/

int modal_btn_done_activateCB( PtWidget_t *widget, ApInfo_t *apinfo,
    PtCallbackInfo_t *cbinfo )
{
    PtArg_t arg[1];
    int *users_choice;
    PtSetArg( &arg[0], Pt_ARG_NUMERIC_VALUE, &users_choice, 0 );
    PtGetResources( ABW_modal_nint_choice, 1, arg );
    choice = users_choice;
    PtModalUnblock( &modalctrl, NULL );

    /* eliminate ‘unreferenced’ warnings */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    return( Pt_CONTINUE );
}

/******************************************************************************
* nonmodal_btn_launchmodaldlg_activateCB
*******************************************************************************/

int nonmodal_btn_launchmodaldlg_activateCB( PtWidget_t *widget,
    ApInfo_t *apinfo,
    PtCallbackInfo_t *cbinfo )
{
    char choice_as_string[3]; /* we’re expecting only 2-digit #’s between 0 and 42 */
    unsigned short *ptr2oldcursor, base_oldcursor,
        nonmodal_oldcursor;
    int cursor_override_state, base_oldcursoroverride,
        nonmodal_oldcursoroverride;
    PtArg_t args[3];

    /*
    * Create and realize a dialog to get a user response; in
    * it we’ll ask the user to choose a value with a
    */

    return( Pt_CONTINUE );
}
* PtNumericInteger. (We created the dialog in PhAB, along
* with an internal link to it. Note that the internal
* link needs a pointer to a widget in order to place
* the dialog)
*/
ApCreateModule( ABM_modal, widget, NULL );

/*
* save away cursor data from other windows
*/
PtSetArg( &args[0], Pt_ARG_CURSOR_TYPE, &ptr2oldcursor, 0 );
PtSetArg( &args[1], Pt_ARG_CURSOR_OVERRIDE,
&cursor_override_state, 0 );
PtGetResources( ABW_base, 2, args );
base_oldcursor = *ptr2oldcursor;
basescursoroverride = cursor_override_state;
PtGetResources( ABW_nonmodal, 2, args );
nonmodal_oldcursor = *ptr2oldcursor;
onmodal_oldcursoroverride = cursor_override_state;

/*
* block user interaction to windows - note that instead of
* blocking each window individually, we’d normally use
* PtBlockAllWindows(); we’ll also alter the windows’
* cursors to reflect their blocked state
*/
PtSetArg( &args[0], Pt_ARG_FLAGS, Pt_BLOCKED, Pt_BLOCKED );
PtSetArg( &args[1], Pt_ARG_CURSOR_TYPE,
Ph_CURSOR_NOINPUT, 0 );
PtSetArg( &args[2], Pt_ARG_CURSOR_OVERRIDE, Pt_TRUE, 0 );
PtSetResources( ABW_base, 3, args );
PtSetResources( ABW_nonmodal, 3, args );

/*
* block; this will return when PtModalUnblock() is called
*/
PtModalBlock( &modalctrl, 0 );

/*
* the global variable "choice" has now been filled in;
* we’ll display that value now
*/
itoa( *choice, choice_as_string, 10 );
PtSetArg( &args[0], Pt_ARG_TEXT_STRING,
choice_as_string, 0 );
PtSetResources( ABW_nonmodal_lbl_choice, 1, args );
PtModalBlock()

/*
 * unblock user interaction to windows and restore cursors to
 * original states
 */
PtSetArg( &args[0], Pt_ARG_FLAGS, 0, Pt_BLOCKED );
PtSetArg( &args[1], Pt_ARG_CURSOR_TYPE,
    base_oldcursor, 0 );
if( base_oldcursoroVERRIDE )
    PtSetArg (&args[2], Pt_ARG_CURSOR_OVERRIDE, Pt_TRUE, 0);
else
    PtSetArg (&args[2], Pt_ARG_CURSOR_OVERRIDE, Pt_FALSE, 0);
PtSetResources( ABW_base, 3, args );
PtSetArg( &args[1], Pt_ARG_CURSOR_TYPE, nonmodal_oldcursor,
    0 );
if( nonmodal_oldcursoroVERRIDE )
    PtSetArg (&args[2], Pt_ARG_CURSOR_OVERRIDE, Pt_TRUE, 0);
else
    PtSetArg (&args[2], Pt_ARG_CURSOR_OVERRIDE, Pt_FALSE, 0);
PtSetResources( ABW_nonmodal, 3, args );

/* eliminate ‘unreferenced’ warnings */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

return( Pt_CONTINUE );
}

Classification:

Photon

Safety

Interrupt handler  No
Signal handler  No
Thread  No
PtModalBlock()

See also:

PtModalEnd(), PtModalStart(), PtModalUnblock(), PtProcessEvent()

“Threads” in the Parallel Operations chapter, and “Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
**PtModalEnd()**

Terminates modal-window processing

**Synopsis:**

```c
void PtModalEnd( void );
```

**Library:**

ph

**Description:**

This function terminates a modal-processing loop that was initiated by `PtModalStart()`.

We recommend using `PtModalBlock()` and `PtModalUnblock()` instead of a `PtModalStart() / PtProcessEvent() / PtModalEnd()` loop. `PtModalBlock()` makes it easier for you to write modal code that’s compatible with multithreaded applications.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtModalBlock()`, `PtModalStart()`, `PtModalUnblock()`, `PtProcessEvent()`

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
**PtModalStart()**

*Initiate modal-window processing*

**Synopsis:**

```c
void PtModalStart ( void );
```

**Library:**

`ph`

**Description:**

This function initiates modal processing. To do so, it holds the current process loop so that another subloop can be started. This is needed so that the “current event” before the loop is the same as after.

Once the subloop is complete, you must call *PtModalEnd()* to resume processing of the initial process loop.

If you want the parent or any windows in the application to refuse input while the modal dialog is displayed, you need to block them programmatically by setting the Pt_BLOCKED flag.

We recommend using *PtModalBlock()* and *PtModalUnblock()* instead of a *PtModalStart()* / *PtProcessEvent()* / *PtModalEnd()* loop. *PtModalBlock()* makes it easier for you to write modal code that’s compatible with multithreaded applications.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtModalStart()

See also:

PtModalBlock(), PtModalEnd(), PtModalUnblock(), PtProcessEvent()

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PtModalUnblock( PtModalCtrl_t *ctrl,
                    void *result );
```

Library:

`ph`

Description:

`PtModalUnblock()` causes the corresponding `PtModalBlock()` call to return the value passed to the `result` argument. If you call `PtModalUnblock()` more than once before `PtModalBlock()` returns, only the first call matters; don’t call `PtModalUnblock()` after `PtModalBlock()` has returned.

Returns:

- 0 Success.
- -1 An error occurred.

Examples:

See `PtModalBlock()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtModalUnblock()  

See also:  

PtModalBlock(), PtModalEnd(), PtModalStart(), PtProcessEvent()  

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
PtNextTopLevelWidget()  
Get a pointer to the next top-level widget

Synopsis:

```c
PtWidget_t * PtNextTopLevelWidget(
    PtWidget_t *widget);
```

Library:

ph

Description:

This function gets a pointer to the next top-level widget after the given widget.

Returns:

A pointer to the next top-level widget, or NULL if there isn’t one.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtFindFocusChild(), PtGetParent(), PtValidParent(),  
PtWidgetParent(), PtWidgetSkip()  
PtNotice()
Display a message and wait for acknowledgment

Synopsis:

```c
void PtNotice( PtWidget_t *parent,
              PhPoint_t const *location,
              char const *title,
              PhImage_t const *image,
              char const *message,
              char const *msgFont,
              char const *btnText,
              int flags );
```

Arguments:

- **parent**: A pointer to the parent widget of the dialog (usually a window). By setting the `flags`, you can block the parent and/or position the dialog relative to it.

- **location**: A pointer to a `PhPoint_t` structure that specifies the location of the dialog relative to the parent or console, depending on the `flags`. If `location` is NULL, the dialog is centered.

- **title**: The title for the dialog. If you don’t want a title bar, set this argument to NULL.

- **image**: A pointer to a `PhImage_t` that specifies an icon to be displayed beside the message. If you don’t want an icon, set this argument to NULL.

- **message**: The message to display.

- **msgFont**: The font for the message text; the default is `TextFont09`. You should create the font name by calling `PfGenerateFontName()`.

- **btnText**: The text to be displayed in the button. If this is set to NULL, a default of `&OK` is used. The `btnText` argument lets you define a shortcut key — place an ampersand (`&`) in front of the character to be used as the shortcut.
**PtNotice()**

**btnFont** The font to use in the button. If this is NULL, a default font of **TextFont09** is used. You should create the font names by calling **PfGenerateFontName()**.

**flags** Flags that specify the behavior for the dialog. This can be up to one of the following:

- **Pt_CENTER** — center the dialog.
- **Pt_LEFT** — left-align the dialog (the default).
- **Pt_RIGHT** — right-align the dialog.

with any combination of the following:

- **Pt_BLOCK_ALL** — block all of the application’s windows while the dialog is displayed.
- **Pt_BLOCK_PARENT** — block the widget specified by the **parent** argument (if non-NULL).
- **Pt_ESC_DISABLE** — disable the ESC key as a means of dismissing the dialog.
- **Pt_MODAL** — the same as (**Pt_WAIT** | **Pt_BLOCK_ALL**).
- **Pt_RELATIVE** — position the dialog relative to the given parent widget. If this bit isn’t set or **parent** is NULL, the dialog is positioned relative to the current console.
- **Pt_WAIT** — don’t return from the function until the user dismisses the dialog.

**Pt_BLOCK_ALL** overrides **Pt_BLOCK_PARENT**.

**Library:**

**ph**
**PtNotice()**

**Description:**
This function displays a message and waits for you to acknowledge it. By setting the flags, you can make `PtNotice()` work modally, meaning that it doesn’t return until you respond.

![A sample dialog displayed by PtNotice().](image)

**Examples:**

```cpp
char Helvetica12[MAX_FONT_TAG];

PtNotice( ABW_base, NULL, "George Crabbe", NULL,
"Books cannot always please, however good;
Minds are not ever craving for their food.",
PfGenerateFontName("Helvetica", 0, 12, Helvetica12),
"How &true!", NULL, Pt_BLOCK_PARENT);
```

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1548 Chapter 13 • Pt—Widget Toolkit
PtNotice()

See also:
ApError(), PfGenerateFontName(), PhImage_t, PhPoint_t,
PtAlert(), PtPassword(), PtPrompt()

“Dialog modules” in the Working with Modules chapter of the Photon
Programmer’s Guide
**PtPassword()**

Prompt for a password

**Synopsis:**

```c
int PtPassword( PtWidget_t *parent,
                PhPoint_t const *location,
                char const *title,
                PhImage_t const *image,
                char const *message,
                char const *msg_font,
                char const **buttons,
                char const **btn_fonts,
                char const *text_font,
                int (*validate_f)(void *,char const *),
                void *validate_data,
                char const *echo,
                int flags );
```

**Arguments:**

- **parent**
  A pointer to the parent widget of the dialog (usually a window). By setting the *flags*, you can block the parent and/or position the dialog relative to it.

- **location**
  A pointer to a `PhPoint_t` structure that specifies the location of the dialog relative to the parent or console, depending on the *flags*. If *location* is NULL, the dialog is centered.

- **title**
  The title for the dialog. If you don’t want a title bar, set this argument to NULL.

- **image**
  A pointer to a `PhImage_t` structure that specifies an icon to display beside the message. If you don’t want an icon, set this argument to NULL.

- **message**
  The message to display.

- **msg_font**
  The font for the message text; the default is `TextFont09`. You should create the font name by calling `PfGenerateFontName()`.
**PtPassword()**

*buttons*  
A pointer to an array of strings to be displayed in the buttons. If non-NULL, this array must contain exactly two strings. The first is for the cancel button, and the second is for the accept button.

All the button-text arguments let you define shortcut keys. Place an ampersand (&) in front of the character that you want to be the shortcut. For example, if you specify &Yes, the Y is underlined in the button, and you can press y or Y to select the button.

If *buttons* is NULL, the function uses &Cancel and &Ok for the buttons.

*btn_fonts*  
A pointer to an array of strings naming the fonts to be used in the buttons. If this argument is NULL, *TextFont09* is used for all the buttons. Otherwise, this array must contain at least *btnCount* font names. You should create the font names by calling *PfGenerateFontName()*.

*text_font*  
The name of the font to use for the text. You should create the font name by calling *PfGenerateFontName()*.

*validate_f*  
A pointer to a password-validation function, which is of the form:

```c
int validate( void *data,  
              char const *password_entered )
```

The arguments to the validation function are:

- *data* — arbitrary data you need in the function. It’s the *validate_data* that you pass to *PtPassword()*.
- *password_entered* — the password that the user typed.

The validation function must return one of:
PtPassword()

- Pt_PWD_ACCEPT — the password is acceptable.
- Pt_PWD_RETRY — the password is unacceptable. Let the user try again.
- Pt_PWD_REJECT — the password is unacceptable, Don’t let the user try again.

**validate_data**  
User data that’s passed to your validation function.

**echo**  
A multibyte character to replace the characters that the user types. If NULL, * is used. Specify " " if you don’t want any echoing to take place (i.e. the text field appears non-interactive but there’s no indication of how many characters the user types, which might be appealing for higher security restrictions).

**flags**  
Flags that define the behavior for the dialog. This can be up to one of the following:

- Pt_CENTER — center the dialog.
- Pt_LEFT — left-align the dialog (the default).
- Pt_RIGHT — right-align the dialog.

with any combination of the following:

- Pt_BLOCK_ALL — block all of the application’s windows while the dialog is displayed.
- Pt_BLOCK_PARENT — block the widget specified by the parent argument (if non-NULL).
- Pt_ESC_DISABLE — disable the ESC key as a means of dismissing the dialog.
- Pt_MODAL — the same as Pt_BLOCK_ALL.
- Pt_RELATIVE — position the dialog relative to the given parent widget. If this bit isn’t set or parent is NULL, the dialog is positioned relative to the current console.
PtPassword()

Library:

ph

Description:

This function displays a dialog that prompts the user for a password.

Returns:

Pt_PWD_ACCEPT
The user typed an acceptable password.

Pt_PWD_REJECT
The password that the user typed was rejected.

Pt_PWD_CANCEL
The user aborted the operation.

Otherwise, the function returns -1 to indicate some lower-level error (e.g. the dialog couldn’t be created).

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PfGenerateFontName(), PhImage_t, PhPoint_t, PtAlert(), PtNotice(), PtPrompt()
PtPassword()

“Dialog modules” in the Working with Modules chapter of the Photon Programmer’s Guide
**Synopsis:**

```c
void PtPositionMenu( PtWidget_t *menu,
                     PhEvent_t *event );
```

**Library:**

ph

**Description:**

This function sets the Pt_ARG_POS resource of the provided *menu* widget. How the function sets this resource is determined by the type of its parent and the specified event.

If the provided *menu* is a child of a PtMenuButton widget, the menu is positioned relative to that menu button (to the right or below, depending on the menu button’s flags).

If the provided *menu* isn’t a child of a PtMenuButton widget and the specified *event* is a pointer event, the menu is positioned at the event’s position. If the *event* isn’t a pointer event and the menu has a parent, the menu is positioned at the upper-left corner of that parent.

If the provided *menu* isn’t a child of a PtMenuButton widget and the specified *event* isn’t a pointer event, the menu is positioned at 0,0.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtPositionMenu()**

See also:

PhEvent_t
PtPreventExit()  
Prevent a Photon application from exiting

Synopsis:

void PtPreventExit( void );

Library:

ph

Description:

PtPreventExit() lets Photon know that it isn’t safe to exit your application.

In a multithreaded application, any thread can call PtExit(), but another thread might be in the middle of an important operation, such as writing a file. To prevent this situation from arising, call PtPreventExit() before starting the operation, and call PtAllowExit() when it’s done.

Instead of calling PtPreventExit() and PtAllowExit() directly, you’re better off calling PtEnter() and PtLeave() with Pt_DELAY_EXIT set in the flags. For a discussion of the difference between these functions and using Pt_DELAY_EXIT, see “Exiting a multithreaded program” in the Parallel Operations chapter of the Photon Programmer’s Guide.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>
PtPreventExit()

See also:

PtAllowExit(), PtEnter(), PtExit(), PtLeave()

Parallel Operations chapter of the Photon Programmer’s Guide
**PtPrintPropSelect()**

Change the printing options for a selected printer via a modal dialog

**Synopsis:**

```c
int PtPrintPropSelect(
    PtWidget_t *parent,
    char const *title,
    PtPrintPropSelectionInfo_t *info);
```

**Arguments:**

- **parent** A pointer to the parent widget. If this argument isn’t NULL, the parent widget is blocked while the dialog is displayed.
- **title** The title of the print-properties dialog.
- **info** A pointer to a PtPrintPropSelectionInfo_t structure (see below).

**Library:**

ph

**Description:**

This function displays a dialog for modifying most of the print parameters for a particular printer:

![Print Properties Dialog](image)

Chapter 13 • Pt—Widget Toolkit 1559
PtPrintPropSelect()

The display is limited to parameters and values that are valid for the selected printer.

A print context is passed to the function in the info argument. This print context is modified according to the state of the dialog when you press the Apply or Done button.

PtPrintPropSelect() also lets you load and save your own preferences, which are stored in personal print configuration files.

The PtPrintSel widget calls PtPrintPropSelect() when you press the Preferences button.

PtPrintPropSelectionInfo_t structure

The PtPrintPropSelectionInfo_t structure includes at least:

PpPrintContext_t *pcontext

A pointer to a PpPrintContext_t structure that describes the print context. You must create this structure by calling PpCreatePC(). Optionally, you can initialize the print context by calling PpLoadPrinter().

PhPoint_t pos

The position of the print-properties dialog; the meaning of this position depends on the parent argument, and on whether the Pt_PSP_CENTER bit is set or cleared in the flags member:

<table>
<thead>
<tr>
<th>parent</th>
<th>Bit</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>Clear</td>
<td>Relative to screen</td>
</tr>
<tr>
<td>NULL</td>
<td>Set</td>
<td>Ignored; the dialog is centered on the screen</td>
</tr>
<tr>
<td>non-NUL</td>
<td>Clear</td>
<td>Relative to parent</td>
</tr>
</tbody>
</table>

continued...
PtPrintPropSelect()

<table>
<thead>
<tr>
<th>parent</th>
<th>Bit</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-NULL</td>
<td>Set</td>
<td>Ignored; the dialog is centered on the parent</td>
</tr>
</tbody>
</table>

int flags

The flag bits are:

- Pt_PSP_CENTER — center the print-properties dialog with respect to the screen or its parent. The pos argument is ignored if you set this flag.
- Pt_PSP_NO_GRAPHICS — don’t display the Graphics pane.
- Pt_PSP_NO_MARGINS — don’t display the Margins pane.
- Pt_PSP_NO_PAPER — don’t display the Paper pane.
- Pt_PSP_NO_PRINT_ORDER — don’t display the Print Order pane.
- Pt_PSP_NO_PRINTERS — don’t display the Printers pane.
- Pt_PSP_NO_DEFAULTS — don’t display the Defaults pane.
- Pt_PSP_NOCANCEL_BUTTON — don’t display the Cancel button in the main button pane.
- Pt_PSP_NOAPPLY_BUTTON — don’t display the Apply button in the main button pane.
- Pt_PSP_NODONE_BUTTON — don’t display the Done button in the main button pane.
- Pt_PSP_NOSAVE_DFLT_BUTTON — don’t display the Save Personal Defaults button.
- Pt_PSP_NOLOAD_DFLT_BUTTON — don’t display the Load Personal Defaults button.
- Pt_PSP_NOLOAD_GLOBAL_DFLT_BUTTON — don’t display the Load Global Defaults button.
PtPrintPropSelect()

```c
int num_args
   The number of resources specified in the args array.
PtArg_t *args
   A pointer to an array of resources for the dialog; see below.
```

**Dialog “resources”**

You can customize the print-properties dialog as if it were a widget with resources. You can set or get the values of these pseudo-resources for a `PtPrintSel` widget (see the `Widget Reference`).

All the resources are of this type:

<table>
<thead>
<tr>
<th>C type</th>
<th>Pt type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *</td>
<td>String</td>
<td>See below</td>
</tr>
</tbody>
</table>

**Main dialog buttons**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Pt ARG PSP LBL CANCEL</code></td>
<td>Cancel</td>
</tr>
<tr>
<td><code>Pt ARG PSP LBL APPLY</code></td>
<td>Apply</td>
</tr>
<tr>
<td><code>Pt ARG PSP LBL DONE</code></td>
<td>Done</td>
</tr>
</tbody>
</table>

**Main dialog titles**
### Resource Selection

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE</td>
<td>Printer Properties</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE_PAPER</td>
<td>Paper</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE_GRAPHICS</td>
<td>Graphics</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE_MARGINS</td>
<td>Margins</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE_PRINT_ORDER</td>
<td>Print Order</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE_PRINTERS</td>
<td>Printers</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_TITLE_DFLT</td>
<td>Defaults</td>
</tr>
</tbody>
</table>

#### Paper pane

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_PSP_LBL_PAPERSIZE</td>
<td>Paper Size</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_PAPERSOURCE</td>
<td>Paper Source</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_PAPERTYPE</td>
<td>Paper Type</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_ORIENTATION</td>
<td>Orientation</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_PORTRAIT</td>
<td>Portrait</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_LANDSCAPE</td>
<td>Landscape</td>
</tr>
</tbody>
</table>

#### Graphics pane

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_PSP_LBL_COLORMODE</td>
<td>Color Mode</td>
</tr>
</tbody>
</table>

*continued...*
# PtPrintPropSelect()

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Pt_ARG_PSP_LBL_DITHERING</code></td>
<td>Dithering</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_RESOLUTION</code></td>
<td>Resolution</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_INTENSITY</code></td>
<td>Intensity</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_DARKEST</code></td>
<td>Darkest</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_LIGHTEST</code></td>
<td>Lightest</td>
</tr>
</tbody>
</table>

## Margins pane

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Pt_ARG_PSP_LBL_TOP</code></td>
<td>Top</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_BOTTOM</code></td>
<td>Bottom</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_LEFT</code></td>
<td>Left</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_RIGHT</code></td>
<td>Right</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_UNITS</code></td>
<td>Units</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_INCHES</code></td>
<td>1000th inch</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_MILLIMETERS</code></td>
<td>100th mm</td>
</tr>
</tbody>
</table>

## Defaults pane

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Pt_ARG_PSP_LBL_SAVE_DFLT</code></td>
<td>Save Personal Defaults</td>
</tr>
<tr>
<td><code>Pt_ARG_PSP_LBL_LOAD_DFLT</code></td>
<td>Load Personal Defaults</td>
</tr>
</tbody>
</table>

*continued*
PtPrintPropSelect()

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_PSP_LBL_LOAD_GLOBAL_DFLT</td>
<td>Load Factory Settings</td>
</tr>
</tbody>
</table>

Printers pane

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_PSP_LBL_DEFAULT_PRINTER</td>
<td>Default Printer</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_CURRENT_PRINTER</td>
<td>Current Printer</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_FONTMAP</td>
<td>Font Map</td>
</tr>
</tbody>
</table>

Print Order pane

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_PSP_LBL_REVERSED</td>
<td>Print Reversed Order</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_DOUBLE_SIDED</td>
<td>Print Double Sided</td>
</tr>
<tr>
<td>Pt_ARG_PSP_LBL_COLLATED</td>
<td>Print Collated</td>
</tr>
</tbody>
</table>

Returns:

Pt_PSP_ERROR  An error occurred:

- The info argument is NULL or points to an invalid structure.
- The printcontext is NULL.
- No printers have been installed.
PtPrintPropSelect()

Pt_PSP_DONE You pressed the Done button.

Pt_PSP_CANCEL You pressed the Cancel button.

If you press the Apply button, the contents of the print context might change, no matter which button you ultimately use to close the dialog. If PtPrintPropSelect() returns Pt_PSP_CANCEL, your application shouldn’t assume that nothing changed.

Examples:

/*****************************/
/*
 * psp.c
 * Sample program illustrates usage of
 * PtPrintPropSelect() convenience function.
 * Compile as follows:
 * $ gcc -lph -o psp psp.c
 * Run as follows:
 * $ ./psp
 * *****************************/

#include <stdio.h>
#include <stdlib.h>
#include <Ph.h>
#include <Pt.h>

int main(int argc, char **argv )
{
    PtWidget_t *win;
    PtPrintPropSelectionInfo_t info;
    int r;
    PhDim_t dim;
    PtArg_t args[3];

    // Set base window dimension to 250x250 pixels
    dim.w = dim.h = 250;
    PtSetArg( args, Pt_ARG_DIM, &dim, 0 );

    // Connect to Photon, initialize widget lib, and
// create a base window
if ( NULL == (win = PtAppInit( NULL, &argc,
        argv, 1, args )) ) {
    fprintf( stderr, "\nPtAppInit failed.\n" );
        exit( -1 );
}

// Realize the base window
PtRealizeWidget( win );

// Initialize the info structure
memset( &info, 0x0, sizeof( PtPrintPropSelectionInfo_t ) );

// Modify a couple of string resources
// Change the 'Apply' button's label
PtSetArg( &args[0], Pt_ARG_PSP_LBL_APPLY,
        "MyApply", 0 );
// Change the 'Done' button's label
PtSetArg( &args[1], Pt_ARG_PSP_LBL_DONE,
        "MyDone", 0 );
// Change the 'Margins' pane
PtSetArg( &args[2], Pt_ARG_PSP_LBL_TITLE_MARGINS,
        "MyMargins", 0 );

info.num_args = 3;
info.args = args;

// Set up the flags to prevent the display of the
// 'Cancel' button.
info.flags = Pt_PSP_NO_CANCEL_BUTTON;

// Create a print-context.
if ( NULL == (info.pcontext = PpCreatePC()) ) {
    fprintf( stderr, "\nUnable to create print context.\n" );
        PtExit( -1 );
}

// PtPrintPropSelect() will be blocked in its own modal loop
// until the user presses the 'Esc' key, the 'MyDone' button,
// or closes the dialog.
//
// r = PtPrintPropSelect( win, "Adjust Settings", &info );

// If the 'MyApply' button is pressed, then the current
// settings in the dialog are applied to the print context.
if ( Pt_PSP_ERROR == r )
    fprintf( stderr,
        "\nPtPrintPropSelect() failed. The print context was not \modified.\n" );
PtPrintPropSelect()

else if ( Pt_PSP_DONE == r )
    fprintf( stderr,
             "\n'MyDone' button pressed. The print context may have \nbeen modified.\n" );

else // Pt_PSP_CANCEL == r
    fprintf( stderr,
             "\n'Esc' key pressed or dialog closed. The print \ncontext may have been modified.\n" );

    // Free the resources used by our print context.
    PpReleasePC( info.pcontext );

    PtMainLoop();
    return EXIT_SUCCESS;
}

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PpCreatePC(), PpPrintContext_t, PpSetPC(), PtPrintSelect(), PtPrintSelection()

PtPrintSel in the Widget Reference

“Dialog modules” in the Working with Modules chapter, and the Printing chapter of the Photon Programmer’s Guide
**PtPrintSelect()**

Display a custom modal dialog for selecting print options

### Synopsis:

```c
int PtPrintSelect(
    PtWidget_t *parent,
    const char *title,
    PtPrintSelectionInfo_t *info);
```

### Library:

ph

### Description:

This function displays a modal dialog that lets you select print options and initiate printing. This function is similar to `PtPrintSelection()`, but `PtPrintSelect()` lets you customize the appearance and function of the dialog and the underlying `PtPrintSel` widget.

The dialog is parented off the `parent` widget, which may be NULL; if non-NULL, that parent widget is blocked and its cursor is changed to reflect this.

The title of the dialog is given by `title`; if this is NULL a default title of “Select Printer” is used.

The user may click one of these buttons:

- Print—initiate printing
- Preview—view the material to be printed
- Cancel—cancel the operation

### Returns:

An integer that indicates which button was pressed:

- `Pt_PRINTSEL_PRINT`
- `Pt_PRINTSEL_PREVIEW`
- `Pt_PRINTSEL_CANCEL`
or Pt_PRINTSEL_ERROR, which indicates that an error was made in custom_args.

The do_preview member of the print context is also set to indicate whether Print or Preview was selected. This means that the context can be passed to the printing function, which will spawn the print-preview application if necessary.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**


Printing chapter, and “Setting resources” in the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PtPrintSelection( PtWidget_t *parent,
    PhPoint_t const *pos,
    const char *title,
    PpPrintContext_t *context,
    unsigned flags);
```

Arguments:

- **parent**: A pointer to the parent widget for the dialog. If the parent widget isn’t NULL, it’s blocked and its cursor is changed to reflect this.
  
The `parent` argument is also used to position the dialog, as described below.

- **pos**: A pointer to a `PhPoint_t` structure that specifies the position of the dialog (see below).

- **title**: The title of the dialog; if this is NULL, a default title of “Select Printer” is used.

- **context**: A pointer to a `PpPrintContext_t` structure that was created by `PpCreatePC()`. This pointer must not be NULL. `PtPrintSelection()` updates the print context.

- **flags**: Flags that enable or disable parts of the user interface (see below). This argument should normally be set to `Pt_PRINTSEL_DFLT_LOOK`.

Library:

- **ph**

Description:

This convenience function displays a `PtPrintSel` widget and a button-pane in a modal dialog. It lets you select print options and initiate printing:
The `parent` and `pos` arguments determine where the dialog is to appear:

continued…
### PtPrintSelection()

<table>
<thead>
<tr>
<th>parent</th>
<th>pos</th>
<th>Position of dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>Center of the screen</td>
</tr>
<tr>
<td>NULL</td>
<td>Non-NULL</td>
<td><em>pos</em> relative to the screen</td>
</tr>
<tr>
<td>Non-NULL</td>
<td>NULL</td>
<td>Center of <em>parent</em></td>
</tr>
<tr>
<td>Non-NULL</td>
<td>Non-NULL</td>
<td><em>pos</em> relative to <em>parent</em></td>
</tr>
</tbody>
</table>

Your application can call `PpSetPC()` to modify the print context before calling `PtPrintSelection()`. These modifications are propagated to the `PtPrintSel` widget. Note that some modified context settings may not be displayed or may be reset (e.g., if the selected printer doesn’t support double-sided printing and the `Pp_PC_DUPLEX` member was set in the context before calling `PtPrintSelection()`).

The valid flags are those defined for `Pt_ARG_PRINT_FLAGS`:

- `Pt_PRINTSEL_FILE_PANE` Enable the Send to file pane.
- `Pt_PRINTSEL_NO_COPIES` Disable the Copies widget.
- `Pt_PRINTSEL_NO_PAGE_RANGE` Disable the Print Range toggle button and the From and To fields.
- `Pt_PRINTSEL_NO_PRINTSELECT` Disable the printer-name combobox. Physical output goes to the default physical printer whose name is shown.
- `Pt_PRINTSEL_NO_SELECT_RANGE` Disable the Print Selection toggle button.
- `Pt_PRINTSEL_PREFERENCES` Enable the Preferences button.
PtPrintSelection()

Pt_PRINTSEL_SETTINGS_PANE
Enable the Print Pages, Print Order and Copies panes.

The following flag macros are defined in <PtPrintSel.h>:

Pt_PRINTSEL_ALL_PANES
Pt_PRINTSEL_FILE_PANE | Pt_PRINTSEL_SETTINGS_PANE

Pt_PRINTSEL_DFLT_LOOK
Pt_PRINTSEL_FILE_PANE | Pt_PRINTSEL_SETTINGS_PANE
| Pt_PRINTSEL_PREFERENCES

The user may click one of these buttons:
• Print
• Preview
• Cancel

Returns:
An integer that indicates which button was pressed:
• Pt_PRINTSEL_PRINT
• Pt_PRINTSEL_PREVIEW
• Pt_PRINTSEL_CANCEL

The Pp_PC_DO_PREVIEW member of the print context is set when the user presses the Preview button. This means that the context can be passed to the printing function, which spawn the print-preview application if necessary.
PtPrintSelection()

Examples:

See PpContinueJob().

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhPoint_t, PpPrintContext_t, PpPrintWidget(), PpSetPC(), PtPrintSelect(), PtPrintPropSelect()`

PtPrintSel in the Photon Widget Reference

“Dialog modules” in the Working with Modules chapter, and the Printing chapter of the Photon Programmer’s Guide
**PtProcessEvent()**

Standard Photon event-handling function

**Synopsis:**

```c
void PtProcessEvent( void );
```

**Library:**

```c
ph
```

**Description:**

This function is used primarily for modal-dialog-event handling. If this case, be sure to call PtModalStart() before the event-handling loop, and PtModalEnd() after it.

---

We recommend using PtModalBlock() and PtModalUnblock() instead of a PtModalStart() / PtProcessEvent() / PtModalEnd() loop. PtModalBlock() makes it easier for you to write modal code that’s compatible with multithreaded applications.

---

If a Photon event is pending, this function processes the event and returns. If no event is pending, or if no work procedure has been defined, the function blocks until an event is received.

Before waiting for an event, this function performs an equivalent of PtLeave(Pt_EVENT_PROCESS_ALLOW). This turns the calling thread into an event reader if it wasn’t one already. If you passed the Pt_DELAY_EXIT flag to PtEnter() before calling this function, it will also disable the effect of that flag.

After getting an event, PtProcessEvent() performs an equivalent of PtEnter(0). This means that if another thread called PtExit() while the function was waiting for an event, PtProcessEvent() doesn’t invoke any callbacks and will not return.

**Classification:**

Photon
PtProcessEvent()

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtModalBlock(), PtModalEnd(), PtModalStart(), PtModalUnblock(),
“Modal dialogs” in the Window Management chapter of the Photon
Programmer’s Guide
PtPrompt()
Display a message and get textual input from the user

Synopsis:

```c
int PtPrompt( PtWidget_t *parent,
             PhPoint_t const *location,
             char const *title,
             PhImage_t const *image,
             char const *message,
             char const *msgFont,
             int btnCount,
             char const **buttons,
             char const **btnFonts,
             int defBtn,
             int escBtn,
             short textLength,
             char *text,
             char const *textFont,
             PhDim_t const *text_dim,
             int flags );
```

Arguments:

- **parent**: A pointer to the parent widget of the dialog (usually a window). By setting the flags, you can block the parent and/or position the dialog relative to it.

- **location**: A pointer to a `PhPoint_t` structure that specifies the location of the dialog relative to the parent or console, depending on the flags. If `location` is NULL, the dialog is centered.

- **title**: The title for the dialog. If you don’t want a title bar, set this argument to NULL.

- **image**: A pointer to a `PhImage_t` structure that specifies an icon to display beside the message. If you don’t want an icon, set this argument to NULL.

- **message**: The message to display.

- **msgFont**: The font for the message text; the default is `TextFont09`. You should create the font name by calling `PfGenerateFontName()`.
PtPrompt()

**btnCount**  The number of buttons to display

**buttons**  A pointer to an array of strings to be displayed in the buttons. This array must contain at least *btnCount* strings.

All the button-text arguments let you define shortcut keys. Place an ampersand (&) in front of the character that you want to be the shortcut. For example, if you specify &Yes, the Y is underlined in the button, and you can press y or Y to select the button.

**btnFonts**  A pointer to an array of strings naming the fonts to be used in the buttons. If this argument is NULL, **TextFont09** is used for all the buttons. Otherwise, this array must contain at least *btnCount* font names. You should create the font names by calling **PfGenerateFontName()**.

**defBtn**  The number of the button that initially has focus when the dialog is realized.

**escBtn**  The number of the button that’s bound to the Esc key. If you wish to disable the Esc key, set this argument to 0. If Esc is enabled, the close button is included in the dialog’s titlebar (if there is one). Closing the dialog in this manner is the same as pressing the Esc key; the dialog closes and the escBtn button is selected.

**text**  A pointer to a buffer that sets the initial textbox text, and in which the text is stored.

You need to initialize this buffer, or unwanted characters may appear in the text box.

**textLength**  The size of the text buffer, in bytes.

**textFont**  The name of the font to use for the text. You should create the font name by calling **PfGenerateFontName()**.
PtPrompt()

`text_dim` A pointer to a `PhDim_t` structure that specifies the dimensions of the text-input area.

`flags` Flags that define the behavior for the dialog. This can be up to one of the following:

- `Pt_CENTER` — center the dialog.
- `Pt_LEFT` — left-align the dialog (the default).
- `Pt_RIGHT` — right-align the dialog.

with any combination of the following:

- `Pt_BLOCK_ALL` — block all of the application’s windows while the dialog is displayed.
- `Pt_BLOCK_PARENT` — block the widget specified by the `parent` argument (if non-NULL).
- `Pt_ESC_DISABLE` — disable the ESC key as a means of dismissing the dialog.
- `Pt_MODAL` — the same as `Pt_BLOCK_ALL`.
- `Pt_MULTITEXT` — use a multiline instead of a single-line text field.
- `Pt_RELATIVE` — position the dialog relative to the given parent widget. If this bit isn’t set or `parent` is NULL, the dialog is positioned relative to the current console.

`Pt_BLOCK_ALL` overrides `Pt_BLOCK_PARENT`.

Library:

`ph`

Description:

`PtPrompt()` displays a dialog that prompts you for input, and can contain any number of buttons so that you can respond. This function is similar to `PtAlert()`, but lets you type a line of text. `PtPrompt()`
PtPrompt() works modally, which means that it doesn’t return until you choose a button.

![A sample dialog displayed by PtPrompt().](image)

**Returns:**

The number of the button pressed, or -1 if an error occurred.

**Examples:**

```c
int answer;
char const *btns[] = { "&OK", "&Cancel" };
char text[31]="Default text";

answer = PtPrompt( base_wgt, NULL, "Identify yourself!", NULL,
                   "Enter your name:", NULL, 2, btns, NULL, 1, 2,
                   30, text, NULL, NULL, 0 );

switch( answer ) {
  case 1:
    /* ok */
    printf("You pressed OK and typed: %s\n", text);
    break;
  case 2:
    /* cancel */
    break;
  case -1:
    printf("An error occurred.\n");
    break;
}
```
**PtPrompt()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`ApError()`, `PfGenerateFontName()`, `PhDim_t`, `PhImage_t`, `PhPoint_t`, `PtAlert()`, `PtNotice()`, `PtPassword()`

“Dialog modules” in the Working with Modules chapter of the Photon

*Programmer’s Guide*
PtPulseArm()
Arm a Photon pulse for delivery

Synopsis:

```c
int PtPulseArm( PtAppContext_t app,
                pid_t pulse,
                struct sigevent *msg );
```

Library:

ph

Description:

This function arms a Photon pulse and creates a “pulse message” to be sent to another process. The other process can use the pulse message and MsgDeliverEvent() to send the pulse.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

The `pulse` argument is a pulse ID returned by PtAppCreatePulse().

The `msg` argument points a `sigevent` that’s filled in by the function. You’ll need to send it to the process that’s going to deliver the pulse.

Returns:

0 Success.

-1 An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 13 • Pt—Widget Toolkit  1583
PtPulseArm()

See also:

PtAppCreatePulse(), PtAppDeletePulse(), PtAppPulseTrigger(), PtChannelCreate()

MsgDeliverEvent() in the QNX Neutrino Library Reference

Interprocess Communication in the Photon Programmer’s Guide
Synopsis:

```
PhSysInfo_t * PtQuerySystemInfo(
    PtWidget_t *widget,
    PhSysInfo_t *sys_ptr);
```

Library:

```
ph
```

Description:

This function queries the system for information on the given `widget`:

- system bandwidth
- graphics drivers and their capabilities
- pointing devices
- keyboard devices.

The information is stored in the `PhSysInfo_t` structure pointed to by `sys_ptr`.

This function calls `PhQuerySystemInfo()`, but buffers the information, reducing the number of messages sent to the Photon server. It calls `PhQuerySystemInfo()` if the data has been made invalid since the previous call because:

- the window containing the widget was moved or resized
- you switched consoles
- a graphics or input driver was started or stopped
- someone started or stopped dittoing you
- ...

The rectangular area passed to `PhQuerySystemInfo()` is the extent of the window containing the widget (or the widget itself if it’s a window).
PtQuerySystemInfo()

Returns:
A pointer to the **PhSysInfo_t** structure passed to the function, or NULL if an error occurred.

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PtQuerySystemInfo()*, *PhSysInfo_t*

**Synopsis:**

```
int PtQuitMainLoop( void );
```

**Library:**

```
ph
```

**Description:**

This function causes `PtMainLoop()` in the calling thread to return right after it finishes processing the current event.

`PtQuitMainLoop()` doesn’t affect any modal operations that the thread is currently doing; if you call `PtQuitMainLoop()` from within a modal loop, there’s no way for `PtMainLoop()` to return until after the modal loop has completed.

Keep in mind that if you let your `main()` function return, `exit()` is called and your application is terminated without letting any widgets or threads do any cleaning up. It’s better to call `PtExit()` instead – the main purpose of `PtQuitMainLoop()` is to let you terminate threads running `PtMainLoop()` without terminating the application.

To ensure your application doesn’t terminate, put a `pthread_exit()` call after the `PtMainLoop()` call in `main()`. In PhAB, changes to `main()` get overwritten by PhAB, so you should declare a global header for your application and map `PtMainLoop()` to your own function using a macro. For example:

```c
#define PtMainLoop
void MyMainLoop( void ) {
    PtMainLoop();
    pthread_exit(NULL);
}
```
**PtQuitMainLoop()**

**Returns:**

- 0  Success.
- -1  The thread has already called *PtQuitMainLoop()*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtExit(), PtMainLoop()*

“Threads” in the Parallel Operations chapter of the Photon *Programmer’s Guide*
**PtRealizeWidget()**

*Make a widget and its children visible and possibly interactive*

**Synopsis:**

```c
int PtRealizeWidget( PtWidget_t *widget );
```

**Library:**

ph

**Description:**

This function makes a widget and its children visible to the user and possibly interactive. To create a hierarchy of widgets, you typically make successive calls to `PtCreateWidget()`, and then call `PtRealizeWidget()`, passing it the root of the hierarchy.

Some widgets (for example, menus) have `Pt_DELAY_REALIZE` set in their `Pt_ARG_FLAGS`. Such delay-realized widgets aren’t visibly rendered when their ancestors are realized. Although they’re present in the hierarchy, delay-realized widgets become visible only when the application realizes them specifically with a call to `PtRealizeWidget()`. An application might do this, for example, if the user requested it to activate a menu.

**Returns:**

0 Success.

-1 Out of memory, or an invalid widget class was specified.

**Examples:**

See `PtContainerGiveFocus()` and `PtClearWidget()`.

**Classification:**

Photon
PtRealizeWidget()

Safety

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtCreateWidget(), PtDestroyWidget(), PtUnrealizeWidget()

“Widget life cycle” in the Introduction to the Photon Programmer’s Guide
PtReattach()  
Send an application to another Photon server

Synopsis:

```c
int PtReattach( char *device );
```

Library:

```c
ph
```

Description:

This function unrealizes all top-level widgets in an application, disconnects them from the current Photon server, connects to the Photon server indicated by `device` (e.g. `/net/sam/dev/photon`), and realizes the top-level widgets at the new location.

As a result, the PHOTON environment variable is set and the user interface is rehosted to the new Photon server.

Returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>-1</td>
<td>Unable to connect to <code>device</code> due to insufficient memory.</td>
</tr>
</tbody>
</table>

Examples:

```c
int transport_edit_activate( PtWidget_t *widget, void *data, PtCallbackInfo_t cbinfo );
{
    PtTextCallback_t *tcb=cbinfo->cbdata;
    PtReattach( tcb->text );
}
```

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>continued...</td>
<td></td>
</tr>
</tbody>
</table>
**PtReattach()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtRelease()**

*Decrement the hold count, potentially permitting all widgets to be repaired*

**Synopsis:**

```c
int PtRelease( void );
```

**Library:**

`ph`

**Description:**

This function decrements the hold count, which was previously incremented by a call to `PtHold()`. When the count reaches 0, the Photon libraries repair any damaged widgets.

*This function is the same as `PtUpdate()`.*

**Returns:**

The current hold count, or -1 if an error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtFlush()`, `PtHold()`, `PtUpdate()`

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon *Programmer’s Guide*
**PtReleaseTransportCtrl()**

*Release a transport control structure used with drag and drop*

**Synopsis:**

```c
void PtReleaseTransportCtrl(
    PtTransportCtrl_t *ctrl
);`

**Library:**

`ph`

**Description:**

This function releases the *PtTransportCtrl_t* structure pointed to by *ctrl*, as well as its members.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtCreateTransportCtrl()*, *PtInitDnd()*, *PtTransportCtrl_t*, *PtTransportType()*

Drag and Drop chapter of the Photon *Programmer’s Guide*
Synopsis:

```c
void PtRemoveCallback( PtWidget_t *widget,
                       unsigned long callback_type,
                       PtCallbackF_t *callback,
                       void *data );
```

Library:

`ph`

Description:

This function removes the first callback entry that matches `callback` and `data`. It removes the entry from the `callback_type` callback list that belongs to `widget`.

Some types of callback resources have special routines that you should use instead of this one:

- **Pt_CB_FILTER**
  
  `PtRemoveFilterCallback()` or
  `PtRemoveFilterCallbacks()`

- **Pt_CB_HOTKEY**

  `PtRemoveHotkeyHandler()`

- **Pt_CB_RAW**

  `PtRemoveEventHandler()` or
  `PtRemoveEventHandlers()`

The `callback` argument points to a function that takes this form:

```c
int (*callback)( PtWidget_t *, void *,
                 PtCallbackInfo_t *)
```
PtRemoveCallback()  

Examples:  

See PtAddCallback().  

Classification:  

Photon  

Safety  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:  

PtAddCallback(), PtAddCallbacks(), PtRemoveCallbacks(), PtRemoveEventHandler(), PtRemoveEventHandlers(), PtRemoveFilterCallback(), PtRemoveFilterCallbacks(), PtRemoveHotkeyHandler()  
PtCallbackInfo_t in the Photon Widget Reference  
Synopsis:

```c
void PtRemoveCallbacks(
    PtWidget_t *widget,
    unsigned long callback_type,
    PtCallback_t const *callback_defs,
    unsigned int num_callbacks);
```

Library:

ph

Description:

This function removes the first callback entries that exactly match an entry in the `callback_defs` array. It removes these entries from the `callback_type` callback list that belongs to `widget`. The `num_callbacks` argument specifies the length of the array.

Some types of callback resources have special routines that you should use instead of this one:

- **Pt_CB_FILTER**
  - `PtRemoveFilterCallback()` or
  - `PtRemoveFilterCallbacks()`

- **Pt_CB_HOTKEY**
  - `PtRemoveHotkeyHandler()`

- **Pt_CB_RAW**
  - `PtRemoveEventHandler()` or
  - `PtRemoveEventHandlers()`

Classification:

Photon
PtRemoveCallbacks()

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddCallback(), PtAddCallbacks(), PtRemoveCallback(),
PtRemoveEventHandler(), PtRemoveEventHandlers(),
PtRemoveFilterCallback(), PtRemoveFilterCallbacks(),
PtRemoveHotkeyHandler()

PtCallback_t in the Photon Widget Reference

“Callbacks” in the Managing Widgets in Application Code chapter of
the Photon Programmer’s Guide
PtRemoveData()

Remove a link from a data chain

Synopsis:

```c
int PtRemoveData( PtDataHdr_t **ptr,
                  long type,
                  long subtype );
```

Library:

ph

Description:

This function removes a link from the ptr data chain. If a remove function is provided, it’s called prior to the release of the node and data:

- If the remove function returns Pt_END, the node shouldn’t be removed, no action is taken, and PtRemoveData() returns EOK.
- If the remove function returns Pt_CONTINUE, the data is freed.
- If the remove function returns Pt_END or Pt_HALT, the data isn’t be freed here as it may have been freed by the remove function.

Returns:

-1 The data wasn’t found.

Pt_CONTINUE

The data was found and released.

Pt_HALT The data was found, the node was released, and the data was taken care of by the remove function.

Pt_END The node wasn’t removed; refused by the remove function.

Classification:

Photon
PtRemoveData()

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddData(), PtFindData(), PtFindNextData(), PtUnlinkData()
PtRemoveEventHandler()

Remove a single Pt_CB_RAW entry from a widget

Synopsis:

```c
void PtRemoveEventHandler(
    PtWidget_t *widget,
    unsigned long event_mask,
    PtCallbackF_t *callback,
    void *data);
```

Library:

ph

Description:

This function removes the first callback entry that matches `event_mask`, `callback`, and `data`. It removes the entry from the Pt_CB_RAW callback list that belongs to `widget`.

The `callback` argument points to a function that takes this form:

```c
int (*callback)(PtWidget_t *, void *, PtCallbackInfo_t *)
```

Examples:

See PtAddEventHandler().

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtRemoveEventHandler()

See also:

PtAddEventHandler(), PtAddEventHandlers(), PtRemoveCallback(),
PtRemoveCallbacks(), PtRemoveEventHandlers()
PtRemoveFilterCallback(), PtRemoveFilterCallbacks(),
PtRemoveHotkeyHandler()

PtCallbackInfo_t in the Photon Widget Reference

“Event handlers” in the Managing Widgets in Application Code
chapter of the Photon Programmer’s Guide
PtRemoveEventHandlers()
Remove several Pt_CB_RAW entries from a widget

Synopsis:

```c
void PtRemoveEventHandlers(
    PtWidget_t *widget,
    PtRawCallback_t const *callback_defs,
    int num_handlers );
```

Library:

ph

Description:

This function removes the first handler entries that exactly match an entry in the `callback_defs` array. It removes the entries from the `Pt_CB_RAW` callback list that belongs `widget`. The `num_handlers` argument specifies the length of the array.

For information about the `PtRawCallback_t` structure, see the Photon Widget Reference.

Examples:

See `PtAddEventHandler()`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtRemoveEventHandlers()

See also:

PtAddEventHandler(), PtAddEventHandlers(), PtRemoveCallback(), PtRemoveCallbacks(), PtRemoveEventHandler()
PtRemoveFilterCallback(), PtRemoveFilterCallbacks(), PtRemoveHotkeyHandler(),

PtRawCallback_t, Pt_CB_RAW in the Photon Widget Reference
**PtRemoveFilterCallback()**

Remove a single Pt_CB_FILTER entry from a widget

**Synopsis:**

```c
void PtRemoveFilterCallback(
    PtWidget_t *widget,
    unsigned long event_mask,
    PtCallbackF_t *callback,
    void *data );
```

**Library:**

ph

**Description:**

This function removes the first callback entry that matches `event_mask`, `callback`, and `data`. It removes the entry from the Pt_CB_FILTER callback list that belongs to `widget`.

The `callback` argument points to a function that takes this form:

```c
int (*callback)(PtWidget_t *, void *, PtCallbackInfo_t *)
```

**Examples:**

See `PtAddFilterCallback()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtRemoveFilterCallback()

See also:

PtAddCallback(), PtAddCallbacks(), PtAddEventHandler(),
PtAddEventHandlers(), PtAddFilterCallback(),
PtAddFilterCallbacks(), PtAddHotkeyHandler(),
PtRemoveFilterCallbacks()

PtCallbackInfo_t in the Photon Widget Reference

“Event handlers” in the Managing Widgets in Application Code
chapter of the Photon Programmer’s Guide
**PtRemoveFilterCallbacks()**

Remove several Pt_CB_FILTER entries from a widget

**Synopsis:**

```c
void PtRemoveFilterCallbacks(
    PtWidget_t *widget,
    PtRawCallback_t const *callback_defs,
    int num_handlers);
```

**Library:**

`ph`

**Description:**

This function removes the first handler entries that exactly match an entry in the `callback_defs` array. It removes the entries from the `Pt_CB_FILTER` callback list that belongs `widget`. The `num_handlers` argument specifies the length of the array.

For information about the `PtRawCallback_t` structure, see the Photon Widget Reference.

**Examples:**

See `PtAddFilterCallback()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtRemoveFilterCallbacks()

See also:

PtAddCallback(), PtAddCallbacks(), PtAddEventHandler(), PtAddEventHandlers(), PtAddFilterCallback(), PtAddFilterCallbacks(), PtAddHotkeyHandler(), PtRemoveFilterCallback()

PtRawCallback_t, Pt_CB_FILTER in the Photon Widget Reference

**PtRemoveHotkeyHandler()**

Remove a single hotkey handler entry from a widget

**Synopsis:**

```c
void PtRemoveHotkeyHandler(
    PtWidget_t *widget,
    unsigned key_sym_cap,
    unsigned key_mods,
    short flags,
    void *data,
    PtCallbackF_t *callback );
```

**Library:**

ph

**Description:**

This function removes the specified callback if the callback matches `key_sym_cap`, `key_mods`, `flags`, `data`, and `callback`. The function removes the callback from the `Pt_CB_HOTKEY` callback list that belongs to `widget`.

The `callback` argument points to a function that takes this form:

```c
int (*callback)(PtWidget_t *, void *, PtCallbackInfo_t *)
```

**Examples:**

See `PtAddHotkeyHandler()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtRemoveHotkeyHandler()**

**See also:**

- PtAddHotkeyHandler(), PtRemoveCallback(), PtRemoveCallbacks(),
- PtRemoveEventHandler(), PtRemoveEventHandlers(),
- PtRemoveFilterCallback(), PtRemoveFilterCallbacks()

**PtCallbackInfo_t** in the Photon Widget Reference

Synopsis:

```c
int PtReparentWidget( PtWidget_t *widget,
    PtWidget_t *parent );
```

Library:

ph

Description:

This function takes the specified `widget` from its current parent and gives it to the specified `parent`. The `parent` must be a container widget.

Returns:

0 Success.

-1 The widget couldn’t be reparented.

Examples:

```c
PtWidget_t *label, *window1, *window2;
/* create widget within window1 */
    ...  
label = PtCreateWidget( PtLabel, window1, 5, args );
/* use widget within window1 */
    ...  
/* reparent label to window2 */
    PtReparentWidget( label, window2 );
/* use widget within window2 */
    ...  
```

Classification:

Photon
PtReparentWidget()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtCreateWidget(), PtFindGuardian(), PtGetParent(), PtValidParent(), PtWidgetParent()

PtReRealizeWidget()

Force a widget to unrealize and then rerealize itself

Synopsis:

```c
int PtReRealizeWidget( PtWidget_t *widget );
```

Library:

ph

Description:

This function forces the specified widget and all its descendants to unrealize and rerealize themselves.

CAUTION: This function is heavy handed, so use it sparingly. As we add more advanced geometry negotiation to the widget engine, this function will become obsolete.

Returns:

0 Successful completion.

-1 An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtReRealizeWidget()

See also:

PtRealizeWidget(), PtUnrealizeWidget()
Synopsis:

```c
int PtResizeEventMsg( PtAppContext_t app,
                      int msg_size);
```

Library:

ph

Description:

This function sets the size the Photon event buffer, which receives all events on behalf of a Photon application. Using this function to increase the acceptable message size allows larger non-Photon messages to be received by the Photon application without requiring `MsgReadv()` (see the QNX Neutrino Library Reference).

`PtResizeEventMsg()` won’t reduce the message buffer beyond a certain minimum size. This is so that the widget library will continue to function.

`PtResizeEventMsg()` doesn’t actually reallocate any message buffers. It just sets the size to be used for them.

The `app` argument is the address of the application context, a structure that manages all the data associated with this application. This must be specified as NULL, so that the default context is used.

Returns:

- 0 Successful completion.
- -1 An error occurred.

Classification:

Photon
PtResizeEventMsg()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

“Receiving QNX messages” in the Interprocess Communications chapter of the Photon *Programmer’s Guide*
PtSendEventToWidget()

Give an event to a widget

Synopsis:

```c
int PtSendEventToWidget( PtWidget_t *widget, PhEvent_t *event );
```

Library:

```
ph
```

Description:

This function passes the event in the given `PhEvent_t` structure directly to the given widget for processing.

The widget expects the event to be followed immediately by the associated set of rectangles and event data. See the example below.

Returns:

The value returned by the last callback function invoked by the event, or -1 if an error occurred.

Examples:

```c
// Send a Phantom release event to a widget.
Phantom( PtWidget_t *widget, PhEvent_t *event )
{
    struct{
        PhEvent_t event;
        PhRect_t rect;
        PhPointerEvent_t pevent;
    } new_event;

    memset( &new_event.rect, -1, sizeof( new_event.rect ) );

    if( event ) {
        new_event.event = *event;
    }

    new_event.event.processing_flags = Ph_FAKE_EVENT;
    new_event.event.type = Ph_EV_BUT_RELEASE;
    new_event.event.subtype = Ph_EV_RELEASE_PHANTOM;
    new_event.pevent.click_count = 1;
}
```
PtSendEventToWidget()

```c
new_event.event.buttons = Ph_BUTTON_SELECT;
new_event.event.num_rects = 1;
PtSendEventToWidget( widget,
        (PhEvent_t *) &new_event);
```

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhEvent_t, PhEmit(), PhEmitmx(), PhPointerEvent_t

Events chapter of the Photon *Programmer's Guide*
PtSetAreaFromCanvas()

Calculate an area based on the canvas and borders of a widget

Synopsis:

```c
PhArea_t * PtSetAreaFromCanvas(
    PtWidget_t *widget,
    PhRect_t const *canvas_rect,
    PhArea_t *area);
```

Library:

```
ph
```

Description:

This function sets the `PhArea_t` structure pointed to by `area` to an area that produces a widget canvas of `canvas_rect`, given the attributes, borders, etc. of `widget`.

The `area` argument must be provided and have its own storage. This function doesn’t allocate any memory.

Returns:

The same pointer as the `area` argument, or NULL if an error occurred.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSetAreaFromCanvas()

See also:

\texttt{PhArea\_t, PhAreaToRect()}, \texttt{PhRect\_t}, \texttt{PhRectToArea()}
Synopsis:

```c
PtSetArg( PtArg_t *arg,
    long type,
    long value,
    long len );
```

Library:

ph

Description:

This macro builds argument lists to be used with `PtCreateWidget()`, `PtSetResources()`, and `PtGetResources()`.

- If the values don’t need to be calculated at runtime, you might be able to use `Pt_ARG()` instead to initialize the argument list.

- A common mistake is to think that this macro actually sets the resources. It doesn’t; be sure to call `PtCreateWidget()`, `PtSetResources()`, or `PtGetResources()`.

- If you’re setting or getting one resource, it’s easier to call `PtSetResource()` or `PtGetResource()`.

The `arg` argument is normally part of an array of `PtArg_t` data structures. The `type` argument contains the resource manifest and `value` contains the value of the argument being passed. The way the `len` argument is used depends on the resource type.

For information on getting and setting resources, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

Examples:

```c
PtArg_t args[4];
PhPoint_t pos = { 100, 100 };

/* Position the widget at (100,100) */
```
PtSetArg()

PtSetArg( &args[0], Pt_ARG_POS, &pos, 0 );
/* Make its primary color blue; in this case, blue text */
PtSetArg( &args[1], Pt_ARG_COLOR, Pg_BLUE, 0 );
/* Set the string drawn with the widget */
PtSetArg( &args[2], Pt_ARG_TEXT_STRING, "Button", 0 );
/* Place the button widget in the widget hierarchy */
PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 3, args );

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtArg_t, Pt_ARG(), PtCreateWidget(), PtGetResources(), PtSetResources()
**PtSetClassStyleMethods()**

Set multiple members of a style from an array

**Synopsis:**

```c
int PtSetClassStyleMethods(
    PtWidgetClassStyle_t *style,
    int num_methods,
    PtStyleMethods_t *meth_array);
```

**Library:**

ph

**Description:**

This function is similar to `PtSetStyleMember()`, but lets you specify an array of (manifest, value) pairs to set multiple members in a single call.

**Returns:**

0 on success, or the number of the first manifest that wasn’t successfully set.

If the return is greater than 0, subtract 1 from it to determine the index into the array of manifests. For example, if `PtSetClassStyleMethods()` returns 1, `meth_array[0]` contains the first manifest that couldn’t be set.

**Examples:**

```c
PtStyleMethods_t neon_methods[2] = {
    {Pt_STYLE_DRAW, neon_draw},
    {Pt_STYLE_SIZING, neon_sizing}
};

PtSetClassStyleMethods (neon_style, 2, neon_methods);
```
PtSetClassStyleMethods()

Classification:
Photon

Safety
<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetStyleMember(), PtSetStyleMembers(), PtSetWidgetStyle()

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**PtSetParentWidget()**

*Set the current parent widget*

**Synopsis:**

```c
PtWidget_t *PtSetParentWidget( PtWidget_t *widget );
```

**Library:**

ph

**Description:**

This function sets the current parent widget. Some widget classes (for example, windows, lists and menus) call this function when they’re created.

Widgets that belong to the PtContainer class become the current parent widget when created. If you’re creating multiple PtContainer-class widgets, you must ensure that each one is placed in the correct container.

To do this, either call PtSetParentWidget() or specify the appropriate widget in the parent argument to PtCreateWidget().

If the widget argument to PtSetParentWidget() is NULL and you then call PtCreateWidget() with a Pt_DEFAULT_PARENT parent argument, the new widget has no parent. It’s easier to create a widget with no parent by calling PtCreateWidget() with a parent argument of Pt_NO_PARENT.

**Returns:**

A pointer to the previous parent widget, or NULL if there is none.

**Examples:**

See PtCreateWidget().
**PtSetParentWidget()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtCreateWidget(), PtFindGuardian(), PtGetParent(), PtGetParentWidget(), PtReparentWidget(), PtValidParent(), PtWidgetParent()*

**PtSetResource()**

*Set one resource for a widget*

**Synopsis:**

```c
#define PtSetResource( widget, type, value, len ) ...  
```

**Library:**

`ph`

**Description:**

This function sets a resource for the specified widget. The `type` argument contains the resource manifest and `value` contains the value of the argument being passed. The way the `len` argument is used depends on the resource type.

For information on getting and setting resources, see the Manipulating Resources in Application Code chapter of the Photon Programmer’s Guide.

If the widget has been realized, changing its resources may change how it appears on the screen.

**Returns:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The given resources was applied to the widget.</td>
</tr>
<tr>
<td>-1</td>
<td>The widget wasn’t modified because it doesn’t contain the given resource or the value of the resource was the same as that already stored in the widget.</td>
</tr>
</tbody>
</table>

**Examples:**

Turn the widget blue:

```c
PtWidget_t *widget;

PtSetResource( widget, Pt_ARG_FILL_COLOR, Pg_BLUE, 0 );
```
PtSetResource()

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtArg_t, Pt_ARG(), PtGetResource(), PtGetResources(), PtSetArg(), PtSetResources()

**PtSetResources()**

*Set one or more resources for a widget*

**Synopsis:**

```c
int PtSetResources( PtWidget_t *widget,
                  int n_args,
                  PtArg_t const *args );
```

**Library:**

`ph`

**Description:**

This function sets resources for the specified widget. The `args` array indicates which resources to set, and `n_args` indicates the number of items in the `args` array. Before calling this function, you must initialize the `args` array with `PtSetArg()` or `Pt_ARG()`.

If you’re setting only one resource, it’s easier to call `PtSetResource()`.

For more information, see the Manipulating Resources in Application Code chapter of the Photon *Programmer’s Guide*.

If the widget has been realized, changing its resources may change how it appears on the screen.

**Returns:**

- **0** At least one of the given resources was applied to the widget.
- **-1** The widget wasn’t modified because it doesn’t contain the given resources or the values of the resources were the same as those already stored in the widget.

A return code of 0 doesn’t necessarily mean that all the resources were successfully set. The only way to be sure that a resource was set is to set it, then get it and compare the values.


**PtSetResources()**

**Examples:**

Turn the widget blue and highlight it:

```c
PtArg_t args[2];
PtWidget_t *widget;

PtSetArg( &args[0], Pt_ARG_FILL_COLOR, Pg_BLUE, 0 );
PtSetArg( &args[1], Pt_ARG_FLAGS,
    Pt_HIGHLIGHTED, Pt_HIGHLIGHTED );
PtSetResources( widget, 2, args );
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtArg_t, Pt_ARG(), PtGetResource(), PtGetResources(), PtSetArg(), PtSetResource()`

Synopsis:

```c
int PtSetStyleMember( PtWidgetClassStyle_t *style,
                     int manifest,
                     void *value );
```

Library:

ph

Description:

This function sets the member of the given style associated with the given manifest to the provided value.

The type of value is dictated by the manifest as follows:

**Pt_STYLE_DRAW**

The address of a function of the following form:

```c
void (*draw_f)( PtWidget_t *widget,
                PhTile_t const *damage );
```

This function is called whenever any widget that’s using this style needs to draw. The widget argument is a pointer to the widget currently rendering, and the damage argument is a list of tiles (see PhTile_t) that describes the damaged areas (the areas the widget must redraw/repair). The damage list is relative to the window.

**Pt_STYLE_EXTENT** or **Pt_STYLE_SIZING**

The address of a function of the following form:

```c
void (*sizing_f)( PtWidget_t *widget );
```

This function is called whenever a widget that’s using this style is moved, resized, or modified in some fashion that may require the widget to move or resize (change in widget data). This function is responsible for setting the widget’s dimension to the appropriate values.
PtSetStyleMember()

Pt_STYLE_ACTIVATE

The address of a function of the following form:

```c
void (*activate_f)( PtWidget_t *widget,
                   PtWidgetClassStyle_t *old_style );
```

This function is called whenever a widget is created that defaults to this style, and whenever a widget’s style is changed from some other style to this one. This function is the place to put manipulation of a widget’s control surfaces, the addition of callbacks, or the setting of resources (to override widget’s defaults).

Pt_STYLE_CALC_BORDER

The address of a function of the following form:

```c
void (*calc_border_f)( PtWidget_t const *widget,
                       PhRect_t *border );
```

This function is responsible for reporting how much space is required to render the widget’s edge decorations and margins. The `border` rectangle represents the distances to be reserved on each edge for this purpose. These amounts are subtracted from the widget’s extent to determine the widget’s canvas. For example, a border of `{5,5,5,5}` has a border of 5 pixels all the way around.

Pt_STYLE_CALC_OPAQUE

The address of a function of the following form:

```c
void (*calc_opaque_f)( PtWidget_t *widget );
```

This function is responsible for calculating the list of tiles that represents the opaque areas of a widget. This list is used to determine what needs to be damaged below this widget when it’s modified.

Pt_STYLE_DEACTIVATE

The address of a function of the following form:
void (*deactivate_f)( PtWidget_t *widget,
PtWidgetClassStyle_t *new_style );

This method is called whenever a widget using this style is
either being destroyed or is switching to a different style. If
switching to a different style, the style being switched to is
defined by new_style. If new_style isn’t NULL, this function
must remove any callbacks that were added in the
Pt_STYLE_ACTIVATE method.

Pt_STYLE_NAME
A char *. This is the name of the style.

Pt_STYLE_DATA
A pointer to an arbitrary data block for the style’s use.

Returns:

0 Success.

-1 The manifest wasn’t valid.

Examples:
PtSetStyleMember (neon_style, Pt_STYLE_DRAW, neon_draw);
PtSetStyleMember (neon_style, Pt_STYLE_SIZING, neon_sizing);

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSetStyleMember()

See also:

PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMembers(), PtSetWidgetStyle()

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**PtSetStyleMembers()**

Set multiple members of a style from a variable-length argument list

**Synopsis:**

```c
int PtSetStyleMembers (
    PtWidgetClassStyle_t *style, ...);
```

**Library:**

ph

**Description:**

This function is similar to *PtSetStyleMember()*, but lets you specify multiple (manifest, value) pairs at once.

---

The last (manifest, value) pair must be -1, -1.

---

**Returns:**

0 on success, or the number of manifests that were successfully set.

---

If the return is greater than 0, it will be less than the number of manifests provided, because at least one of the manifests couldn’t be set.

---

**Examples:**

```c
PtSetStyleMembers( neon_style,
    Pt_STYLE_DRAW, neon_draw,
    Pt_STYLE_SIZING, neon_sizing,
    -1, -1 );
```

**Classification:**

Photon
PtSetStyleMembers()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetWidgetStyle()

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**Synopsis:**

```
int PtSetWidgetStyle( PtWidget_t *widget,
                     char *name);
```

**Library:**

```
ph
```

**Description:**

This function causes the provided widget to use the style called `name` from its widget class’s style set.

If there isn’t a style called `name`, this function creates a duplicate of the default style and calls the copy `name`. The widget then references this new style. If at a later time you modify the style, all widgets referencing it resize and redraw themselves.

You can also set the style for a widget instance by setting its `Pt_ARG_STYLE` resource (defined by `PtBasic`). Setting this resource has the same effect as calling `PtSetWidgetStyle()`.

**Returns:**

The index of the style selected.

**Examples:**


**Classification:**

Photon
PtSetWidgetStyle()

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddClassStyle(), PtCreateClassStyle(), PtDupClassStyle(), PtFindClassStyle(), PtGetStyleMember(), PtGetWidgetStyle(), PtSetClassStyleMethods(), PtSetStyleMember(), PtSetStyleMembers()

Pt_ARG_STYLE resource of PtBasic in the Photon Widget Reference

**PtShowSurface(), PtShowSurfaceById()**
Show a hidden control surface

**Synopsis:**

```c
int PtShowSurface( PtWidget_t *widget,
                   PtSurface_t *surface );

int PtShowSurfaceById( PtWidget_t *widget,
                        uchar_t surface_id );
```

**Library:**

`ph`

**Description:**

These functions show a control surface, restoring it from a hidden state. The `widget` argument specifies the widget owning the surface. The functions differ in how they identify the control surface:

- **PtShowSurface()**
  - Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be `NULL`.

- **PtShowSurfaceById()**
  - Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

**Returns:**

- `0` Success.
- `-1` The specified surface couldn’t be found or wasn’t hidden.

**Classification:**

Photon
**PtShowSurface(), PtShowSurfaceById()**

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtHideSurface(), PtHideSurfaceByAction(), PtHideSurfaceById(), PtShowSurfaceByAction(), PtSurfaceIsHidden(), PtSurfaceIsShown()*

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtShowSurfaceByAction()**

*Show all hidden control surfaces associated with an action*

**Synopsis:**

```c
int PtShowSurfaceByAction(
    PtWidget_t *widget,
    PtWidgetClassRef_t const *cref,
    ushort_t action_id);
```

**Library:**

*ph*

**Description:**

This function shows all hidden control surfaces associated with an action. The `widget` argument specifies the widget owning the surfaces, while `cref` and `action_id` specify the class and manifest of the action associated with surfaces to be shown.

**Returns:**

0 if any surfaces were shown by this function, or -1 if no surfaces were affected.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtHideSurface()`, `PtHideSurfaceByAction()`, `PtHideSurfaceById()`, `PtShowSurface()`, `PtShowSurfaceByAction()`, `PtShowSurfaceById()`, `PtSurfaceIsHidden()`, `PtSurfaceIsShown()`
PtShowSurfaceByAction()

Control Surfaces chapter of the Photon Programer’s Guide
PtSignalProcF_t, PtSignalProc_t
Type for defining a signal-handling function

Synopsis:

typedef int PtSignalProcF_t( int, void *);
typedef PtSignalProcF_t *PtSignalProc_t;

Description:

These data types define pointers to signal-handling functions. The
PtSignalProcF_t type is the function type that the
PtSignalProc_t type points to. This allows you to do something
like this:

PtSignalProcF_t my_signal_proc;
int my_signal_proc( int, void * ) {
    ...
}

The compiler should detect any inconsistencies between the two
declarations of my_signal_proc() and give you an error (which is
better than a “pointer mismatch” warning on the call to
PtAppAddSignalProc()).

Classification:

Photon

See also:

PtAppAddSignalProc(), PtAppRemoveSignal()

Interprocess Communication in the Photon Programmer’s Guide
**PtSpawn()**

*Spawn a new process*

**Synopsis:**

```c
pid_t PtSpawn( const char *cmd,
               const char * const *argv,
               const char * const *env,
               const PtSpawnOptions_t *opt,
               PtSpawnCbF_t *cb,
               void *data,
               PtSpawnCbId_t **csp );
```

**Arguments:**

- `cmd` The program to be started. If it doesn’t contain a slash, directories listed in the `PATH` environment variable are searched.
- `argv` A pointer to an argument vector. The last member of `argv` must be a NULL pointer.
- `env` Environment variables for the new process. If it’s NULL, the value of the global variable `extern char **environ` is used.
- `opt` A pointer to a `PtSpawnOptions_t` structure that can be used to specify some extra details of how the child should be spawned; see “Options,” below.
- `cb, data` A callback to be called after the child process terminates, and data to pass to the callback. See “Callback function,” below.
- `csp` If non-NULL, the function stores in `*csp` a pointer to a control structure that can be later used to change or remove the callback. See `PtSpawnSetCallback()` and `PtSpawnDeleteCallback()`.
This control structure exists only until the termination callback is called; don’t call `PtSpawnSetCallback()` or `PtSpawnDeleteCallback()` after the callback has been called.

**Library:**

`ph`

**Description:**

This function spawns a new process and optionally installs a callback that’s called when the child process terminates.

**Options**

Under QNX Neutrino, `PtSpawnOptions_t` consists of:

- `iov` An fd-redirection array.
- `options` A structure of type `inheritance` (see `spawn()` in the QNX Neutrino Library Reference).

If `opt` is NULL, the function uses the defaults specified in:

```c
extern const PtSpawnOptions_t PtSpawnDefaults;
```

By default, the new process inherits all of the parent’s valid file descriptors whose values are less than or equal to 9.

If you want to specify a non-NULL value for `opt`, it’s a good idea to modify a copy of the default structure. For example:

```c
PtSpawnOptions_t my_opts;
my_opts = PtSpawnDefaults;
my_opts.iov[1] = fd; // Redirect stdout
```
PtSpawn()

Callback function

PtSpawnCbF_t is a function type:

typedef void PtSpawnCbF_t( void *data,
int status );

If cb isn’t NULL, PtSpawn() attaches a signal handler for SIGCHLD that calls waitpid() to determine whether the child process has terminated. If waitpid() succeeds, the function specified by cb is called, and the signal handler is removed.

If cb is NULL, PtSpawn() doesn’t attach any signal handlers or call waitpid().

If you don’t need a callback but you also don’t want to have to worry about zombie processes, specify cb as PtSpawnNoCb — it’s an empty callback function defined in the library.

If cb is NULL but csp isn’t, no callback is attached, and *csp is set to NULL.

Returns:

The process ID of the spawned process, or -1 on error.

Errors:

See spawn() in the QNX Neutrino Library Reference.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PtSpawnSetCallback(), PtSpawnDeleteCallback(), PtSpawnWait()
**PtSpawnDeleteCallback()**

Remove a child-termination callback

**Synopsis:**

```c
void PtSpawnDeleteCallback( PtSpawnCbId_t *cs );
```

**Library:**

```
ph
```

**Description:**

This function can be used to remove a callback function for a child process created by a previous call to `PtSpawn()`. The `cs` argument is the control structure created by that call to `PtSpawn()` and returned via the `csp` argument.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtSpawn()`, `PtSpawnSetCallback()`, `PtSpawnWait()`
PtSpawnSetCallback()

Change the callback in a PtSpawn() control structure

Synopsis:

```c
void PtSpawnSetCallback( PtSpawnCbId_t *cs,
                        PtSpawnCbF_t *cb,
                        void *data );
```

Library:

ph

Description:

This function can be used to specify a new callback function to be called when a child process created by a previous call to PtSpawn() terminates. The cs argument is the control structure created by that call to PtSpawn() and returned via the csp argument.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtSpawn(), PtSpawnDeleteCallback(), PtSpawnWait()
**PtSpawnWait()**

Spawn a process and wait for its termination

**Synopsis:**

```c
int PtSpawnWait( const char *cmd,
                  const char **argv,
                  const char **env,
                  const PtSpawnOptions_t *opt,
                  pid_t *pidp );
```

**Library:**

ph

**Description:**

This function spawns a new process and waits for its termination. While the child process is running, Photon events are processed.

If *pidp* isn’t NULL, the process ID of the spawned command is stored in *pidp*. This can be used if callback functions need to communicate with the running child process.

The meaning of all the other arguments is the same as for the PtSpawn() function.

**Returns:**

The termination status of the child (see waitpid()), or -1 if the child process couldn’t be started.

**Classification:**

Photon

---

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PtSpawn(), PtSpawnSetCallback(), PtSpawnDeleteCallback()
PtStartFlux()
Prevent repairs to a container widget and its children

Synopsis:

```c
int PtStartFlux( PtWidget_t *container );
```

Library:

```
ph
```

Description:

This function increments the flux count for the given container, to prevent repairs to the specified container and to all its children. You can still modify the widgets, but the damage to them isn’t recorded.

You typically use this function when you’re about to make a lot of changes to the container and its children, and you don’t want to update the display until you’re done. When you want repairs to be made, call `PtEndFlux()`.

Returns:

The container’s new flux count, or -1 if an error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtEndFlux()`, `PtIsFluxing()`

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon Programmer’s Guide
**PtSurfaceActionId()**

*Get the action ID for a surface*

**Synopsis:**

```c
int PtSurfaceActionId( PtSurface_t *surface );
```

**Library:**

`ph`

**Description:**

This macro retrieves the numeric action ID associated with the provided surface.

**Returns:**

The action ID, or -1 if the provided surface isn’t an action surface.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PtSurfaceId()*

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceAddData()**, **PtSurfaceAddDataById()**

Add data to a control surface

**Synopsis:**

```c
int PtSurfaceAddData( PtWidget_t *widget,
                      PtSurface_t *surface,
                      void *data,
                      long len );

int PtSurfaceAddDataById( PtWidget_t *widget,
                          uchar_t surface_id,
                          void *data,
                          long len );
```

**Library:**

ph

**Description:**

These functions attach data to a control surface belonging to the given widget. They differ in how they identify the control surface:

*PtSurfaceAddData()*

Uses the *surface* argument, which points to a *PtSurface_t* structure that describes the control surface. This pointer must not be NULL.

*PtSurfaceAddDataById()*

Searches the control surfaces belonging to the widget for the one with an ID of *surface_id*.

The *data* arguments points to the data to attach. This data can be anything you need to store with the surface. Any data previously added to this surface is removed and overwritten.

The *len* argument specifies the length (in bytes) of the data. If *len* is nonzero, then *len* bytes of data are copied from the data pointer, and that copy is attached to the surface. This copy is freed when the data is removed. If *len* is 0, no copying is performed, and no deallocation takes place when the data is removed.
**PtSurfaceAddData(), PtSurfaceAddDataById()**

**Returns:**

0  Success.

-1  The operation failed due to a lack of memory, or the specified surface couldn’t be found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtSurfaceGetData(), PtSurfaceGetDataById(), PtSurfaceRemoveData(), PtSurfaceRemoveDataById()*

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceBrotherBehind()**

*Get the control surface behind a given one*

**Synopsis:**

```c
PtSurface_t *PtSurfaceBrotherBehind(
    PtWidget_t *widget,
    PtSurface_t *surface);
```

**Library:**

ph

**Description:**

This function gets the control surface behind the surface described by `surface`.

**Returns:**

A pointer to the `PtSurface_t` structure describing the control surface behind, or NULL if there isn’t one.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtInsertSurface(), PtInsertSurfaceById(), PtSurfaceInBack(), PtSurfaceInFront(), PtSurfaceBrotherInFront(), PtSurfaceToBack(), PtSurfaceToBackById(), PtSurfaceToFront(), PtSurfaceToFrontById()*

Control Surfaces chapter of the Photon *Programmer’s Guide*
Synopsis:

```
PtSurface_t *PtSurfaceBrotherInFront(
    PtWidget_t *widget,
    PtSurface_t *surface
);
```

Library:

ph

Description:

This function gets the control surface in front of the surface described by `surface`.

Returns:

A pointer to the `PtSurface_t` structure describing the control surface in front, or NULL if there isn’t one.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtInsertSurface()`, `PtInsertSurfaceById()`, `PtSurfaceBrotherBehind()`, `PtSurfaceInBack()`, `PtSurfaceInFront()`, `PtSurfaceToBack()`, `PtSurfaceToBackById()`, `PtSurfaceToFront()`, `PtSurfaceToFrontById()`

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById()**

*Calculate the bounding box for a control surface*

**Synopsis:**

```c
PhRect_t *PtSurfaceCalcBoundingBox(
    PtSurface_t *surface);

PhRect_t *PtSurfaceCalcBoundingBoxById(
    PtWidget_t *widget,
    uchar_t surface_id);
```

**Library:**

`ph`

**Description:**

These functions calculate the bounding box for a control surface. They differ in the way they identify the control surface:

- **PtSurfaceCalcBoundingBox()** uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

- **PtSurfaceCalcBoundingBoxById()** searches the control surfaces belonging to the given widget for the one with an ID of `surface_id`.

You should call one of these functions whenever you modify the points defining a surface. Although these functions are mainly targeted as a convenience function for polygonal surfaces, they also ensure that the upper-left/lower-right corners of a rectangular or elliptical surface never get inverted.

**Returns:**

A pointer to a `PhRect_t` structure that defines the bounding rectangle of the surface. `PtSurfaceCalcBoundingBoxById()` returns NULL if it couldn’t find the control surface with the given ID.
PtSurfaceCalcBoundingBox(),
PtSurfaceCalcBoundingBoxById()

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t, PtCalcSurface(), PtCalcSurfaceByAction(), PtCalcSurfaceById(), PtSurfaceExtent(), PtSurfaceExtentById(), PtSurfaceRect(), PtSurfaceRectById()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtSurfaceExtent(), PtSurfaceExtentById()**

*Calculate the extent of a control surface*

**Synopsis:**

```c
PhRect_t *PtSurfaceExtent(
    PtWidget_t *widget,
    PtSurface_t const *surface,
    PhRect_t *extent);

PhRect_t *PtSurfaceExtentById(
    PtWidget_t *widget,
    uchar_t surface_id,
    PhRect_t *extent);
```

**Library:**

`ph`

**Description:**

These functions calculate the extent of a control surface belonging to a given widget. They differ in how they identify the control surface:

*PtSurfaceExtent()*

Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

*PtSurfaceExtentById()*

Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

The `extent` argument points to a `PhRect_t` structure in which the result is stored.

**Returns:**

The same pointer as `extent`. `PtSurfaceExtentById()` returns NULL if it can’t find the control surface with the given ID.
PtSurfaceExtent(), PtSurfaceExtentById()

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t, PtCalcSurface(), PtCalcSurfaceByAction(), PtCalcSurfaceById(), PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById(), PtSurfaceRect(), PtSurfaceRectById()

Control Surfaces chapter of the Photon Programmer’s Guide
**PtSurfaceGetData()**

*Get data associated with a control surface*

**Synopsis:**

```c
void *PtSurfaceGetData( PtWidget_t *widget,
                         PtSurface_t *surface );

void *PtSurfaceGetDataById( PtWidget_t *widget,
                               uchar_t surface_id );
```

**Library:**

ph

**Description:**

These functions retrieve the user data stored with a control surface belonging to the given widget. They differ in the way they identify the control surface:

- **PtSurfaceGetData()**
  
  Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

- **PtSurfaceGetDataById()**
  
  Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

**Returns:**

A pointer to the data, or NULL if either the specified surface couldn’t be found, or it doesn’t have any data attached to it.

**Classification:**

Photon
PtSurfaceGetData()
**PtSurfaceHit()**

*Find the control surface hit by a given point*

**Synopsis:**

```c
PtSurface_t *PtSurfaceHit(
    PtWidget_t *widget,
    PhPoint_t const *point,
    ulong_t event_mask,
    PtSurface_t const *surface);
```

**Library:**

```
ph
```

**Description:**

`PtSurfaceHit()` finds the control surface, belonging to the given widget, at the coordinates pointed to by `point`.

The `event_mask` specifies the event types to which the control surface must be sensitive. A value of 0 means any event types.

The `surface` argument, if not NULL is the control surface to test first.

**Returns:**

A pointer to the control surface hit by the given point, or NULL if there isn’t one or an error occurred.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

\texttt{PhPoint\_t, PtSurfaceCalcBoundingBox(), PtSurfaceRect(), PtSurfaceTestPoint()}

Control Surfaces chapter of the Photon \textit{ Programmer’s Guide}
**PtSurfaceId()**
*Get the ID of a control surface*

**Synopsis:**

```c
uchar_t PtSurfaceId( PtSurface_t *surface);
```

**Library:**

`ph`

**Description:**

This macro retrieves the numeric ID from a control surface structure.

**Returns:**

The surface ID.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtSurfaceActionId()*

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceInBack()**

*Get the backmost control surface belonging to a widget*

**Synopsis:**

```c
PtSurface_t *PtSurfaceInBack( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This function gets the backmost control surface belonging to the given widget.

**Returns:**

A pointer to the `PtSurface_t` structure describing the control surface, or NULL if the widget doesn’t have any control surfaces.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtInsertSurface()`, `PtInsertSurfaceById()`, `PtSurfaceBrotherBehind()`, `PtSurfaceBrotherInFront()`, `PtSurfaceInFront()`, `PtSurfaceToBack()`, `PtSurfaceToBackById()`, `PtSurfaceToFront()`, `PtSurfaceToFrontById()`

Control Surfaces chapter of the Photon *Programmer’s Guide*
PtSurfaceInFront()
Get the frontmost control surface belonging to a widget

Synopsis:

```
PtSurface_t *PtSurfaceInFront( PtWidget_t *widget );
```

Library:

ph

Description:

This function gets the frontmost control surface belonging to the
given widget.

Returns:

A pointer to the PtSurface_t structure describing the control
surface, or NULL if the widget doesn’t have any control surfaces.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtInsertSurface(), PtInsertSurfaceById(), PtSurfaceBrotherBehind(), PtSurfaceBrotherInFront(), PtSurfaceInBack(), PtSurfaceToBack(), PtSurfaceToBackById(), PtSurfaceToFront(), PtSurfaceToFrontById()

Control Surfaces chapter of the Photon Programmer’s Guide
PtSurfaceIsDisabled()

Synopsis:

```c
int PtSurfaceIsDisabled( PtSurface_t *surface );
```

Library:

`ph`

Description:

This macro evaluates to nonzero if the provided surface is currently disabled. Otherwise, it evaluates to 0.

Returns:

Nonzero if the surface is disabled, 0 if it’s enabled.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtDisableSurface()`, `PtDisableSurfaceByAction()`, `PtEnableSurface()`, `PtEnableSurfaceByAction()`, `PtSurfaceIsEnabled()`

Control Surfaces chapter of the Photon Programmer’s Guide
**PtSurfaceIsEnabled()**

*Determine if a control surface is enabled*

**Synopsis:**

```c
int PtSurfaceIsEnabled( PtSurface_t *surface);
```

**Library:**

`ph`

**Description:**

This macro evaluates to nonzero if the provided surface is currently enabled. Otherwise, it evaluates to 0.

**Returns:**

Nonzero if the surface is enabled, 0 if it’s disabled.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtDisableSurface()`, `PtDisableSurfaceByAction()`, `PtEnableSurface()`, `PtEnableSurfaceByAction()`, `PtSurfaceIsDisabled()`

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceIsHidden()**

*Determine if a control surface is hidden*

**Synopsis:**

```c
int PtSurfaceIsHidden( PtSurface_t *surface );
```

**Library:**

`ph`

**Description:**

This macro evaluates to nonzero if the provided surface is hidden. Otherwise, it evaluates to 0.

**Returns:**

Nonzero if the surface is hidden, 0 if it’s shown.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtHideSurface()`, `PtHideSurfaceByAction()`, `PtShowSurface()`, `PtShowSurfaceByAction()`, `PtSurfaceIsShown()`, `Control Surfaces` chapter of the Photon *Programmer’s Guide*
PtSurfaceIsShown()
Determine if a control surface is shown

Synopsis:

```c
int PtSurfaceIsShown( PtSurface_t *surface );
```

Arguments:

- `surface`: A pointer to a `PtSurface_t` structure that describes the control surface.

Library:

- `ph`

Description:

This macro evaluates to nonzero if the provided surface is being shown. Otherwise, it evaluates to 0.

Returns:

Nonzero if the surface is shown, 0 if it’s hidden.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

- `PtHideSurface()`, `PtHideSurfaceByAction()`, `PtShowSurface()`, `PtShowSurfaceByAction()`, `PtSurfaceIsHidden()`
- Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceRect(), PtSurfaceRectById()**

*Get the bounding box of a control surface*

**Synopsis:**

```c
PhRect_t *PtSurfaceRect( PtSurface_t *surface,
                        PhRect_t *rect );

PhRect_t *PtSurfaceRectById( PtWidget_t *widget,
                             uchar_t surface_id,
                             PhRect_t *rect );
```

**Library:**

ph

**Description:**

These functions retrieve the bounding box of a control surface. They differ in the way they identify the control surface:

- **PtSurfaceRect()**
  
  Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

- **PtSurfaceRectById()**
  
  Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

**Returns:**

If `rect` is non-NULL, the result is copied there and the functions return `rect`. Otherwise, a pointer to the surface’s internal data is returned.

**Classification:**

Photon
PtSurfaceRect(), PtSurfaceRectById()

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Caveats:

PtSurfaceRect() is a macro.

See also:

PhRect_t, PtCalcSurface(), PtCalcSurfaceByAction(), PtCalcSurfaceById(), PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById(), PtSurfaceExtent(), PtSurfaceExtentById()

Control Surfaces chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PtSurfaceRemoveData( PtWidget_t *widget,
                         PtSurface_t *surface );

int PtSurfaceRemoveDataById( PtWidget_t *widget,
                             uchar_t surface_id );
```

Library:

ph

Description:

These functions remove user data from a control surface belonging to the given widget. They differ in the way they identify the control surface:

- **PtSurfaceRemoveData()**
  
  Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be NULL.

- **PtSurfaceRemoveDataById()**
  
  Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

If, when the data was added, the `len` argument was nonzero, the copy of the data that was made is automatically freed, otherwise no deallocation takes place.

Data is automatically removed when a control surface is destroyed.

Returns:

- `-1` The control surface or data wasn’t found.

  `Pt_CONTINUE`

  The data was found and released.
**PtSurfaceRemoveData(), PtSurfaceRemoveDataById()**

- **Pt_HALT**: The data was found, the node was released, and the data was taken care of by the remove function.
- **Pt_END**: The node wasn’t removed; refused by the remove function.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PtSurfaceAddData()`, `PtSurfaceAddDataById()`, `PtSurfaceGetData()`, `PtSurfaceGetDataById()`
- Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceTestPoint()**

*Test whether or not a point is inside a control surface*

**Synopsis:**

```c
int PtSurfaceTestPoint( PtSurface_t const *surface,
                        PhPoint_t const *point );
```

**Library:**

ph

**Description:**

This function determines whether or not the given point is inside the control surface described by the structure pointed to by *surface*.

**Returns:**

A nonzero value if the point is inside the control surface, or zero if it isn’t.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhPoint_t, PtSurfaceCalcBoundingBox(), PtSurfaceHit(), PtSurfaceRect()

Control Surfaces chapter of the Photon *Programmer’s Guide*
PtSurfaceToBack(), PtSurfaceToBackById()

Move a control surface behind all other control surfaces belonging to a widget

Synopsis:

```c
int PtSurfaceToBack( PtWidget_t *widget,
                     PtSurface_t *surface );
int PtSurfaceToBackById( PtWidget_t *widget,
                         uchar_t surface_id );
```

Library:

ph

Description:

These functions move a control surface behind all other control surfaces belonging to the given widget. They differ in how they identify the control surface:

*PtSurfaceToBack()*

Uses the *surface* argument, which points to a *PtSurface_t* structure that describes the control surface. This pointer must not be NULL.

*PtSurfaceToBackById()*

Searches the control surfaces belonging to the widget for the one with an ID of *surface_id*.

Returns:

0 Success.

-1 The specified surface couldn’t be found.

Classification:

Photon
PtSurfaceToBack(), PtSurfaceToBackByld()

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PtInsertSurface(), PtInsertSurfaceByld(), PtSurfaceBrotherBehind(), PtSurfaceBrotherInFront(), PtSurfaceInBack(), PtSurfaceInFront(), PtSurfaceToFront(), PtSurfaceToFrontByld()

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSurfaceToFront(), PtSurfaceToFrontById()**

*Move a control surface in front of all other control surfaces belonging to a widget*

**Synopsis:**

```c
int PtSurfaceToFront( PtWidget_t *widget,
                   PtSurface_t *surface );

int PtSurfaceToFrontById( PtWidget_t *widget,
                          uchar_t surface_id );
```

**Library:**

`ph`

**Description:**

These functions move a surface in front of all other surfaces belonging to the given widget. They differ in how they identify the control surface:

- **PtSurfaceToFront()**
  - Uses the `surface` argument, which points to a `PtSurface_t` structure that describes the control surface. This pointer must not be `NULL`.

- **PtSurfaceToFrontById()**
  - Searches the control surfaces belonging to the widget for the one with an ID of `surface_id`.

**Returns:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>-1</td>
<td>The specified surface couldn’t be found.</td>
</tr>
</tbody>
</table>

**Classification:**

Photon
**PtSurfaceToFront()**, **PtSurfaceToFrontById()**

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtInsertSurface*, *PtInsertSurfaceById*, *PtSurfaceBrotherBehind*, *PtSurfaceBrotherInFront*, *PtSurfaceInBack*, *PtSurfaceInFront*, *PtSurfaceToBack*, *PtSurfaceToBackById*

Control Surfaces chapter of the Photon *Programmer’s Guide*
**PtSyncWidget()**

Synchronize widget

**Synopsis:**

```c
int PtSyncWidget( PtWidget_t *widget );
```

**Library:**

ph

**Description:**

This function ensures that all flagged actions (such as a complete rebuild, a resize including extending, and any required region changes) are performed on the specified `widget`.

**Returns:**

0 Success.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtAppAddInput()`, `PtAppAddWorkProc()`, `PtFlush()`, `PtUpdate()`
PtTimerArm()

Arm a timer event on a widget

Synopsis:

```c
int PtTimerArm( PtWidget_t *widget,
               unsigned msec );
```

Library:

ph

Description:

This function arms a timer event to be trigger after `msec` milliseconds. When the timer event is triggered, it’s sent to the widget specified by `widget`. For this widget to receive the event, it must provide a raw callback of type Ph_EV_TIMER.

Any pending timers for a widget are removed automatically when the widget is unrealized.

You typically use this routine when you’re building custom widgets. Some widgets (such as PtTerminal) already use this type of timer, so calling PtTimerArm() may have unexpected results.

To disarm any timers that might be pending for the widget, call PtTimerArm() with `msec` set to 0.

Returns:

0 Successful completion.

-1 An error occurred.

Classification:

Photon
**PtTimerArm()**

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

No

See also:

- `PhTimerArm()`, `RtTimerCreate()`, `RtTimerDelete()`, `RtTimerGetTime()`, `RtTimerSetTime()`

“Timers” in the Working with Code chapter of the Photon Programmer’s Guide
PtTransportCtrl_t
Transport-control structure used in a drag-and-drop operation

Synopsis:

typedef struct ptdatatransctrl PtTransportCtrl_t;

struct ptdatatransctrl {
PtTransportHdr_t hdr;
PhTransportCtrl_t *ctrl;
PtWidget_t *widget;
PtRequestables_t *requestables;
PtRequestedLink_t *requested;
PhTransportLink_t *response;
PhTransportLink_t *last_response;
PtConnector_t *connector;
PtConnectionServer_t *connection;
int (*complete)(PtTransportCtrl_t *tctrl,
    PhTransportCtrl_t *ctrl,
    short unsigned event_subtype);
    void *data;
};

Description:

The PtTransportCtrl_t structure is used with the Photon transport mechanism in a drag-and-drop operation. Your application can pack multiple pieces of data into a single control structure and transport them simultaneously. Data can be packed inline or can be requestable.

The PtTransportCtrl_t structure includes:

hdr Information about the connection and the size of the structure.

ctrl A pointer to the low-level transport control structure. For more information, see PhTransportCtrl_t.

widget A pointer to the widget that’s currently involved in a drag-and-drop operation, or NULL if there’s no operation in progress.

requestables A list of descriptions of requestable data.
PtTransportCtrl_t

requested A list of data that the destination of the operation has requested.

response A queue of response data (structures of type PhTransportLink_t) that can satisfy the destination’s requests. The source of the operation builds this list by calling PtAddResponseType().

last_response A pointer to the last addition to the response queue.

connector A pointer to the connector. For more information, see PtConnectorCreate().

connection A pointer to the connection structure, if a drag-and-drop operation is in progress and the destination has connected.

complete A function to be called when all the drag-and-drop transactions (for a single drag-and-drop operation) are complete.

data A pointer to the inline data being dragged.

Classification:
Photon

See also:
PhTransportCtrl_t, PhTransportLink_t, PtAddResponseType(), PtCreateTransportCtrl(), PtInitDnd(), PtReleaseTransportCtrl(), PtTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
Synopsis:

PtRequestables_t *
PtTransportRequestable(
    PtTransportCtrl_t *ctrl,
    char const * const type,
    char const * const desc,
    int unsigned const flags,
    int unsigned transport,
    PtTransportReqDataCB_t *rq_callback,
    void *rq_callback_data);

Arguments:

ctrl  A pointer to the PtTransportCtrl_t control structure for the drag-and-drop operation.

type  A descriptive type name, such as image, text, filename, or files. This is simply added to the header for the packed data.

desc  The specifics of what’s in the data. The extractor uses a regular-expression match against the description to determine if the data should be unpacked or discarded. This is simply added to the header for the packed data.

flags  Flags that affect the operation:

- Ph_DONT_COPY — refer to the data to be transported instead of physically copying the data to the transport control’s stream buffers.

If this bit is set, any modifications to the data that occur between packing and actual transport of the data will be reflected in the data transported.

transport  The available transport types that can be specified when requesting data from the source:
**PtTransportRequestable()**

Ph_TRANSPORT_INLINE
The data being transported is in memory and can be unpacked immediately.

Ph_TRANSPORT_FILEREF
The data being transported is in the temporary file(s) named in the inlined data.

Ph_TRANSPORT_SHMEM
The data being transported is in the temporary shared object(s) named in the inlined data.

Ph_TRANSPORT_STREAM
The data being transported will be inlined a small piece at a time.

Ph_TRANSPORT_NAMED_STREAM
The data being transported will be inlined a small piece at a time. The streamed data is named so multiple streams of data can be transferred serially.

Ph_TRANSPORT_FILE_STREAM
The contents of files streamed using extended named streams. This is like the named stream but with extra information with each data block, including file information and so on. The requester of data must choose one of the available request transport types when requesting delivery of additional data.

rq_callback A pointer (of type PtTransportReqDataCB_t *) to the callback routine that will provide the data if the destination of the drag-and-drop operation asks for requestable data. For more information, see below.

This can be NULL if the source widget calls PtAddResponseType() to pack the requestable data before initiating the drag-and-drop operation.
PtTransportRequestable()

rq_callback_data
A pointer to arbitrary data for the rq_callback function. This pointer is stored in the
rq_callback_data member of the
PtRequestables_t structure that’s passed to the callback.

Library:

ph

Description:

This function adds an entry for requestable data to the data being packed up for a drag-and-drop operation. It’s called by the source widget when it’s starting the operation.

Request callback function

The callback function specified by the rq_callback argument has the following prototype:

```c
int unsigned rq_callback ( int unsigned type,
                          PtReqResponseHdr_t *req_hdr,
                          PtRequestables_t *requestables );
```

The arguments are:

**type**
Currently, this is Pt_DND_REQUEST_DATA, which indicates that data is being requested.

**req_hdr**
A pointer to a PtReqResponseHdr_t structure that provides useful information and tracks streams; see below.

**requestables**
A pointer to a description of the requestable type. The callback should use this when it calls PtAddResponseType().

The callback function should return:
PtTransportRequestable()

Pt_CONTINUE
The data has been or will be provided.

Pt_END The source is canceling the destination’s request.

PtReqResponseHdr_t structure
The PtReqResponseHdr_t data structure determines how a data request is responded to. It’s defined as:

typedef struct req_resp_msg_hdr PtReqResponseHdr_t;

struct req_resp_msg_hdr {
  int unsigned transport;
  int unsigned cmd;
  int unsigned chunk_size;
  int total_size;
  int unsigned byte_offset;
  void *source_handle;
  void *dest_handle;
  int unsigned data_size;
};

This structure’s members can be used in the callback function as described below:

transport If the Ph_TRANSPORT_ENDIAN_OK bit is set in this member, then the source and destination are of the same endianness. For example:

if (! (req_hdr->transport & Ph_TRANSPORT_ENDIAN_OK))
  // Destination is a different endian

cmd The action to take. If the callback’s type argument is Pt_DND_REQUEST_DATA, cmd is Pt_DND_CMD_PROVIDE_DATA.

chunk_size The size of the data that the client will accept. If this is too small, return Pt_END from the callback to make the request fail.
PtTransportRequestable()

- **total_size**: The size of data to be sent to the destination. This is equal to `chunk_size` if you’re not streaming.

- **byte_offset**: The offset into the data from which to start sending data. This is useful for streams of data.

- **source_handle**: An identifier for the source that’s passed back in this member if more than one transmission is required to provide the requested data. The destination shouldn’t modify this member.

- **dest_handle**: An identifier for the destination that’s passed back in this member if more than one transmission is required to provide the requested data. The source shouldn’t modify this member.

- **data_size**: If you’re responding with packed data, this is the packed size, not the unpacked size.

**Returns:**

- 0  Success.
- -1  An error occurred; `errno` is set.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtTransportRequestable()

See also:

PtAddResponseType(), PtCreateTransportCtrl(), PtInitDnd(), PtTransportCtrl_t, PtTransportType()

Drag and Drop chapter of the Photon Programmer’s Guide
Synopsis:

```c
int PtTransportType(
    PtTransportCtrl_t *ctrl,
    char const * const type,
    char const * const desc,
    int unsigned const grouping_num,
    int unsigned inlined_transport,
    char *packing_type,
    void *vdata,
    int unsigned len,
    int unsigned const flags);
```

Arguments:

- `ctrl` A pointer to the `PtTransportCtrl_t` control structure for the drag-and-drop operation. Multiple data blocks can be packed into the same `PtTransportCtrl_t`.
  This structure must have been created via a call to `PtCreateTransportCtrl()`.

- `type` A descriptive type name, such as `image`, `text`, `filename`, or `files`. This is simply added to the header for the packed data.
  Each type has its most common and expected `packing_type` associated with it:

<table>
<thead>
<tr>
<th>Type</th>
<th>Packing type</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>PhImage</td>
</tr>
<tr>
<td>text</td>
<td>string</td>
</tr>
<tr>
<td>files</td>
<td>files or PhTransFiles</td>
</tr>
</tbody>
</table>

Other packing types can be used, but there’s no guarantee the reader/recipient of the data is expecting the type of data packed. In this case, the data is ignored.
**PtTransportType()**

*desc*  
The specifics of what’s in the data. The extractor uses a regular-expression match against the description to determine if the data should be unpacked or discarded. This is simply added to the header for the packed data.

*grouping_num*  
When used with Photon’s drag and drop mechanism, the *grouping_num* is used to indicate which stream is just a different representation of other data also packed into the same `PhTransportCtrl_t` structure. Only one of each *grouping_num* should be unpacked by the reader/destination.

This value is simply added to the header for the packed data.

*inlined_transport*  
The transport type to be used for the inlined data. This may be one of:

- `Ph_TRANSPORT_INLINE` — the data being transported is in memory and can be unpacked immediately.
- `Ph_TRANSPORT_FILEREF` — the data being transported is in the temporary file(s) named in the inlined data.
- `Ph_TRANSPORT_SHMEM` — the data being transported is in the temporary shared object(s) named in the inlined data.
- `Ph_TRANSPORT_STREAM` — the data being transported will be inlined a small piece at a time.
- `Ph_TRANSPORT_NAMED_STREAM` — the data being transported will be inlined a small piece at a time. The streamed data is named so multiple streams of data can be transferred serially.
PtTransportType()

- Ph_TRANSPORT_FILE_STREAM — the contents of files streamed using extended named streams. This is like the named stream but with extra information with each data block, including file information and so on.

packing_type  The name of the transport registry entry to be used when packing the data.

vdata  A pointer to the data to be packed inline.

len  The size, in bytes, of the data pointed to by vdata; This argument is only used for packing raw data.

flags  Flags that affect the operation:

- Ph_DONT_COPY — refer to the data to be transported instead of physically copying the data to the transport control’s stream buffers.

If this bit is set, any modifications to the data that occur between packing and actual transport of the data will be reflected in the data transported.

Library:

ph

Description:

This function is used to pack inline data for a drag-and-drop operation. It’s called by the source widget.

Returns:

0  Success.

-1  The call has failed; errno is set.
**PtTransportType()**

**Errors:**

- **EINV AL** One of the following occurred:
  - One of the parameters was passed incorrectly.
  - No transport control was provided.
  - No data was provided via `vdata`.
  - No `inlined_transport` was specified.
- **ENOENT** No transport registry entry was found for the provided `packing_type`.
- **ENOMEM** Not enough memory to pack the provided data.

**Classification:**

- Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PhTransportCtrl_t`, `PtCreateTransportCtrl()`, `PtInitDnd()`,
- `PtTransportCtrl_t`, `PtTransportRequestable()`
- Drag and Drop chapter of the Photon *Programmer’s Guide*
**PtUnblockWindows()**

 Unblock a set of previously blocked windows

**Synopsis:**

```c
void PtUnblockWindows( PtBlockedList_t *bl );
```

**Library:**

`ph`

**Description:**

This function unblocks windows that were blocked by `PtBlockAllWindows()` or `PtBlockWindow()` and restores their cursors.

The `bl` argument must either be `NULL` or the pointer to a control structure returned by `PtBlockAllWindows()` or `PtBlockWindow()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`ApModalWait()`, `PtBlockAllWindows()`, `PtBlockWindow()`, `PtMakeModal()`

“Modal dialogs” in the Window Management chapter of the Photon Programmer’s Guide
**PtUnlinkData()**

*Remove the provided data link from the data chain*

**Synopsis:**

```c
int PtUnlinkData( PtDataHdr_t **ptr,
                  PtDataHdr_t *node );
```

**Library:**

`ph`

**Description:**

This function removes the provided data link from the data chain. The link is freed, but its data isn’t.

**Returns:**

0 on success, or -1 if `node` couldn’t be found in the data list.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtAddData()`, `PtFindData()`, `PtFindNextData()`, `PtRemoveData()`
**PtUnrealizeWidget()**  
*Unrealize a widget*

**Synopsis:**

```c
int PtUnrealizeWidget( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This function unrealizes the specified widget and all its children: the widgets are removed from the display, and the widget engine will no longer invoke their callbacks.

Unrealized widgets still exist in the widget hierarchy and can be realized again.

Unrealizing and realizing a widget can take some time. If you want to hide a widget quickly, you can set its *Pt_ARG_POS* to a very negative value. You should also set *Pt_BLOCKED* in its *Pt_ARG_FLAGS* so the widget won’t get focus.

**Returns:**

- 0 Successful completion.
- -1 An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 13 • Pt—Widget Toolkit  1699
PtUnrealizeWidget()  

See also:  

PtDestroyWidget(), PtRealizeWidget()  

“Widget life cycle” in the Introduction to the Photon Programmer’s Guide
Synopsis:

```c
int PtUpdate( void );
```

Library:

```c
ph
```

Description:

This function decrements the hold count, which was previously incremented by a call to `PtHold()`. When the count reaches 0, the Photon libraries repair any damaged widgets.

This function is the same as `PtRelease()`.

Returns:

The current hold count, or -1 if an error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PtContainerHold()`, `PtContainerRelease()`, `PtFlush()`, `PtHold()`, `PtModalStart()`, `PtRelease()`

“Delaying and forcing updates to the display” in the Working with Code chapter of the Photon *Programmer’s Guide*
PtValidParent()
Identify a valid parent for the specified widget

Synopsis:
```
PtWidget_t * PtValidParent(
    PtWidget_t *widget,
    PtWidgetClassRef_t *class_ref);
```

Library:
```
ph
```

Description:
This function determines the real parent widget for widgets of type
```
class_ref
```
, given that its parent is specified as `widget_parent` when
created or reparented. Examples of a `class_ref` include `PtWidget`,
`PtBasic`, `PtLabel`, `PtMeter`, and so on.
The real parent of a widget might not be the specified `widget_parent`
in special circumstances where the specified parent widget redirects
the child to an alternate parent. Some examples:

- `PtScrollArea` redirects all widgets created as its children or
  reparented to it to its virtual canvas (a container widget within the
  scroll area).

- `PtMenuBar` redirects all widgets that aren’t of type
  `PtMenuButton` to the `PtMenuBar`’s parent (i.e. `PtMenuBar`
  accepts only `PtMenuButton` widgets as its children).

Returns:
A pointer to the widget that will be the real parent of any widgets of
```
type class_ref created in or reparented to widget_parent.
```

Examples:
```
PtWidget_t *
MyRedirector( PtWidget_t *widget )
{
    MyWidget_t *my = (MyWidget_t * ) widget;

    PtWidget_t *parent;
    if( ( parent =
```
PtValidParent()

PtValidParent( my->scroll_area,
      widget->class_ref ) ) == widget )
return PtWidgetParent( widget );
return( parent );

/*
 * Returning my->scroll_area would allow the child
 * to be created as a direct child of my->scroll_area.
 * This would be undesirable because scroll_area is a
 * compound widget, which also needs to redirect its
 * children to work correctly.
 */
}

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtFindGuardian(), PtGetParent(), PtGetParentWidget(), PtReparentWidget(), PtSetParentWidget(), PtWidgetParent()

PtWidgetActiveSurface()
Get a widget’s currently active control surface

Synopsis:
uchar_t PtWidgetActiveSurface( PtWidget_t *widget );

Library:
ph

Description:
This macro retrieves the provided widget’s currently “active” surface. The active surface is the last surface that consumed an event of type Ph_EV_BUT_PRESS.

Returns:
The ID of the currently active control surface, or 0 if there isn’t one.

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:
PtFindSurface(), PtFindSurfaceByAction()
Control Surfaces chapter of the Photon Programmer’s Guide
Synopsis:

\[
\text{PhArea}_t \, \ast \, \text{PtWidgetArea}( \text{PtWidget}_t \, \ast \, \text{widget}, \\
\text{PhArea}_t \, \ast \, \text{area} );
\]

Library:

\text{ph}

Description:

This macro retrieves a copy of \text{widget}'s area and stores it in the \text{PhArea}_t structure pointed to by \text{area}.

Returns:

The same pointer as \text{area}, or NULL if \text{widget} or \text{area} is NULL.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

\text{PhArea}_t, \text{PtWidgetDim()}
**PtWidgetBrotherBehind()**

*Get the brother behind a widget*

**Synopsis:**

```c
PtWidget_t *PtWidgetBrotherBehind(
    PtWidget_t *widget);
```

**Arguments:**

- `widget` A pointer to the widget whose brother you want to find.

**Library:**

`ph`

**Description:**

This macro returns a pointer to the brother behind `widget`. If there's no brother behind `widget`, the macro returns NULL.

**Examples:**

```c
#include <stdlib.h>
#include <Pt.h>

int main ()
{
    PtWidget_t *window, *group, *child;
    PtArg_t argt[5];
    char *name;

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    group = PtCreateWidget( PtGroup, Pt_DEFAULT_PARENT, 0, NULL );

    /* Create some buttons in the group. */
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 1", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 2", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 3", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
    PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 4", 0 );
    PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
```
PtWidgetBrotherBehind()

/* Traverse the group from back to front. */

printf("From back to front:\n");
for (child = PtWidgetChildBack(group); child;
    child = PtWidgetBrotherInFront(child))
{
    PtGetResource(child, Pt_ARG_TEXT_STRING, &name, 0);
    printf(" %s\n", name);
}

/* Traverse the group from front to back. */

printf("\nFrom front to back:\n");
for (child = PtWidgetChildFront(group); child;
    child = PtWidgetBrotherBehind(child))
{
    PtGetResource(child, Pt_ARG_TEXT_STRING, &name, 0);
    printf(" %s\n", name);
}

PtRealizeWidget(window);
PMainLoop();
return EXIT_SUCCESS;

The above code produces this output:

From back to front:
    Child 1
    Child 2
    Child 3
    Child 4

From front to back:
    Child 4
    Child 3
    Child 2
    Child 1

Classification:

Photon
PtWidgetBrotherBehind()

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetParent()

PtWidgetBrotherInFront()
Get the brother in front of a widget

Synopsis:

\[
\text{PtWidget\_t \ast PtWidgetBrotherInFront(}
\text{PtWidget\_t \ast \text{widget})}
\]

Arguments:

widget A pointer to the widget whose brother you want to find.

Library:

ph

Description:

This macro returns a pointer to the brother in front of widget. If there’s no brother in front of widget, the macro returns NULL.

Examples:

See PtWidgetBrotherBehind().

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtWidgetBrotherBehind(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetParent()  
**PtWidgetChildBack()**
Get the child that’s farthest back in a container

**Synopsis:**

```c
PtWidget_t *PtWidgetChildBack( PtWidget_t *widget );
```

**Library:**

```c
ph
```

**Description:**

This macro returns a pointer to the child that’s farthest back in the specified container `widget`. If `widget` doesn’t have any children, the macro returns NULL.

**Examples:**

```c
#include <stdlib.h>
#include <Pt.h>

int main ()
{
    PtWidget_t *window, *group, *child;
    PhPoint_t pos;
    PtArg_t argt[5];

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    pos.x = pos.y = 0;
    PtSetArg(&argt[0], Pt_ARG_POS, &pos, 0);
    PtSetArg(&argt[1], Pt_ARG_RESIZE_FLAGS, Pt_TRUE, Pt_RESIZE_XY_ALWAYS);
    PtSetArg(&argt[2], Pt_ARG_GROUP_ORIENTATION, Pt_GROUP_HORIZONTAL, 0);
    group = PtCreateWidget(PtGroup, Pt_DEFAULT_PARENT, 3, argt);

    // Create some buttons in the group.
    PtSetArg(&argt[0], Pt_ARG_TEXT_STRING, "Child 1", 0);
    PtCreateWidget(PtButton, Pt_DEFAULT_PARENT, 1, argt);

    PtSetArg(&argt[0], Pt_ARG_TEXT_STRING, "Child 2", 0);
    PtCreateWidget(PtButton, Pt_DEFAULT_PARENT, 1, argt);
}
```
PtWidgetChildBack()

PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 3", 0 );
PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );

// Make the front child red, and the backmost one blue.

child = PtWidgetChildFront( group );
PtSetResource ( child, Pt_ARG_COLOR, Pg_RED, 0);

child = PtWidgetChildBack( group );
PtSetResource ( child, Pt_ARG_COLOR, Pg_BLUE, 0);

PtRealizeWidget (window);
PtMainLoop();
return EXIT_SUCCESS;

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtWidgetBrotherBehind(), PtWidgetBrotherInFront(),
PtWidgetChildFront(), PtWidgetInsert(), PtWidgetParent(),
PtWidgetToBack(), PtWidgetToFront()

PtWindowToBack() PtWindowToFront() in the Photon Widget Reference.

**PtWidgetChildFront()**

*Get the child at the very front of a container*

**Synopsis:**

```c
PtWidget_t *PtWidgetChildFront(
    PtWidget_t *widget);
```

**Library:**

ph

**Description:**

This macro returns a pointer to the child at the very front of the specified container `widget`. If `widget` doesn’t have any children, the macro returns NULL.

**Examples:**

See `PtWidgetChildBack()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetInsert(), PtWidgetParent(), PtWidgetToBack(), PtWidgetToFront() PtWindowToBack() PtWindowToFront() in the Photon Widget Reference.*
PtWidgetChildFront()

**PtWidgetClass()**

Return the class of a widget

**Synopsis:**

```c
PtWidgetClassRef_t *PtWidgetClass(
    PtWidget_t *widget);
```

**Library:**

ph

**Description:**

This macro lets you determine a widget’s class. Using the `PtWidgetClassRef_t` pointer, you can create new widgets of the same class or check for specific widget classes.

**Returns:**

A pointer to a `PtWidgetClassRef_t`, or NULL if the widget is NULL.

**Examples:**

```c
/* check the class type of a widget */
if ( PtWidgetClass( widget ) == PtWindow ) {
    /* window processing */
} else {
    /* nonwindow processing */
}
```

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1714 Chapter 13 • Pt—Widget Toolkit
PtWidgetClass()

See also:

PtWidgetIsClass(), PtWidgetIsClassMember()
PtWidgetClassFlags()

Retrieve a widget's class structure flags

Synopsis:

\[
\text{unsigned long PtWidgetClassFlags(}
\quad \text{PtWidget_t *widget)};
\]

Library:

ph

Description:

This macro retrieves widget's class structure flags:

- \texttt{Pt\_CLEAN\_RESOURCES} (set, cleared, and used internally)
  
  Indicates that all resources in a widget class's resource list are in a single range. This is used as an optimization allowing a widget's resources to be indexed as an array.

- \texttt{Pt\_CONTAINER}
  
  The widget class is a container.

- \texttt{Pt\_DISJOINT} (e.g. PtWindow, PtMenu, PtRegion)
  
  Indicates that widgets of this class own regions that aren’t children of the regions of their widget parents. Any clipping set by the parent of a disjoint widget won’t be applied to the disjoint widget. The regions of disjoint widgets are sensitive and opaque to expose events.

- \texttt{Pt\_FORCE\_UNREALIZE} (set automatically when required)
  
  The class Unrealization function (\texttt{unrealize\_f}) for this class and its superclasses will be called when this widget is unrealized.

- \texttt{Pt\_INDEX\_RESOURCES} (set, cleared, and used internally)
  
  Indicates the resources are clean and continuously sequential—the resources may be indexed instead of traversed.

- \texttt{Pt\_NO\_INHERITED\_RESOURCES}
  
  Prevents the search for a resource from walking up through superclasses. Only the resources for this class are handled; all
others are ignored. This doesn’t prevent resources from being propagated to procreated widgets. This is handy for allowing common resources such as 

*Pt_ARG_COLOR* to pass to procreated children without having to write a resource-redirection function.

**Pt_OCCLUSIVE**

Drawing routines skip all children of a widget derived from an occlusive class.

**Pt_RECTANGULAR**

Rectangular widgets are opaque when filled. Opaque widgets don’t damage widgets below them when they are modified (unless their size or position is modified).

**Pt_UNCLEAN_RESOURCES**

Prevents the resources of a widget class from being indexed. This is necessary if the widget class only defines resources to override a superclass (i.e. the widget class doesn’t define any new resources).

### Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

*PtWidgetClass(), PtWidgetFlags()*
PtWidgetDim()
Retrieve a copy of a widget's dimension

Synopsis:

```
PhDim_t * PtWidgetDim( PtWidget_t *widget,
                       PhDim_t *dim );
```

Library:

```
ph
```

Description:

This macro retrieves a copy of `widget`'s dimension and stores it in the `PhDim_t` structure pointed to by `dim`.

Returns:

The same pointer as `dim`, or NULL if `widget` or `dim` is NULL.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

```
PhDim_t, PtWidgetArea()
```

1718 Chapter 13 • Pt—Widget Toolkit
PtWidgetExtent()
Get a widget’s extent

Synopsis:

```c
PhRect_t *PtWidgetExtent( PtWidget_t *widget,
                          PhRect_t *extent);
```

Library:

`ph`

Description:

This macro sets the specified `PhRect_t` structure to the extent of the specified `widget` and returns a pointer to that structure.

Returns:

The same pointer as `extent`, or NULL if no `PhRect_t` structure is provided or if the widget pointer is invalid.

A widget’s extent isn’t calculated until the widget is either realized or forced to calculate the extent by a call to `PtExtentWidget()`. If the widget hasn’t been realized, be sure to call `PtExtentWidget()` first.

Examples:

```c
PhRect_t extent;
PtWidget_t *labelwidget;

PtRealizeWidget( labelwidget );
PtWidgetExtent( labelwidget, &extent);

See also `PtContainerGiveFocus()`.
```

Classification:

Photon

Safety

Interrupt handler  No

continued…
PtWidgetExtent()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t, PtExtentWidget(), PtRealizeWidget()

“Widget geometry” in the Introduction to the Photon Programmer’s Guide
**Synopsis:**

```c
PtWidget_t *PtWidgetFamily( PtWidget_t *root,
                             PtWidget_t *current );
```

**Library:**

`ph`

**Description:**

This function walks the depth of the widget hierarchy, starting from the top widget.

**Returns:**

A pointer to the next widget below `root` in the widget hierarchy. When the hierarchy has been fully traversed, the function returns NULL.

**Examples:**

```c
PtWidget_t *mycontainer;
PtWidget_t *root;
PtWidget_t *current;
int n;

mycontainer = PtCreateWidget( PtContainer, Pt_DEFAULT_PARENT,
                              n, args );

/*
 * Set and highlight all PtLabel widgets
 * within "mycontainer"
 */
root = current = mycontainer;
PtSetArg( &arg, Pt_ARG_FLAGS, Pt_TRUE,
          Pt_HIGHLIGHTED|Pt_SET );
while( current = PtWidgetFamily( root, current ) )
  if( PtWidgetIsClass( current, PtLabel ) )
    PtSetResources( current, 1, &arg );
PtRealizeWidget( mycontainer );
```
**PtWidgetFamily()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetParent(), PtWidgetSkip(), PtWidgetToBack(), PtWidgetToFront(), PtWidgetTree(), PtWidgetTreeTraverse(*

Synopsis:

```c
long PtWidgetFlags( PtWidget_t *widget );
```

Library:

`ph`

Description:

This function retrieves a widget’s flags. For the meanings of the bits in this flag variable, see:

- The description of the `Pt_ARG_FLAGS` resource for `PtWidget` in the Widget Reference
- Or
- `PtWidget.h`

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtWidgetClassFlags()`
**PtWidgetHelpHit()**

*Find the first widget at a given position that has a help topic*

**Synopsis:**

```c
PtWidget_t * PtWidgetHelpHit(
    PtWidget_t *container,
    PhPoint_t const *pos);
```

**Arguments:**

- `container` A pointer to the container to be searched for widgets.
- `pos` A pointer to a `PhPoint_t` structure that specifies the position, relative to the container widget’s canvas.

**Library:**

`ph`

**Description:**

This function finds the first widget inside the given container at the given position that contains a help topic (i.e. `Pt_ARG_HELP_TOPIC` contains a non-NULL string).

**Returns:**

A pointer to the widget found; NULL if none was found.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
See also:

PhPoint_t, PtContainerHit()
**PtWidgetInsert()**

Insert a widget in the widget family hierarchy

**Synopsis:**

```c
int PtWidgetInsert ( PtWidget_t *widget,
                     PtWidget_t *new_sibling,
                     int behind);
```

**Library:**

ph

**Description:**

This function inserts `widget` into the widget family hierarchy as a brother of the widget `new_sibling`, based on the value of `behind`:

- **0** Insert `widget` in front of `new_sibling`.
- **1** Insert `widget` behind `new_sibling`.

Use this function to insert a widget into the focus order of a group.

Remember that the focus order goes from the back of the widget family hierarchy to the front.

For example, if you have widget A and widget C with a focus order of `A→C`, you can insert widget B into the focus order after widget A by making the following call:

```c
PtWidgetInsert (B, A, 0);
```

The focus order is then `A→B→C`.

**Returns:**

- **0** Success.
- **-1** An error occurred.
**PtWidgetInsert()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler  No</td>
</tr>
<tr>
<td>Signal handler       No</td>
</tr>
<tr>
<td>Thread               No</td>
</tr>
</tbody>
</table>

**See also:**

- PtGetParentWidget(), PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetParent(), PtWidgetToFront(), PtWidgetToBack(), PtWidgetToFront()  
  
**PtWidgetIsClass()**

Determine whether a widget is a specific class type

Synopsis:

```c
int PtWidgetIsClass( PtWidget_t *widget,
                    PtWidgetClassRef_t *class );
```

Library:

ph

Description:

This macro determines whether the specified `widget` is of the specified `widget class`.

Returns:

0 The widget isn’t of the given class type.
1 The widget is of the given class type.

Examples:

Test to see if `widget` is a `PtLabel`-class widget:

```c
if( PtWidgetIsClass( widget, PtLabel ) )
    printf( "PtLabel-class widget\n" );
else
    printf( "non PtLabel-class widget\n" );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtWidgetIsClass()

See also:

PtWidgetIsClassMember()
**PtWidgetIsClassMember()**

*Determine whether a widget belongs to a specified class*

**Synopsis:**

```c
int PtWidgetIsClassMember(
    PtWidget_t *widget,
    PtWidgetClassRef_t *class
);
```

**Library:**

`ph`

**Description:**

This function determines whether or not the specified `widget` is a member of the specified widget `class`. You can use this function to determine whether the widget is of the specified `class` or a subclass of the specified `class`.

**Returns:**

- 0 if `widget` isn’t a member of the given `class`.
- 1 if `widget` is a member of the given `class`.

**Examples:**

Test to see if `widget` belongs to the `PtGraphic` class (i.e. is it a line, rectangle, polygon, etc.):

```c
if( PtWidgetIsClassMember( widget, PtGraphic ) )
    printf( "PtGraphic-class widget\n" );
else
    printf( "non-PtGraphic-class widget\n" );
```

**Classification:**

Photon
PtWidgetIsClassMember()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtWidgetIsClass()
**PtWidgetIsRealized()**

*Determine whether or not a widget is realized*

**Synopsis:**

```c
int PtWidgetIsRealized ( PtWidget_t *widget );
```

**Library:**

```c
ph
```

**Description:**

This macro checks to see if the given widget is realized.

**Returns:**

A nonzero value if the widget is realized, otherwise 0.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
**PtWidgetMinimumSize()**

* Determine the minimum permissible size of a widget

**Synopsis:**

```
PhDim_t *PtWidgetMinimumSize(
    PtWidget_t const *widget,
    PhDim_t *dim);
```

**Library:**

`ph`

**Description:**

This function calculates the minimum size a widget can be, taking into account its current border sizes, resize policy, and value for its `Pt_ARG_MINIMUM_DIM` resource.

The resulting dimension is stored in the `PhDim_t` structure pointed to by `dim`.

**Returns:**

A pointer to the `PhDim_t` structure that was passed in.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PhDim_t`, `PtWidgetPreferredSize()`
**PtWidgetOffset()**

*Find the offset of a widget’s origin*

**Synopsis:**

```c
PhPoint_t *PtWidgetOffset( PtWidget_t *widget,
                           PhPoint_t *offset );
```

**Library:**

ph

**Description:**

This function determines the offset of `widget`’s origin from its disjoint parent widget and stores it in the `PhPoint_t` structure pointed to by `offset`.

**Returns:**

A pointer to a `PhPoint_t` structure that’s the offset of the widget’s origin from its parent window (the widget’s position is relative to this point), or NULL if an error occurs.

**Examples:**

```c
PtArg_t arg;
PhPoint_t *widget_pos;
PtSetArg( &arg, Pt_ARG_POS, &widget_pos, 0);
PtGetResources( labelwidget, 1, &arg );
if(PtWidgetOffset( labelwidget, &point))
{
    widget_position_relative_to_window.x =
        point.x + widget_pos->x;
    widget_position_relative_to_window.y =
        point.y + widget_pos->y;
}
```

**Classification:**

Photon
PtWidgetOffset()

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhPoint_t, PhTranslateRect(), PtGetAbsPosition()
**PtWidgetParent()**
*Get a widget's parent*

**Synopsis:**

```c
PtWidget_t *PtWidgetParent( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This macro returns a pointer to the parent of the specified `widget`. If no parent exists, the macro returns NULL.

Some container widgets, including `PtDivider`, `PtMenuBar`, `PtMultiText`, and `PtScrollArea` redirect children to an alternate parent. For all container widgets, it’s best to call `PtValidParent()` to determine the “real” parent of the children.

**Examples:**

See `PtValidParent()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

1736 Chapter 13 • Pt—Widget Toolkit
PtWidgetParent()

See also:

PtCreateWidget(), PtFindGuardian(), PtGetParent(),
PtGetParentWidget(), PtReparentWidget(), PtSetParentWidget(),
PtValidParent(), PtWidgetBrotherBehind(),
PtWidgetBrotherInFront(), PtWidgetChildBack(),
PtWidgetChildFront()

PtWidgetPreferredSize()
Retrieve the preferred size of a widget

Synopsis:

PhDim_t *PtWidgetPreferredSize(
    PtWidget_t const *widget,
    PhDim_t *dim);

Library:

ph

Description:

This function retrieves the optimal size of a widget, which is recorded whenever the widget calls PtResizeCanvas() or PtAttemptResize() while calculating its extent (see Building Custom Widgets).

The preferred size is the size the widget would be if Pt_RESIZE_XY_ALWAYS were set in its Pt_ARG_RESIZE_FLAGS. The preferred size is never any smaller than the widget’s minimum size. The resulting dimension is stored in the PhDim_t structure pointed to by dim.

Returns:

A pointer to the PhDim_t structure that was passed in.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtWidgetPreferredSize()

See also:

PhDim_t, PtWidgetMinimumSize()

PtAttemptResize(), PtResizeCanvas() in Building Custom Widgets
**PtWidgetRid()**

Get a widget's region ID

**Synopsis:**

```c
PhRid_t PtWidgetRid( PtWidget_t *widget );
```

**Library:**

`ph`

**Description:**

This macro returns the region ID of the specified `widget`. If `widget` doesn’t have a region, the macro returns 0.

**Examples:**

See `PtGetParent()`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PtWidgetBrotherBehind()`, `PtWidgetBrotherInFront()`, `PtWidgetChildFront()`, `PtWidgetChildBack()`, `PtWidgetFamily()`, `PtWidgetParent()`, `PtWidgetToFront()`, `PtWidgetToBack()`
PtWidgetSkip()

Skip to a widget in the next hierarchy

Synopsis:

```c
PtWidget_t * PtWidgetSkip( PtWidget_t *root,
                          PtWidget_t *widget );
```

Library:

ph

Description:

This function skips the hierarchy rooted at `widget` but does not return widgets any higher than `root`.

Returns:

A pointer to a widget in the next hierarchy, or NULL if there isn’t another hierarchy to skip to.

Examples:

```c
// Set the fill color of all REALIZED widgets to white.

PtArg_t argt;
int flags, skip = 0;
PtSetArg( &argt, Pt_ARG_FILL_COLOR, Pt_WHITE, 0 );

for( wp = widget; wp; wp = skip ? PtWidgetSkip(root, wp )
    : PtWidgetFamily( root, wp ) ){  
    skip = 0;
    flags = PtWidgetFlags( wp );

    if( !( flags & Pt_REALIZED ) ) {  
        // completely skip this hierarchy.
        skip = 1;
        continue;
    }

    PtSetResources( wp, 1, &argt );
}
```
**PtWidgetSkip()**

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtNextTopLevelWidget()`, `PtWidget Brother Behind()`,
`PtWidget Brother In Front()`, `PtWidget Child Back()`,
`PtWidget Child Front()`, `PtWidget Family()`, `PtWidget Tree()`,
`PtWidget Tree Traverse()`

**PtWidgetToBack()**

*Move a widget behind all its brothers*

**Synopsis:**

```c
int PtWidgetToBack( PtWidget_t *widget );
```

**Library:**

ph

**Description:**

This function moves the specified `widget` behind all its brothers (i.e. away from the user). All of `widget`'s children are moved too. Any widgets damaged as a result of this operation are automatically repaired.

This function doesn’t work for `PtWindow` widgets — their positions are controlled by the Window Manager. To move a window to the back of the workspace, use `PtWindowToBack()`, which is described in the Photon Widget Reference.

**Returns:**

- 0 Successful completion.
- -1 An error occurred.

**Examples:**

```c
#include <stdlib.h>
#include <Pt.h>

int main ()
{
    PtArg_t argt[5];
    char *name;

    if (PtInit(NULL) == -1)
        exit(EXIT_FAILURE);

    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL)) == NULL)
        PtExit(EXIT_FAILURE);

    printf("Initial Position: %s\n", window->name);

    if (PtWidgetToBack(window) == 0)
        printf("Moved to Back\n");

    printf("Final Position: %s\n", window->name);

    return 0;
}
```

Chapter 13 • Pt—Widget Toolkit 1743
group = PtCreateWidget( PtGroup, Pt_DEFAULT_PARENT, 0, NULL );

// Create some buttons in the group.
PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 1", 0 );
PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 2", 0 );
child2 = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 3", 0 );
child3 = PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );
PtSetArg( &argt[0], Pt_ARG_TEXT_STRING, "Child 4", 0 );
PtCreateWidget( PtButton, Pt_DEFAULT_PARENT, 1, argt );

/* Traverse the group from back to front. */
printf ("From back to front:\n");
for (child = PtWidgetChildBack( group );
    child;
    child = PtWidgetBrotherInFront( child ))
{
    PtGetResource ( child, Pt_ARG_TEXT_STRING, &name, 0);
    printf (" %s\n", name);
}

/* Move Child 2 to the front, and Child 3 to the back. */
PtWidgetToFront (child2);
PtWidgetToBack (child3);

/* Traverse the group from back to front. */
printf ("From back to front:\n");
for (child = PtWidgetChildBack( group );
    child;
    child = PtWidgetBrotherInFront( child ))
{
    PtGetResource ( child, Pt_ARG_TEXT_STRING, &name, 0);
    printf (" %s\n", name);
}

PtRealizeWidget (window);
PtMainLoop();
return EXIT_SUCCESS;

The above code produces this output:

From back to front:
Child 1
Child 2
PtWidgetToBack()

Child 3  
Child 4  
From back to front:  
Child 3  
Child 1  
Child 4  
Child 2

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetInsert(), PtWidgetParent(), PtWidgetToFront()

PtWindowToBack() PtWindowToFront() in the Photon Widget Reference.

PtWidgetToFront()

Move a widget in front of all its brothers

Synopsis:

```
int PtWidgetToFront( PtWidget_t *widget );
```

Library:

```
ph
```

Description:

This function moves the specified `widget` in front of all its brothers (i.e. toward the user). All of `widget`'s children are moved too. Any widgets damaged as a result of this operation are automatically repaired.

This function doesn’t work for PtWindow widgets — their positions are controlled by the Window Manager. To move a window to the front of the workspace, use PtWindowToFront(), which is described in the Photon Widget Reference.

Returns:

0 Successful completion.

-1 An error occurred.

Examples:

See PtWidgetToBack().

Classification:

Photon

Safety

Interrupt handler  No

continued . . .
**PtWidgetToFront()**

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetInsert(), PtWidgetParent(), PtWidgetToBack()*

*PtWindowToBack() PtWindowToFront() in the Photon Widget Reference.*

**PtWidgetTree()**

Walk the widget tree from front to back

**Synopsis:**

```c
int PtWidgetTree( PtWidget_t *root,
                 PtWidget_t **cur,
                 int D );
```

**Library:**

ph

**Description:**

This function walks the widget tree pointed to by `root` from front to back. The `cur` argument is the address of a pointer to the current widget in the tree. The `D` argument is the direction control.

This function performs a simple traversal of the widget tree. If you need more control over how the tree is traversed (for example, specifying the direction, or skipping certain branches), use `PtWidgetTreeTraverse()`.

To start the traversal, set `cur` to be the address of a pointer to the root widget, and set `D` to `Pt_TRAVERSE_START`. Use the result returned by `PtWidgetTree()` as the value of `D` for the next call.

The traversal is done when `PtWidgetTree()` returns `Pt_TRAVERSE_DONE`.

**Returns:**

`Pt_TRAVERSE_DONE`

All the widgets in the tree have been traversed.

Any other value

Pass this value as `D` in the next call.
PtWidgetTree()

Examples:

```c
PtWidget_t *cur, *window;
int d;
::
cur=window
d=Pt_TRAVERSE_START;
while( ( d=PtWidgetTree( window, &cur, d ) ) != Pt_TRAVERSE_DONE)
    PtSetResources( cur, 1, argt );
::
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- PtNextTopLevelWidget()
- PtValidParent()
- PtWidgetBrotherBehind()
- PtWidgetBrotherInFront()
- PtWidgetChildBack()
- PtWidgetChildFront()
- PtWidgetFamily()
- PtWidgetParent()
- PtWidgetSkip()
- PtWidgetTreeTraverse()

PtWidgetTreeTraverse()
Walk the widget family hierarchy from front to back

Synopsis:

```c
int PtWidgetTreeTraverse(
    PtWidget_t *root,
    PtWidget_t **current,
    int direction,
    int (*skip_f)( PtWidget_t *widget,
                   void *data ),
    void *data );
```

Library:

ph

Description:

This function walks the widget family hierarchy from the frontmost widget of the current branch to root.

If a skip_f() function is provided, it’s called prior to the traversal into each branch of the family hierarchy. If the skip_f() function returns a value other than Pt_CONTINUE, that branch may be skipped. The skip_f() function is passed the current widget (root of the branch to be traversed next) and data, as provided in the last parameter to PtWidgetTreeTraverse().

The direction parameter controls the direction of the traversal to the next current widget. To begin a traversal, a direction of Pt_TRAVERSE_START should be passed. New direction values are returned by the function and should be used in subsequent calls during the traversal.

The direction value returned is treated primarily as a bit field in which the bottom four bits (0xF) are reserved for direction and general state control. These bits are:

- Pt_TRAVERSE_ROOT
  
  current is root.

- Pt_TRAVERSE_LAST
  
  current is last widget in hierarchy to be processed.
PtWidgetTreeTraverse()

Pt_TRAVERSE_BACK
Walk towards root.

Pt_TRAVERSE_FORCE
Return the return value from skip_f unaltered (the return value is usually ORed with direction prior to return).

When the traversal is complete direction equals Pt_TRAVERSE_DONE (0).

The return value from skip_f(), if not Pt_CONTINUE, is ORed with the current direction control value unless the Pt_TRAVERSE_FORCE bit is set in that return value. This result is returned to the calling function (the function invoking PtWidgetTreeTraverse()). If the return value from skip_f() is Pt_CONTINUE, the branch is stepped into without returning from PtWidgetTreeTraverse().

Returns:

0 Successfull completion of traversal.

-1 An error occurred.

Other values Traversal is in progress.

Examples:

Example 1 — Implementation of PtWidgetTree():

```c
static int _skip_delay_realize( PtWidget_t *widget, void *data )
{
  data;
  return (
    ( ( widget->flags & ( Pt_DELAY_REALIZE | Pt_REALIZED ) )
      == Pt_DELAY_REALIZED )
      ? Pt_TRAVERSE_BACK : Pt_CONTINUE );
}
```
int PtWidgetTree( PtWidget_t *root, PtWidget_t **cur, int D )
{
    return PtWidgetTreeTraverse( root, cur, D,
        _skip_delay_realize, NULL );
}

Example 2 — Find the frontmost widget in ABW_panel (unconditionally):

PtWidget_t *current;

(void) PtWidgetTreeTraverse( NULL, &current, Pt_TRAVERSE_START,
    NULL, NULL );
// current now points to the widget at the very front
// of ABW_panel

Example 3 — Find the frontmost widget in ABW_panel that isn’t within a disjoint child:

#define FOUND_DISJOINT 0x10

_skip_disjoint( PtWidget_t *widget, void *data )
{
    return( PtWidgetClassFlags( widget ) & Pt_DISJOINT ?
        FOUND_DISJOINT : Pt_CONTINUE );
}

:
:
dir = Pt_TRAVERSE_START;
while( dir = PtWidgetTreeTraverse( NULL, &current,
        dir, _skip_disjoint, NULL ) )
    if( !( dir & FOUND_DISJOINT ) )
        break;
:
:
Example 4 — Walk the widget family hierarchy from the frontmost descendant within ABW_panel back to ABW_base (skipping disjoint subhierarchies):

#define FOUND_DISJOINT 0x10

_skip_disjoint( PtWidget_t *widget, void *data )
{
    return( PtWidgetClassFlags( widget ) & Pt_DISJOINT ?
        FOUND_DISJOINT : Pt_CONTINUE );
}
PtWidgetTreeTraverse()

::
current = ABW_panel;
dir = Pt_TRAVERSE_START;
while( dir = PtWidgetTreeTraverse( ABW_base, &current, dir, _skip_disjoint, NULL ) )
{
    if ( dir & FOUND_DISJOINT )
        // current is the disjoint widget
        continue;
    // do stuff with current...
}
::

Classification:

Photon

Safety

Interrupt handler  No
Signal handler  No
Thread  No

See also:

PtNextTopLevelWidget(), PtWidgetBrotherBehind(), PtWidgetBrotherInFront(), PtWidgetChildBack(), PtWidgetChildFront(), PtWidgetFamily(), PtWidgetParent(), PtWidgetSkip(), PtWidgetTree()

PtWidgetVisibleExtent()

Calculate the visible portion of a widget

Synopsis:

```c
int PtWidgetVisibleExtent ( PtWidget_t *widget,
                          PhRect_t *rect );
```

Library:

ph

Description:

This function determines the portion of a rectangle, defined in the PhRect_t pointed to by rect, that isn’t clipped by any parent of widget.

Returns:

0 No portion of rect is visible
1 Success — rect contains the portion of the original rectangle that isn’t clipped by any of widget’s parents

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhRect_t
**PtWindowConsoleSwitch()**

Switch to the console a given window’s displayed on

**Synopsis:**

```c
int PtWindowConsoleSwitch( PhRid_t rid );
```

**Library:**

ph

**Description:**

This function causes PWM to switch the current display to the console where the window specified by `rid` is located. The `rid` is the region of a task window, and may be obtained with `PtWidgetRid()`.

**Returns:**

0  Success  
-1  An error occurred. Check the value of `errno` for more information.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtConsoleSwitch(), PtWidgetRid()*

Window Management chapter of the Photon *Programmer’s Guide*
PtWindowGetFrameSize()

*Synopsis:*

```c
int PtWindowGetFrameSize( PtWidget_t *window, PhRect_t *rect );
```

*Arguments:*

- `window`: The window you want to query the size of.
- `rect`: A pointer to a `PhRect_t` in which the function stores the size of the `window`.

*Library:*

`ph`

*Description:*

This function determines the size of the frame for the window pointed to by `window`. The results are stored in the `PhRect_t` structure pointed to by `rect`; each member gives the size of a different part of the frame:

<table>
<thead>
<tr>
<th>Member</th>
<th>Part of frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ul.y</code></td>
<td>Top</td>
</tr>
<tr>
<td><code>lr.y</code></td>
<td>Bottom</td>
</tr>
<tr>
<td><code>ul.x</code></td>
<td>Left</td>
</tr>
<tr>
<td><code>lr.x</code></td>
<td>Right</td>
</tr>
</tbody>
</table>

*Returns:*

- `0` Success.
- `-1` The frame size was estimated, or an error occurred.
\textbf{PtWindowGetFrameSize()}

\textbf{Errors:}

- \texttt{EINVAL}  The widget given isn’t a window.
- \texttt{EOK}  The frame size is an estimate.

\textbf{Classification:}

- Photon

\begin{tabular}{ll}
\textbf{Safety} & \\
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}

\textbf{See also:}

- \texttt{PhRect_t}, \texttt{PtCalcCanvas()}, \texttt{PtWidgetArea()}, \texttt{PtWidgetDim()}, \texttt{PtWidgetExtent()}

Window Management chapter of the Photon \textit{Programmer's Guide}
PtWorkProcF_t, PtWorkProc_t
Type for defining a work procedure function

Synopsis:

typedef int PtWorkProcF_t( void *);
typedef PtWorkProcF_t *PtWorkProc_t;

Description:

These data types define pointers to work procedures. The
PtWorkProcF_t type is the function type that the PtWorkProc_t

type points to. This allows you to do something like this:

PtWorkProcF_t my_work_proc;

int my_work_proc( void *data ) {
    ...
}

The compiler should detect any inconsistencies between the two
declarations of my_work_proc() and give you an error (which is
better than a “pointer mismatch” warning on the call to
PtAppAddWorkProc()).

Classification:

Photon

See also:

PtAppAddWorkProc()

Parallel Operations in the Photon Programmer's Guide
Chapter 14

Px—Extended
These functions extend Photon’s basic functionality. Using them, you can:

- Access the Photon helpviewer.
- Load graphic files (GIF, BMP, JPG, JPEG, and so on).
- Access textual configuration files.
- Translate characters to and from UTF-8.

*PxConfig*(*) functions are provided in two versions. The standard versions work on a single configuration file at a time. The *PxConfig*(*Cx*)() versions can work on multiple configuration files concurrently.

These functions are supplied only in static form in the Photon library *pHexLib*. You’ll need to link with this library explicitly.
**PxConfigClose(), PxConfigCloseCx()**

Close a configuration file

**Synopsis:**
```c
#include <photon/PxProto.h>

int PxConfigClose( void );

int PxConfigCloseCx(PxCfgContext_t *cx);
```

**Arguments:**
- `cx` *PxConfigCloseCx()* only. The configuration file handle for the file you want to close. This handle is returned by *PxConfigOpenCx().*

**Library:**
- *phexlib*

**Description:**
These functions close a configuration file, and release all the resources associated with that file.

*PxConfigClose()* closes the currently opened configuration file (opened using *PxConfigOpen()).* It doesn’t do anything and returns *Pt_FALSE* if there’s no currently opened configuration file.

*PxConfigCloseCx()* closes the configuration file indicated by `cx` (opened using *PxConfigOpenCx()).* It doesn’t do anything and returns *Pt_FALSE* if `cx` is NULL.

**Returns:**
- *Pt_TRUE* The file was opened and is now closed.
- *Pt_FALSE* No file was open, `cx` was NULL, or there was an error updating the file to reflect the new configuration.
PxConfigClose(), PxConfigCloseCx()

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PxConfigOpen*()}
**PxConfigDeleteEntry(), PxConfigDeleteEntryCx()**

Delete an entry from a configuration file

**Synopsis:**
```
#include <photon/PxProto.h>

int PxConfigDeleteEntry( const char *section,
                        const char *entry );

int PxConfigDeleteEntryCx( PxCfgContext_t *cx,
                          const char *section,
                          const char *entry );
```

**Arguments:**
- `cx` *PxConfigDeleteEntryCx() only.* The configuration file handle for the file you want to delete an entry from. This handle is returned by `PxConfigOpenCx()`.
- `section` The section that contains `entry`. If `section` is NULL, then the current section is searched for a match for `entry`.
- `entry` The entry to delete.

**Library:**
- `phexlib`

**Description:**
These functions delete the entry `entry` from the section `section` in a configuration file.

The configuration file must have been opened for PXCONFIG_WRITE —see `PxConfigOpen()`.

`PxConfigDeleteEntry()` deletes an entry from the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigDeleteEntryCx()` deletes an entry from the configuration file indicated by `cx`. 
\textit{PxConfigDeleteEntry()}, \textit{PxConfigDeleteEntryCx()}

Returns:

\begin{align*}
\text{Pt\_TRUE} & \quad \text{The entry is deleted.} \\
\text{Pt\_FALSE} & \quad \text{Otherwise.}
\end{align*}

Classification:

Photon

\begin{tabular}{|c|c|}
\hline
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\hline
\end{tabular}

See also:

\textit{PxConfigDeleteSection*()}, \textit{PxConfigOpen*()}
**PxConfigDeleteSection(), PxConfigDeleteSectionCx()**

*Delete a section from a configuration file*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigDeleteSection( const char *section );

int PxConfigDeleteSectionCx( PxCfgContext_t *cx, const char *section );
```

**Arguments:**

- **cx**  
  `PxConfigDeleteSectionCx()` only. The configuration file handle for the file you want to delete a section from. This handle is returned by `PxConfigOpenCx()`.

- **section**  
  The section to delete. All entries within the section (up to either the beginning of the next `[` section or end-of-file) are deleted. If the deleted section was current, current section is set to none; otherwise, the current section is not changed. If `section` is NULL, then the current section is deleted, and the current section is set to “none”.

**Library:**

- phexlib

**Description:**

These functions delete the section `section` from the configuration file.

The file must have been opened for `PXCONFIG_WRITE`—see `PxConfigOpen*()`.

`PxConfigDeleteSection()` deletes a section from the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigDeleteSectionCx()` deletes a section from the configuration file indicated by `cx`. 
Returns:

| Pt_TRUE   | The section is deleted. |
| Pt_FALSE  | Otherwise.              |

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PxConfigDeleteEntry*, PxConfigForceEmptySection*, PxConfigOpen*()}
**PxConfigFirstSection(), PxConfigFirstSectionCx()**

*Seek the beginning of the first section of a configuration file*

**Synopsis:**

```
#include <photon/PxProto.h>

const char *PxConfigFirstSection( void );

const char *PxConfigFirstSectionCx(PxCfgContext_t *cx);
```

**Arguments:**

- `cx`  
  `PxConfigFirstSectionCx()` only. The configuration file handle for the file you want to seek in. This handle is returned by `PxConfigOpenCx()`.

**Library:**

*phexlib*

**Description:**

These functions seek the start of the first section, and return the name of this section; this section is made the internal current section. These functions may be used to process a configuration file consisting of unknown sections, but where each section has known entries. If there are no sections, the function returns NULL.

`PxConfigFirstSection()` works on the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigFirstSectionCx()` works on the configuration file indicated by `cx`.

**Returns:**

- A string containing the first section name if one exists, NULL otherwise

**Classification:**

*Photon*
PxConfigFirstSection(), PxConfigFirstSectionCx()  

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:  
PxConfigNextEntry*, PxConfigNextSection*,  
PxConfigNextString*, PxConfigOpen*, PxConfigSection*
**PxConfigForceEmptySection(), PxConfigForceEmptySectionCx()**

*Create an empty section in a configuration file*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigForceEmptySection( const char *section );

int PxConfigForceEmptySectionCx( PxCfgContext_t *cx,
                                 const char *section );
```

**Arguments:**

- `cx`  
  `PxConfigForceEmptySectionCx()` only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- `section`  
  The section to create. If `section` already exists, it becomes the current section, and its entries are left intact. If `section` doesn’t exist, the function creates it and makes it the current section.

**Library:**

`phexlib`

**Description:**

These functions create an empty section `section` if one doesn’t exist.

---

The configuration file must have been opened by `PxConfigOpen*()` with a mode of `PXCONFIG_WRITE`.

Normally sections are created as necessary by the `PxConfigWrite*()` functions to hold entries, but sometimes the mere presence of a configuration section conveys application information. The new section is made the internal current section.

`PxConfigForceEmptySection()` writes to the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigForceEmptySectionCx()` writes to the configuration file indicated by `cx`. 

---

1770 Chapter 14 • Px—Extended
Returns:

Pt_TRUE The section is created or exists.
Pt_FALSE Otherwise.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PxConfigDeleteSection*, PxConfigOpen*
** PxConfigNextEntry(), PxConfigNextEntryCx() **

Get the next entry in the current section

**Synopsis:**

```c
#include <photon/PxProto.h>

const char *PxConfigNextEntry( char **value );

const char *PxConfigNextEntryCx( PxCfgContext_t *cx,
                                 char **value );
```

**Arguments:**

- `cx`  
  PxConfigNextEntryCx() only. The configuration file handle for the file you want to get the next string in. This handle is returned by PxConfigOpenCx().

- `value`  
  If passed in as a non-NULL pointer, the function fills this in with pointer to a string containing the value of the next entry.

**Library:**

phexlib

**Description:**

These functions return the next entry in the current section as a pointer to a string. A pointer to the entry name is returned, and `value` is filled in with a pointer to the string containing its configuration value. This may be used to process a configuration section consisting of unknown entries, but where each entry is to be processed in a similar fashion.

These functions are similar to PxConfigNextString*(), but because they don’t copy the string, they are faster.

If there is no current section (for example, if the file has just been opened), PxConfigNextEntry*() seeks to the beginning of the first section, and returns the first entry.
Returns:

A string containing the next entry name within the current section if one exists, NULL otherwise.

If PxConfigNextEntry*() detects the end of the section, it returns NULL. If you call PxConfigNextEntry() again, it gets the next entry in the next section.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PxConfigFirstSection*(), PxConfigNextEntry*(), PxConfigNextSection*(), PxConfigOpen*(), PxConfigSection*()
PxConfigNextSection(), PxConfigNextSectionCx()
Seek the beginning of the next section of a configuration file

Synopsis:
#include <photon/PxProto.h>

const char *PxConfigNextSection( void );

const char *PxConfigNextSectionCx(PxCfgContext_t *cx);

Arguments:

    cx       PxConfigNextSectionCx() only. The configuration file handle
              for the file you want to seek in. This handle is returned by
              PxConfigOpenCx().

Library:

    phexlib

Description:

These functions seek the start of the next section, and return the name
of this section; this section is made the internal current section. These
functions may be used to process a configuration file consisting of
unknown sections, but where each section has known entries. If there
aren’t any more sections (that is, the current section is the last section)
the function returns NULL, but the current section and file pointer are
left untouched.

PxConfigNextSection() works on the currently open configuration file
opened by PxConfigOpen(), while PxConfigNextSectionCx() works on
the configuration file indicated by cx.

Returns:

A string containing the next section name if one exists, NULL
otherwise
**PxConfigNextSection(), PxConfigNextSectionCx()**

**Examples:**

```c
char *section;
while ((section = PxConfigNextSection()) != NULL) {
    PxConfigReadShort(NULL, "Size", 0, &recsize);
    PxConfigReadShort(NULL, "Max", 0, &maxrecs);
}
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PxConfigFirstSection*(), PxConfigNextEntry*(), PxConfigNextString*(), PxConfigOpen*(), PxConfigSection*()`
**PxConfigNextString(), PxConfigNextStringCx()**

Get the next entry in the current section

**Synopsis:**

```c
#include <photon/PxProto.h>

const char *PxConfigNextString( char *value, size_t maxlen );

const char *PxConfigNextStringCx( PxCfgContext_t *cx, char *value, size_t maxlen );
```

**Arguments:**

- `cx`  
  *PxConfigNextStringCx() only. The configuration file handle for the file you want to get the next string in. This handle is returned by `PxConfigOpenCx()`.

- `value`  
  A buffer where the function stores the value it reads.

- `maxlen`  
  The maximum length of `value`. `PxConfigNextString()` copies a maximum of `(maxlen-1)` characters (with a `\0` terminator appended) in the string into the buffer at address `value`.

**Library:**

`phexlib`

**Description:**

These functions return the next entry in the current section as a string. A pointer to the entry name is returned, and its configuration value (up to a maximum of `maxlen` characters including trailing NULL) is copied as a string into the buffer at address `value`. This may be used to process a configuration section consisting of unknown entries, but where each entry is to be processed in a similar fashion.

If there is no current section (for example, if the file has just been opened), `PxConfigNextString*()` seeks to the beginning of the first section, and returns the first entry.
**PxConfigNextString()**, **PxConfigNextStringCx()**

**Returns:**

A string containing the next entry name within the current section if one exists, NULL otherwise.

If `PxConfigNextString*()` detects the end of the section, it returns NULL. If you call `PxConfigNextString()` again, it gets the next entry in the next section.

**Examples:**

```c
char *env, val[128];
if (PxConfigSection("Environment") != NULL)
    while ((env = PxConfigNextString(val, sizeof(val))) != NULL)
        setenv(env, val, "0");
```

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PxConfigFirstSection*()`, `PxConfigNextEntry*()`, `PxConfigNextSection*()`, `PxConfigOpen*()`, `PxConfigSection*()`
 PxConfigOpen(), PxConfigOpenCx()  
Open a configuration file

Synopsis:

```c
#include <photon/PxProto.h>

int PxConfigOpen(const char *cfgfile, int mode);

PxCfgContext_t *PxConfigOpenCx(const char *cfgfile, int mode);
```

Arguments:

- `cfgfile`  The configuration file to open
- `mode`    A bitfield specifying what operations may be performed on the configuration file; it consists of a combination of the following values:

  PXCONFIG_READ
  Open the file for reading only. The file must exist.

  PXCONFIG_WRITE
  Allow reading and modification of an existing config file (using PxConfigWrite*() and PxConfigDelete*()), or create a new config file if one doesn’t exist.

  PXCONFIG_CLEAR (formerly PXCONFIG_CREATE)
  Clear the file by removing all entries and sections. This flag only applies when PXCONFIG_WRITE is set. PXCONFIG_CREATE is deprecated.

Depending on which of the above flags are set for `mode`, it can take a combination of these additional flags:

- PXCONFIG_UNLINK
  Remove the file after it’s been read by calling `unlink()` on it. If it is a symbolic link, just the link is removed. This setting is useful for “read-once” scenarios, or if your application wishes to migrate...
PXConfigOpen(), PxConfigOpenCx()

the config file from one location to another. This flag only applies if PXCONFIG_READ is set.

PXCONFIG_HOME

Interpret *cfgfile* as a path relative to the user’s Photon configuration path, which is always
>$HOME/.ph/. If this flag is set, the real file *PxConfigOpen()* attempts to open is a concatenation of this path and *cfgfile*.

PXCONFIG_GLOBAL

Interpret *cfgfile* as a path relative to the system-wide Photon configuration area, which is currently always $PHOTON_PATH/config/. If this flag is set, the real file *PxConfigOpen()* attempts to open is a concatenation of this path and *cfgfile*.

If you set both PXCONFIG_HOME and PXCONFIG_GLOBAL, *PxConfigOpen()* attempts to open the PXCONFIG_HOME file first, then the PXCONFIG_GLOBAL file. The first open to succeed causes *PxConfigOpen()* to return with success, and all subsequent operations are done on that file.

If neither PXCONFIG_HOME nor PXCONFIG_GLOBAL are set, then *cfgfile* is interpreted as the full path of the file to open.

Library:

phexlib

Description:

These functions open the configuration file specified by *cfgfile* for reading and/or writing. The file should be closed using the corresponding *PxConfigClose()* when the configuration procedure is complete.

With *PxConfigOpen()* , you may have only one configuration file open any one time. If there is a file already open, it is first closed and
**PxConfigOpen(), PxConfigOpenCx()**

*PxConfigOpen()* attempts to open the new file regardless of whether or not the close of the old file succeeded. The return code from *PxConfigOpen()* in this case reflects the status of the open of the new file, and does not indicate whether or not the old file was successfully closed. To avoid this ambiguity, you should always call *PxConfigClose()* on an open file prior to opening a new one.

You can use *PxConfigOpenCx()* to open more than one configuration file. The *PxCfgContext_t* returned from this function is the configuration file handle you use to manage multiple files, and you pass it as the *cx* argument to other *PxConfigCx()* functions.

*PxConfigOpenCx()* returns NULL if it fails to open the configuration file. It’s acceptable to pass a NULL *cx* pointer to any of the other *PxConfigCx()* functions. In this case, *PxConfigReadCx()* simply gives back the default value. The other functions return an error code and do nothing more.

The configuration file consists of simple text and is divided into sections, introduced by *section_name*. Each section is made up of entries (one per line) of the form:

```
entry_name = value
```

Lines (entry name and value) are currently limited to 1024 characters; lines longer than that are truncated. Comments follow #, anywhere on a line. Here’s an example:

```
[WWW Section]
Heading Font = swiss
Body Font = dutch
Link Color = 0000FF
Visited Color = 008080
Cache Size = 10240

[File Section]
File Font = swiss12
Print Command = lp @
Display Mode = 1
```
You can use duplicate section names in configuration files, although it is not recommended. Functions such as `PxConfigSection*()` that directly reference a section by name, will return or operate on the first matched section in a file. To advance to the next section with the same name, you have to use `PxConfigNextSection*()` until you reach the desired section.

**Returns:**

`PxConfigOpen()`:

- `Pt_TRUE` The given configuration file has been opened for the specified mode.
- `Pt_FALSE` Otherwise.

`PxConfigOpenCx()`:

- A non-NULL pointer to a valid `PxCfgContext_t`
  - The given configuration file has been opened for the specified access mode.
- `NULL` Otherwise.

**Examples:**

```c
if (PxConfigOpen(fname, PXCONFIG_READ)) {
    // read parameters from the file
    PxConfigClose();
}
```

For user “joe” with home directory `/home/joe`, first attempt to open `/home/joe/.ph/foo/bar.cfg` for read. Failing that, try `/usr/photon/config/foo/bar.cfg` (assuming the default setting for `PHOTON_PATH`):

```c
PxConfigOpen("foo/bar.cfg", PXCONFIG_READ |
            PXCONFIG_HOME | PXCONFIG_GLOBAL);
```
Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PxConfigClose*(), PxConfigDeleteEntry*(),
PxConfigDeleteSection*(), PxConfigFirstSection*(),
PxConfigForceEmptySection*(), PxConfigNextSection*(),
PxConfigNextString*(), PxConfigReadBool*(),
PxConfigReadChar*(), PxConfigReadDouble*(),
PxConfigReadInt*(), PxConfigReadLLong*(), PxConfigReadLong*(),
PxConfigReadShort*(), PxConfigReadString*(), PxConfigSection*(),
PxConfigWriteBool*(), PxConfigWriteChar*(),
PxConfigWriteDouble*(), PxConfigWriteInt*(),
PxConfigWriteLLong*(), PxConfigWriteLong*(),
PxConfigWriteShort*(), PxConfigWriteString*()
**PxConfigReadBool(), PxConfigReadBoolCx()**

*Read a Boolean value from a configuration file*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigReadBool( const char *section,
const char *entry,
int dflt,
int *value);

int PxConfigReadBoolCx( PxCfgContext_t *cx,
const char *section,
const char *entry,
int dflt,
int *value);
```

**Arguments:**

- **cx**  
  `PxConfigReadBoolCx()` only. The configuration file handle for the file you want to read from. This handle is returned by `PxConfigOpenCx()`.

- **section**  
  The section to read a value from. If `section` is NULL, the function reads the value from `entry` in the current section. If `section` is a valid section name, it becomes the current section.

- **entry**  
  The entry to read the value from.

- **dflt**  
  The default to store in `value` if the specified `section` or `entry` don’t exist, or if the entry value is not ON or OFF, YES or NO, TRUE or FALSE (case-insensitive).

- **value**  
  A pointer to an `int` where the function stores the value it reads.

**Library:**

phexlib

Chapter 14 • Px—Extended  1783
**PxConfigReadBool(), PxConfigReadBoolCx()**

**Description:**
These functions read a Boolean parameter from a configuration file. `PxConfigReadBool()` reads from the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigReadBoolCx()` reads from the configuration file indicated by `cx`.

**Returns:**
`Pt_TRUE` if the required section/entry exists and the given value is valid, otherwise `Pt_FALSE`.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**
`PxConfigOpen*()`, `PxConfigReadChar*()`, `PxConfigReadDouble*()`, `PxConfigReadInt*()`, `PxConfigReadLLong*()`, `PxConfigReadLong*()`, `PxConfigReadShort*()`, `PxConfigReadString*()`, `PxConfigWriteBool*()`, `PxConfigWriteChar*()`, `PxConfigWriteDouble*()`, `PxConfigWriteInt*()`, `PxConfigWriteLLong*()`, `PxConfigWriteLong*()`, `PxConfigWriteShort*()`, `PxConfigWriteString*()`
 PxConfigReadChar(), PxConfigReadCharCx()

Read a character parameter from a configuration file

Synopsis:

```c
#include <photon/PxProto.h>

int PxConfigReadChar( const char *section,
                        const char *entry,
                        char dflt,
                        char *value ) ;

int PxConfigReadCharCx( PxCfgContext_t *cx,
                         const char *section,
                         const char *entry,
                         char dflt,
                         char *value ) ;
```

Arguments:

cx  
PxConfigReadCharCx() only. The configuration file handle for the file you want to read from. This handle is returned by PxConfigOpenCx().

section  
The section to read a boolean from.
If section is NULL, the function reads the value from entry in the current section. If section is a valid section name, it becomes the current section.

entry  
The entry to read the value from.

dflt  
The default to store in value if the specified section or entry don’t exist, or if the value is not a digit (signed and unsigned) within the char range, or a single letter,

value  
A pointer to an char where the function stores the value it reads.

Library:

phexlib
**PxConfigReadChar(), PxConfigReadCharCx()**

**Description:**
These functions read a character parameter from a configuration file. `PxConfigReadChar()` reads from the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigReadCharCx()` reads from the configuration file indicated by `cx`.

**Returns:**
Pt_TRUE if the required section/entry exists and the given value is valid, otherwise Pt_FALSE.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadDouble*(), PxConfigReadInt*(), PxConfigReadLLong*(), PxConfigReadLong*(), PxConfigReadShort*(), PxConfigReadString*(), PxConfigWriteBool*(), PxConfigWriteChar*(), PxConfigWriteDouble*(), PxConfigWriteInt*(), PxConfigWriteLLong*(), PxConfigWriteLong*(), PxConfigWriteShort*(), PxConfigWriteString*()`
PxConfigReadDouble(), PxConfigReadDoubleCx()
Read a double-precision float parameter from a configuration file

Synopsis:
#include <photon/PxProto.h>

int PxConfigReadDouble( const char *section,
                        const char *entry,
                        double dflt,
                        double *value );

int PxConfigReadDoubleCx( PxCfgContext_t *cx,
                           const char *section,
                           const char *entry,
                           double dflt,
                           double *value );

Arguments:

 cx  PxConfigReadDoubleCx() only. The configuration file handle for the file you want to read from. This handle is returned by PxConfigOpenCx().

 section The section to read a value from. If section is NULL, the function reads the value from entry in the current section. If section is a valid section name, it becomes the current section.

 entry The entry to read the value from.

 dflt  The default to store in value if the specified section or entry don’t exist.

 value A pointer to a double where the function stores the value it reads.

Library:

phexlib
**PxConfigReadDouble(), PxConfigReadDoubleCx()**

**Description:**
These functions read a double-precision float parameter from a configuration file. 
*PxConfigReadDouble()* reads from the currently open configuration file opened by *PxConfigOpen()* , while *PxConfigReadDoubleCx()* reads from the configuration file indicated by *cx*.

**Returns:**
Pt_TRUE if the required section/entry exists and the given value is valid, otherwise Pt_FALSE.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**
*PxConfigOpen*(), *PxConfigReadBool*(), *PxConfigReadChar*(),
*PxConfigReadInt*(), *PxConfigReadLLong*(), *PxConfigReadLong*(),
*PxConfigReadShort*(), *PxConfigReadString*(),
*PxConfigWriteBool*(), *PxConfigWriteChar*(),
*PxConfigWriteDouble*(), *PxConfigWriteInt*(),
*PxConfigWriteLLong*(), *PxConfigWriteLong*(),
*PxConfigWriteShort*(), *PxConfigWriteString*()
**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigReadInt( const char *section,
                     const char *entry,
                     int dflt,
                     int *value );

int PxConfigReadIntCx( PxCfgContext_t *cx,
                        const char *section,
                        const char *entry,
                        int dflt,
                        int *value );
```

**Arguments:**

- `cx` *PxConfigReadIntCx()* only. The configuration file handle for the file you want to read from. This handle is returned by `PxConfigOpenCx()`.
- `section` The section to read a value from. If `section` is NULL, the function reads the value from `entry` in the current section. If `section` is a valid section name, it becomes the current section.
- `entry` The entry to read the value from.
- `dflt` The default to store in `value` if the specified `section` or `entry` don’t exist, or if the entry value is not a valid integer (see below).
- `value` A pointer to an `int` where the function stores the value it reads.

**Library:**

`phexlib`
PxConfigReadInt(), PxConfigReadIntCx()

Description:

These functions read an integer parameter from a configuration file. The value can be signed. The characters after the sign determine the base:

<table>
<thead>
<tr>
<th>Character(s)</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x or 0X</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>0</td>
<td>Octal</td>
</tr>
<tr>
<td>Other digits</td>
<td>Decimal</td>
</tr>
</tbody>
</table>

PxConfigReadInt() reads from the currently open configuration file opened by PxConfigOpen(), while PxConfigReadIntCx() reads from the configuration file indicated by cx.

Returns:

Pt_TRUE if the required section/entry exists and the given value is valid, otherwise Pt_FALSE.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadChar*(), PxConfigReadDouble*(), PxConfigReadLLong*(), PxConfigReadLong*(), PxConfigReadShort*()
Chapter 14 • Px—Extended 1791
PxConfigReadLLong(), PxConfigReadLLongCx()
Read a long integer parameter from a configuration file

Synopsis:

```
#include <photon/PxProto.h>

int PxConfigReadLLong( const char *section,
                        const char *entry,
                        long long dflt,
                        long long *value );

int PxConfigReadLLongCx( PxCfgContext_t *cx,
                          const char *section,
                          const char *entry,
                          long long dflt,
                          long long *value );
```

Arguments:

cx        PxConfigReadLLongCx() only. The configuration file handle for the file you want to read from. This handle is returned by PxConfigOpenCx().

section    The section to read a value from.
If section is NULL, the function reads the value from entry in the current section. If section is a valid section name, it becomes the current section.

entry      The entry to read the value from.

dflt       The default to store in value if the specified section or entry don’t exist, or if the entry value is not a valid long long (see below).

value      A pointer to a long long where the function stores the value it reads.

Library:

phexlib
**PxConfigReadLLong(), PxConfigReadLLongCx()**

**Description:**

These functions read a `long long` parameter from the specified section and entry of a configuration file and stores it in *value*. The value can be signed. The characters after the sign determine the base:

<table>
<thead>
<tr>
<th>Character(s)</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0x</code> or <code>0X</code></td>
<td>Hexadecimal</td>
</tr>
<tr>
<td><code>0</code></td>
<td>Octal</td>
</tr>
<tr>
<td>Other digits</td>
<td>Decimal</td>
</tr>
</tbody>
</table>

*PxConfigReadLLong()* reads from the currently open configuration file opened by *PxConfigOpen()*, while *PxConfigReadLLongCx()* reads from the configuration file indicated by *cx*.

**Returns:**

*Pt_TRUE* if the required section/entry exists and the given value is valid, otherwise *Pt_FALSE*.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PxConfigReadLLong(), PxConfigReadLLongCx()

See also:

PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadChar*(), PxConfigReadDouble*(), PxConfigReadInt*(), PxConfigReadLong*(), PxConfigReadShort*(), PxConfigReadString*(), PxConfigWriteBool*(), PxConfigWriteChar*(), PxConfigWriteDouble*(), PxConfigWriteInt*(), PxConfigWriteLong*(), PxConfigWriteShort*(), PxConfigWriteString*()
**PxConfigReadLong(), PxConfigReadLongCx()**

*Read a long integer parameter from a configuration file*

**Synopsis:**
```
#include <photon/PxProto.h>

int PxConfigReadLong( const char *section,
                      const char *entry,
                      long dflt,
                      long *value );

int PxConfigReadLongCx( PxCfgContext_t *cx,
                        const char *section,
                        const char *entry,
                        long dflt,
                        long *value );
```

**Arguments:**

- **cx**  
  `PxConfigReadLongCx()` only. The configuration file handle for the file you want to read from. This handle is returned by `PxConfigOpenCx()`.

- **section**  
  The section to read a value from.
  If `section` is NULL, the function reads the value from `entry` in the current section. If `section` is a valid section name, it becomes the current section.

- **entry**  
  The entry to read the value from.

- **dflt**  
  The default to store in `value` if the specified `section` or `entry` don’t exist, or if the entry value is not a valid `long` (see below).

- **value**  
  A pointer to a `long` where the function stores the value it reads.

**Library:**

- `phexlib`
**PxConfigReadLong(), PxConfigReadLongCx()**

**Description:**
These functions read a `long` parameter from the specified section and entry of a configuration file and stores it in `*value`. The value can be signed. The characters after the sign determine the base:

<table>
<thead>
<tr>
<th>Character(s)</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x or 0X</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>0</td>
<td>Octal</td>
</tr>
<tr>
<td>Other digits</td>
<td>Decimal</td>
</tr>
</tbody>
</table>

`PxConfigReadLong()` reads from the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigReadLongCx()` reads from the configuration file indicated by `cx`.

**Returns:**

Pt_TRUE if the required section/entry exists and the given value is valid, otherwise Pt_FALSE.

**Classification:**
Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

-PxConfigOpen*, PxConfigReadBool*, PxConfigReadChar*, PxConfigReadDouble*, PxConfigReadInt*, PxConfigReadLLong*, PxConfigReadShort*, PxConfigReadString*, PxConfigWriteBool*, PxConfigWriteChar*, PxConfigWriteDouble*, PxConfigWriteInt*, PxConfigWriteLLong*, PxConfigWriteLong*, PxConfigWriteShort*, PxConfigWriteString*
** PxConfigReadShort(), PxConfigReadShortCx()  

*Read a short integer parameter from a configuration file*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigReadShort( const char *section,  
                       const char *entry,  
                       short dflt,  
                       short *value );

int PxConfigReadShortCx( PxCfgContext_t *cx,  
                         const char *section,  
                         const char *entry,  
                         short dflt,  
                         short *value );
```

**Arguments:**

- **cx**  
  * PxConfigReadShortCx() only. The configuration file handle for the file you want to read from. This handle is returned by PxConfigOpenCx().

- **section**  
  The section to read a value from. If section is NULL, the function reads the value from entry in the current section. If section is a valid section name, it becomes the current section.

- **entry**  
  The entry to read the value from.

- **dflt**  
  The default to store in value if the specified section or entry don’t exist, or if the entry value is not a valid short.

- **value**  
  A pointer to an short where the function stores the value it reads.

**Library:**

phexlib
**PxConfigReadShort(), PxConfigReadShortCx()**

**Description:**

These functions read a short parameter from the specified section and entry of the configuration file and stores it in *value. The value can be signed. The characters after the sign determine the base:

<table>
<thead>
<tr>
<th>Character(s):</th>
<th>Base:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x or 0X</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>0</td>
<td>Octal</td>
</tr>
<tr>
<td>Other digits</td>
<td>Decimal</td>
</tr>
</tbody>
</table>

**PxConfigReadShort()** reads from the currently open configuration file opened by *PxConfigOpen()*, while **PxConfigReadShortCx()** reads from the configuration file indicated by *cx*.

**Returns:**

Pt_TRUE if the required section/entry exists and the given value is valid, otherwise Pt_FALSE.

**Classification:**

Photon

**Safety**

- Interrupt handler: No
- Signal handler: No
- Thread: No
See also:

PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadChar*(), PxConfigReadDouble*(), PxConfigReadInt*(), PxConfigReadLong*(), PxConfigReadLLong*(), PxConfigReadLong*(), PxConfigReadString*(), PxConfigWriteBool*(), PxConfigWriteChar*(), PxConfigWriteDouble*(), PxConfigWriteInt*(), PxConfigWriteLLong*(), PxConfigWriteLong*(), PxConfigWriteShort*(), PxConfigWriteString*()
**PxConfigReadString(), PxConfigReadStringCx()**

Read a string parameter from a configuration file

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigReadString( const char *section,
                        const char *entry,
                        const char *dflt,
                        char *value,
                        short maxlen );

int PxConfigReadStringCx( PxCfgContext_t *cx,
                          const char *section,
                          const char *entry,
                          const char *dflt,
                          char *value,
                          short maxlen );
```

**Arguments:**

- **cx** 
  - `PxConfigReadStringCx()` only. The configuration file handle for the file you want to read from. This handle is returned by `PxConfigOpenCx()`.

- **section** 
  - The section to read a value from.
  - If `section` is NULL, the function reads the value from `entry` in the current section. If `section` is a valid section name, it becomes the current section.

- **entry** 
  - The entry to read the value from.

- **dflt** 
  - The default to store in `value` if the specified `section` or `entry` don’t exist. The first (`maxlen`-1) characters of the default string are copied (with a ‘\0’ terminator appended).

- **value** 
  - A buffer where the function stores the value it reads.

- **maxlen** 
  - The maximum length of `value`. `PxConfigReadString()` copies a maximum of (`maxlen`-1) characters (with a ‘\0’ terminator appended) in the string into the buffer at address `value`. 
**PxConfigReadString(), PxConfigReadStringCx()**

**Library:**

phexlib

**Description:**

These functions read a string parameter from the specified *section* and *entry* of a configuration file.

`PxConfigReadString()` reads from the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigReadStringCx()` reads from the configuration file indicated by `cx`.

**Returns:**

- `Pt_TRUE` if the required section/entry exists, otherwise `Pt_FALSE`.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PxConfigOpen*()`, `PxConfigReadBool*()`, `PxConfigReadChar*()`, `PxConfigReadDouble*()`, `PxConfigReadInt*()`, `PxConfigReadLLong*()`, `PxConfigReadLong*()`, `PxConfigReadShort*()`, `PxConfigWriteBool*()`, `PxConfigWriteChar*()`, `PxConfigWriteDouble*()`, `PxConfigWriteInt*()`, `PxConfigWriteLLong*()`, `PxConfigWriteLong*()`, `PxConfigWriteShort*()`, `PxConfigWriteString*()`
**PxConfigSection(), PxConfigSectionCx()**

Seek the start of a given section in a configuration file

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigSection( const char *section );

int PxConfigSectionCx(PxCfgContext_t *cx, char const *section);
```

**Arguments:**

- `cx` *PxConfigSectionCx()* only. The configuration file handle for the file you want to seek for a section in. This handle is returned by *PxConfigOpenCx()*.
- `section` The name of a section you want to seek to.

Section names must be an exact match—they must not be abbreviated and the match is case-sensitive. If there is more than one section with the same name, these functions seek to the first matched section in the file. Use *PxConfigNextSection*() to find subsequent sections with the same name.

**Library:**

`phexlib`

**Description:**

These functions seek to the start of the requested section, and return an indication of whether the section exists within the configuration file. These functions may be used to conditionally process an optional section block. Photon also uses it internally to locate a configuration entry; the section is made the internal current section.

If the requested section can’t be found, the file pointer and current section remain untouched, and the function returns Pt_FALSE.
**PxConfigSection(), PxConfigSectionCx()**

**Returns:**

Pt_TRUE if the requested section exists, Pt_FALSE otherwise

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PxConfigDeleteSection*(), PxConfigFirstSection*(), PxConfigForceEmptySection*(), PxConfigNextEntry*(), PxConfigNextSection*(), PxConfigNextString*(), PxConfigOpen*()
**PxConfigWriteBool(), PxConfigWriteBoolCx()**

Write a Boolean parameter in a configuration file

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteBool( const char *section,
                        const char *entry,
                        int format,
                        int value );

int PxConfigWriteBoolCx( PxCfgContext_t *cx,
                          const char *section,
                          const char *entry,
                          int format,
                          int value );
```

**Arguments:**

- `cx` *PxConfigWriteBoolCx()* only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- `section` The section to write `value` to.
  If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.

- `entry` The entry to write `value` to. If the entry already exists in the current section, it’s overwitten; otherwise, it’s added to the end of the section.

- `format` Determines how the `value` is formatted:

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXCONFIG_FMT_BOOL_ON</td>
<td>OFF or ON</td>
</tr>
<tr>
<td>PXCONFIG_FMT_BOOL_YES</td>
<td>NO or YES</td>
</tr>
</tbody>
</table>

  continued...
**PxConfigWriteBool(), PxConfigWriteBoolCx()**

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXCONFIG_FMT_BOOL_TRUE FALSE or TRUE</td>
<td></td>
</tr>
</tbody>
</table>

`value` The value the function writes to the file.

**Library:**

`phexlib`

**Description:**

These functions write a Boolean parameter in the specified `section` and `entry` of a configuration file.

- The configuration file must have been opened by `PxConfigOpen*()` with a mode of PXCONFIG_WRITE.
- If the specified `section` doesn’t exist, it’s created.
- To create an empty section, call `PxConfigForceEmptySection*()`.

`PxConfigWriteBool()` writes to the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigWriteBoolCx()` writes to the configuration file indicated by `cx`.

**Returns:**

Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**

Photon

**Safety**

Interrupt handler No

continued...
**PxConfigWriteBool(), PxConfigWriteBoolCx()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

 PxConfigOpen*(), PxConfigReadBool*, PxConfigReadChar*, PxConfigReadDouble*, PxConfigReadInt*, PxConfigReadLLong*, PxConfigReadLong*, PxConfigReadShort*, PxConfigReadString*, PxConfigWriteChar*, PxConfigWriteDouble*, PxConfigWriteInt*, PxConfigWriteLLong*, PxConfigWriteLong*, PxConfigWriteShort*, PxConfigWriteString*()
** PxConfigWriteChar()  
*Write a character parameter in a configuration file*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteChar( const char *section,
                        const char *entry,
                        int format,
                        char value);

int PxConfigWriteCharCx( PxCfgContext_t *cx,
                         const char *section,
                         const char *entry,
                         int format,
                         char value);
```

**Arguments:**

- `cx`   
  *PxConfigWriteCharCx() only.* The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- `section`  
  The section to write `value` to.  
  If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.

- `entry`  
  The entry to write `value` to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.

- `format`  
  Determines how the `value` is formatted:
  - `PXCONFIG_FMT_CHAR_CHAR`  
    As a single character (letter/digit).
  - `PXCONFIG_FMT_CHAR_HEX`  
    As a 2-digit hex number with leading 0x.

- `value`  
  The value which the function writes to the file.
**Description:**  
These functions write a character parameter in the specified *section* and *entry* of a configuration file.

- The configuration file must have been opened by `PxConfigOpen*()` with a mode of PXCONFIG_WRITE.
- If the specified *section* doesn’t exist, it’s created.
- To create an empty section, call `PxConfigForceEmptySection*()`.

`PxConfigWriteChar()` writes to the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigWriteCharCx()` writes to the configuration file indicated by `cx`.

**Returns:**  
Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**  
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**  
`PxConfigOpen*()`, `PxConfigReadBool*()`, `PxConfigReadChar*()`, `PxConfigReadDouble*()`, `PxConfigReadInt*()`, `PxConfigReadLLong*()`, `PxConfigReadLong*()`
PxConfigWriteChar()

PxConfigReadShort*(), PxConfigReadString*(),
PxConfigWriteBool*(), PxConfigWriteDouble*(),
PxConfigWriteInt*(), PxConfigWriteLong*(),
PxConfigWriteLLong*(), PxConfigWriteShort*(),
PxConfigWriteString*()}
** PxConfigWriteDouble(), PxConfigWriteDoubleCx()  
Write an integer parameter in a configuration file

** Synopsis:**

```
#include <photon/PxProto.h>

int PxConfigWriteDouble( const char * section,
                          const char * entry,
                          int format,
                          int value );

int PxConfigWriteDoubleCx( PxCfgContext_t * cx,
                           const char * section,
                           const char * entry,
                           int format,
                           int value );
```

** Arguments:**

- **cx**  
  `PxConfigWriteDoubleCx()` only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- **section**  
The section to write `value` to.  
  If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.

- **entry**  
The entry to write `value` to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.

- **format**  
Must be PXCONFIG_FMT_DOUBLE.

- **value**  
The value the function writes to the file.

** Library:**

- **phexlib**

Chapter 14 • Px—Extended  1811
** PxConfigWriteDouble(), PxConfigWriteDoubleCx()  

**Description:** 
These functions write a double-precision float parameter in the specified section and entry of the configuration file.

- The configuration file must have been opened by PxConfigOpen*() with a mode of PXCONFIG_WRITE.
- If the specified section doesn’t exist, it’s created.
- To create an empty section, call PxConfigForceEmptySection*().

*PxConfigWriteDouble*() writes to the currently open configuration file opened by PxConfigOpen(), while *PxConfigWriteDoubleCx()* writes to the configuration file indicated by cx.

**Returns:**
Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadChar*(), PxConfigReadDouble*(), PxConfigReadInt*(), PxConfigReadLLong*(), PxConfigReadLong*(), PxConfigReadShort*(), PxConfigReadString*(), PxConfigWriteBool*(), PxConfigWriteChar*(), PxConfigWriteInt*(),

---

1812 Chapter 14 • Px—Extended
PxConfigWriteDouble(), PxConfigWriteDoubleCx()

PxConfigWriteLLong*(), PxConfigWriteLong*(), PxConfigWriteShort*(), PxConfigWriteString*()
**PxConfigWriteInt(), PxConfigWriteIntCx()**

Write an integer parameter in a configuration file

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteInt( const char *section, const char *entry, int format, int value );

int PxConfigWriteIntCx( PxCfgContext_t *cx, const char *section, const char *entry, int format, int value );
```

**Arguments:**

- **cx**
  - `PxConfigWriteIntCx()` only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- **section**
  - The section to write `value` to.
  - If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.

- **entry**
  - The entry to write `value` to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.

- **format**
  - Determines how the `value` is formatted:

    PXCONFIG_FMT_INT_DECIMAL
    - As a decimal number.
    PXCONFIG_FMT_INT_HEX
    - As a hex number with leading 0x.

- **value**
  - The value the function writes to the file.
**Library:** phexlib

**Description:**
These functions write an integer parameter in the specified *section* and *entry* of the configuration file.

- The configuration file must have been opened by `PxConfigOpen*()` with a mode of PXCONFIG_WRITE.
- If the specified *section* doesn’t exist, it’s created.
- To create an empty section, call `PxConfigForceEmptySection*()`.

`PxConfigWriteInt()` writes to the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigWriteIntCx()` writes to the configuration file indicated by `cx`.

**Returns:**
Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**
Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PxConfigOpen*()`, `PxConfigReadBool*()`, `PxConfigReadChar*()`, `PxConfigReadDouble*()`, `PxConfigReadInt*()`, `PxConfigReadLLong*()`, `PxConfigReadLong*()`,
 PxConfigWriteInt(), PxConfigWriteIntCx() 

 PxConfigReadShort*(), PxConfigReadString*(), 
 PxConfigWriteBool*(), PxConfigWriteChar*(), 
 PxConfigWriteDouble*(), PxConfigWriteLLong*(), 
 PxConfigWriteLong*(), PxConfigWriteShort*(), 
 PxConfigWriteString*()
**PxConfigWriteLLong(), PxConfigWriteLLongCx()**

Write a 64-bit long long integer parameter in a configuration file

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteLLong(   const char *section,
                           const char *entry,
                           int format,
                           long long value );

int PxConfigWriteLLongCx( PxCfgContext_t *cx,
                            const char *section,
                            const char *entry,
                            int format,
                            long long value );
```

**Arguments:**

- **cx**  
  `PxConfigWriteLLongCx()` only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- **section**  
  The section to write `value` to. If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.

- **entry**  
  The entry to write `value` to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.

- **format**  
  Determines how the `value` is formatted:

  - `PXCONFIG_FMT_INT_DECIMAL`  
    As a decimal number.

  - `PXCONFIG_FMT_INT_HEX`  
    As a hex number with leading 0x.

- **value**  
  The value the function writes to the file.
**PxConfigWriteLLong(), PxConfigWriteLLongCx()**

**Library:**

`phexlib`

**Description:**

These functions write a long long integer parameter in the specified section and entry of the configuration file.

- The configuration file must have been opened by `PxConfigOpen*()` with a mode of `PXCONFIG_WRITE`.
- If the specified section doesn’t exist, it’s created.
- To create an empty section, call `PxConfigForceEmptySection*()`.

`PxConfigWriteLLong()` writes to the currently open configuration file opened by `PxConfigOpen()`, while `PxConfigWriteLLongCx()` writes to the configuration file indicated by `cx`.

**Returns:**

Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadChar*(), PxConfigReadDouble*(), PxConfigReadInt*(), PxConfigReadLLong*(), PxConfigReadLong*(),`
PxConfigWriteLLong(), PxConfigWriteLLongCx()

PxConfigReadShort*(), PxConfigReadString*(),
PxConfigWriteBool*(), PxConfigWriteChar*(),
PxConfigWriteDouble*(), PxConfigWriteInt*(),
PxConfigWriteLong*(), PxConfigWriteShort*(),
PxConfigWriteString*()
**PxConfigWriteLong(), PxConfigWriteLongCx()**
*Write a long integer parameter in a configuration file*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteLong( const char *section,
                        const char *entry,
                        int format,
                        long value );

int PxConfigWriteLongCx( PxCfgContext_t *cx,
                          const char *section,
                          const char *entry,
                          int format,
                          long value );
```

**Arguments:**

- **cx**  
  *PxConfigWriteLongCx() only. The configuration file handle for the file you want to write to. This handle is returned by *PxConfigOpenCx().*

- **section**  
  The section to write *value* to.  
  If *section* matches a valid section name, it becomes the current section. If *section* is NULL, the function writes to the current section. If *section* doesn’t exist, the function creates it and makes it the current section.

- **entry**  
  The entry to write *value* to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.

- **format**  
  Determines how the *value* is formatted:

  - PXCONFIG_FMT_INT_DECIMAL  
    As a decimal number.
  - PXCONFIG_FMT_INT_HEX  
    As a hex number with leading 0x.

- **value**  
  The value the function writes to the file.
** PxConfigWriteLong(), PxConfigWriteLongCx()  

**Library:**  
phexlib  

**Description:**  
These functions write a long integer parameter in the specified section and entry of the configuration file.

- The configuration file must have been opened by PxConfigOpen*() with a mode of PXCONFIG_WRITE.
- If the specified section doesn’t exist, it’s created.
- To create an empty section, call PxConfigForceEmptySection*().

PxConfigWriteLong() writes to the currently open configuration file opened by PxConfigOpen(), while PxConfigWriteLongCx() writes to the configuration file indicated by cx.

**Returns:**  
Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**  
Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PxConfigOpen*(), PxConfigReadBool*(), PxConfigReadChar*(), PxConfigReadDouble*(), PxConfigReadInt*(), PxConfigReadLLong*(), PxConfigReadLong*(),

Chapter 14 • Px—Extended 1821
PxConfigWriteLong(), PxConfigWriteLongCx()

PxConfigReadShort*(), PxConfigReadString*(),
PxConfigWriteBool*(), PxConfigWriteChar*(),
PxConfigWriteDouble*(), PxConfigWriteInt*(),
PxConfigWriteLLong*(), PxConfigWriteShort*(),
PxConfigWriteString*()
**PxConfigWriteShort(), PxConfigWriteShortCx()**

Write a short integer parameter in a configuration file

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteShort( const char *section,
                        const char *entry,
                        int format,
                        short value );

int PxConfigWriteShortCx( PxCfgContext_t *cx,
                          const char *section,
                          const char *entry,
                          int format,
                          short value );
```

**Arguments:**

- `cx`    
  `PxConfigWriteShortCx()` only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.
- `section`
  The section to write `value` to.
  If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.
- `entry`
  The entry to write `value` to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.
- `format`
  Determines how the `value` is formatted:
  
  - `PXCONFIG_FMT_INT_DECIMAL`
    As a decimal number.
  - `PXCONFIG_FMT_INT_HEX`
    As a hex number with leading 0x.
**PxConfigWriteShort(), PxConfigWriteShortCx()**

**Library:**  
**phexlib**

**Description:**  
These functions write a short integer parameter in the specified 
section and entry of the configuration file.

- The configuration file must have been opened by *PxConfigOpen*() with a mode of PXCONFIG_WRITE.
- If the specified section doesn’t exist, it’s created.
- To create an empty section, call *PxConfigForceEmptySection*().

*PxConfigWriteShort()* writes to the currently open configuration file opened by *PxConfigOpen()* , while *PxConfigWriteShortCx()* writes to the configuration file indicated by cx.

**Returns:**  
Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**  
Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**  
*PxConfigOpen*(), *PxConfigReadBool*(), *PxConfigReadChar*(),  
*PxConfigReadDouble*(), *PxConfigReadInt*(),  
*PxConfigReadLLong*(), *PxConfigReadLong*(),
Chapter 14 • Px—Extended 1825

PxConfigWriteShort(), PxConfigWriteShortCx()

PxConfigReadShort*(), PxConfigReadString*(),
PxConfigWriteBool*(), PxConfigWriteChar*(),
PxConfigWriteDouble*(), PxConfigWriteInt*(),
PxConfigWriteLLong*(), PxConfigWriteLong*(),
PxConfigWriteString*()
** PxConfigWriteString(), PxConfigWriteStringCx()  
Write a string parameter in a configuration file 

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxConfigWriteString( const char *section,
const char *entry,
int format,
const char *value );

int PxConfigWriteStringCx( PxCfgContext_t *cx,
const char *section,
const char *entry,
int format,
const char *value );
```

**Arguments:**

- `cx`  
  `PxConfigWriteStringCx()` only. The configuration file handle for the file you want to write to. This handle is returned by `PxConfigOpenCx()`.

- `section`  
The section to write `value` to.
If `section` matches a valid section name, it becomes the current section. If `section` is NULL, the function writes to the current section. If `section` doesn’t exist, the function creates it and makes it the current section.

- `entry`  
The entry to write `value` to. If the entry already exists in the current section, it’s overwritten; otherwise, it’s added to the end of the section.

- `format`  
This parameter must be PXCONFIG_FMT_STRING.

- `value`  
The value the function writes to the file. The string must not contain a \n
**Library:**

`phexlib`
**PxConfigWriteString(), PxConfigWriteStringCx()**

**Description:**
These functions write a string parameter in the specified section and entry of the configuration file.

- The configuration file must have been opened by PxConfigOpen*() with a mode of PXCONFIG_WRITE.
- If the specified section doesn’t exist, it’s created.
- To create an empty section, call PxConfigForceEmptySection*().

You can write a single comment line in the section by specifying "##" for entry and the comment for value. For example:

```
PxConfigWriteString("My section", "##", PXCONFIG_FMT_STRING,
    "This is a comment");
```

**Returns:**
Pt_TRUE if the entry is written, otherwise Pt_FALSE

**Classification:**
Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 14 • Px—Extended 1827
PxConfigWriteString(), PxConfigWriteStringCx()

See also:

PxConfigOpen*, PxConfigReadBool*, PxConfigReadChar*,
PxConfigReadDouble*, PxConfigReadInt*,
PxConfigReadLLong*, PxConfigReadLong*,
PxConfigReadShort*, PxConfigReadString*,
PxConfigWriteBool*, PxConfigWriteChar*,
PxConfigWriteDouble*, PxConfigWriteInt*,
PxConfigWriteLLong*, PxConfigWriteLong*,
PxConfigWriteShort*()}
**PxGetImageExtensions()**

Query supported image types

**Synopsis:**

```c
#include <photon/PxImage.h>

int PxGetImageExtensions(char *extlist,
                          int len,
                          char const *prefix);
```

**Arguments:**

- **extlist**  A buffer the function fills with a space-separated list of image filename extensions supported by the currently loaded image handler plugins.
- **len**  The length of the extlist buffer.
- **prefix**  An optional prefix string the function can append to each extension to make it easy to create filter lists.

**Library:**

- `phexlib`

**Description:**

This function creates a space-separated list of image filename extensions supported by the currently loaded image handler plugins, and puts the list into `extlist`. You can easily create a filter list by passing a string containing an asterisk and period characters (`"*."`) as the prefix. This will create, for example, a list such as: `*.jpg
*.gif
*.bmp`.

**Returns:**

- 0  Success
- 1  Failure

Chapter 14  *Px—Extended  1829*
PxGetImageExtensions()

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PxLoadImage().

“Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
**PxLoadImage()**

*Read or query a graphic file*

**Synopsis:**

```c
#include <photon/PxImage.h>

PhImage_t * PxLoadImage( char *filename,
                        PxMethods_t *methods );
```

**Arguments:**

- `filename` The name of the graphic file that you want to load or query.
- `methods` A pointer to a `PxMethods_t` structure that lets you modify the behavior of the function; see below.

If this argument is NULL, the function loads the graphic file specified by `filename`.

**Library:**

`phexlib`

**Description:**

This function reads a graphic file into memory. Photon supports at least the BMP, GIF, JPEG, PCX, PNG, and SGI file formats; for a list of the supported formats, see `/usr/photon/dll/pi_io_*`

To draw an image, call `PgDrawPhImage()` or `PgDrawPhImagemx()`.

When you’re finished with the image, you can free the allocated members of the image structure by calling `PhReleaseImage()` after setting the image’s `flag` member to indicate which parts of the image should be freed. You can then free the `PhImage_t` structure. For more information, see `PhImage_t`.

**PxMethods_t**

The `PxMethods_t` structure alters how `PxLoadImage()` behaves. This structure is defined as:

```c
typedef struct pxmethods
```
PxLoadImage()

```
{ int flags;
    void *(*px_alloc)( long nbytes, int type );
    void *(*px_free)( void *memory, int type );
    void *(*px_error)( char *msg );
    void *(*px_warning)( char *msg );
    void *(*px_progress)( int percent );
    PhDim_t scale;
    void *colors;
    int ncolors;
} PxMethods_t;
```

The members are as follows:

**flags**

You can OR the following into `flags`:

- **PX_QUERY** — just query the graphic file. A `PhImage_t` structure is returned on success.
- **PX_LOAD** — read the graphic file into memory and convert it to a format that Photon can display.
- **PX_SUPPRESS_TAG** — don’t calculate tag IDs for the palette and image.
- **PX_DODITHER** — this option currently applies to JPEG images only. Perform FS dithering if the image loading library for the source type supports it.
- **PX_DIRECT_COLOR** — this option currently applies to JPEG images only. Load the graphic file as a 24-bit image if the loader library for the source type supports it.
- **PX_USECOLORS** — if loading a JPEG image, use the palette provided by the application instead of calculating one. The image will look much better on palette-based drivers.
- **PX_TRANSPARENT** — make the image transparent, using the detected transparent color and the image’s chroma scheme.
Memory allocation/deallocation routines that you supply. The deallocation routine is called only if the image can’t be loaded. The type can be one of the following:

- PX_NORMAL — the memory allocation is unspecific.
- PX_IMAGE — the memory allocation is for the image data.
- PX_PALETTE — the memory allocation is for the palette.

Your px_alloc function must return a pointer to the allocated memory.

An error routine that you supply. The loader calls this function if it encounters a fatal error while loading the graphic file. The msg argument is a pointer to an error string.

The loader frees all of the memory that it allocated before calling this function.

A warning routine that you supply. The loader calls this function if it encounters a nonfatal error while loading the graphic file. The msg argument is a pointer to a warning string.

A progress routine that you supply. The loader calls this function after it loads/decodes a scan line. The percent argument is a fixed point number in the following format:

```
####.####
```

The upper 16 bits are the whole portion; the lower 16 bits are the decimal portion.
PxLoadImage()  

You can call PxTerminate() in this function to abort the call to PxLoadImage() if something has gone wrong.

**scale**  
Not currently used.

**colors, ncolors**  
Used in conjunction with the PX_USECOLORS flag. The colors argument points to a palette, and ncolors indicates the number of valid entries in the palette.

PxLoadImage() doesn’t use the value that px_free, px_error, px_warning, and px_progress return. These functions can return NULL to avoid compiler warnings.

**Threads and PxLoadImage()**  
As described in “Threads” in the Parallel Operations chapter of the Photon Programmer’s Guide, you need to be careful when using the Photon library in a multi-threaded program; you need to call PtEnter() and PtLeave() around any calls to the Photon library.

However, PxLoadImage() is completely separate from the rest of the Photon library. It’s not thread-safe, so you do need to make sure that only one thread at a time is trying to use it; but you could use your own mutex instead of the PtEnter() lock, and if you know that you have only one thread that ever loads images, you don’t even need that.

Of course, if the methods that you pass to PxLoadImage() use any Photon calls such as PgShmemCreate(), you need to call PtEnter() and PtLeave() around them.

This can create problems if you’re using your own mutex and you have a Photon callback that wants to call PxLoadImage(): if one thread locks your mutex and then blocks in PtEnter() while the Photon thread invokes the callback, and the callback tries to lock your mutex without calling PtLeave(), you have a deadlock. You have to design carefully to avoid such situations — but that’s normal in multi-threading.
### PxLoadImage()

**Returns:**

A pointer to a `PhImage_t` structure, or NULL if an error occurs.

**Examples:**

This example can use either shared or normal memory. The advantage of using shared memory is that it takes less time to draw the image. If you’re not using shared memory, increasing the draw buffer size causes more drawing to be buffered before being sent to the graphics driver; this isn’t as important if you’re using shared memory.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <malloc.h>
#include <assert.h>
#include <ctype.h>
#include <signal.h>
#include <Ph.h>
#include <Pt.h>

#include <photon/PxImage.h>

void *memory_allocate( long nbytes, int type );
void *memory_free( void *memory, int type );
void *warning( char *msg );
void *error( char *msg );
void *progress( int percent );

int UseShmem = 1;

int main( int argc, char *argv[] )
{
    int c;
    PtArg_t arg[5];
    PtWidget_t *window;
    char fname[255] = { 0 };
    int Query = 0;
    PhImage_t *img;
    PxMethods_t methods;

    /* initialize widget library and attach to Photon */
    if( PtInit( NULL ) )
        exit( EXIT_FAILURE );

    /*...*/
```
while( ( c = getopt( argc, argv, "f:QS" ) ) != -1 ) {
    switch( c ) {
        case 'f': // filename
            strncpy(fname, optarg, 200 );
            break;
        case 'Q': // query file
            Query = 1;
            break;
        case 'S':
            UseShmem^=1;
            break;
    }
}
memset( &methods, 0, sizeof( PxMethods_t ) );
methods.px_alloc = memory_allocate;
methods.px_free = memory_free;
methods.px_warning = warning;
methods.px_error = error;
methods.px_progress = progress;
if( Query )
    methods.flags |= PX_QUERY;
else
    methods.flags |= PX_LOAD;
if( ( img = PxLoadImage( fname, &methods ) ) == NULL ) {
    fprintf( stderr, "Error loading/query %s\n",
             fname );
    PtExit( EXIT_FAILURE );
}
/* Make sure PhReleaseImage() releases any allocated members of the PhImage_t structure. */
img->flags |= Ph_RELEASE_IMAGE_ALL;
if( Query ) {
    printf( "Image width: %d\n", img->size.w );
    printf( "Image height: %d\n", img->size.h );
    printf( "Image BPL: %d\n", img->bpl );
    printf( "Image colors: %d\n", img->colors );
    printf( "Image type: %d\n", img->type );
    PtExit( EXIT_SUCCESS );
}
PxLoadImage()

/* increase the draw buffer */
PgSetDrawBufferSize( 0x8000 );

/* create a window */
PtSetArg( &arg[0], Pt_ARG_DIM, &img->size, 0 );
PtSetArg( &arg[1], Pt_ARG_WINDOW_TITLE,
          "Photon Image Viewer", 0 );
window = PtCreateWidget( PtWindow, Pt_NO_PARENT, 2, arg );

/* Create a label widget with the image. Remember that
the widget creates a copy of the PhImage_t structure.
The widget doesn’t copy data pointed to by the
PhImage_t members. */
PtSetArg( &arg[0], Pt_ARG_LABEL_TYPE, Pt_IMAGE, 0 );
PtSetArg( &arg[1], Pt_ARG_LABEL_IMAGE, img, 0 );
PtCreateWidget( PtLabel, window, 2, arg );

/* Free the PhImage_t structure (but not its contents). */
free( img );
PtRealizeWidget( window );
PtMainLoop();
return EXIT_SUCCESS;

void *memory_allocate( long nbytes, int type )
{
    if( type == PX_IMAGE && UseShmem ) {
        return( PgShmemCreate( nbytes, NULL ) );
    }
    else {
        return( calloc( 1, nbytes ) );
    }
}

void *memory_free( void *memory, int type )
{
    if( type == PX_IMAGE && UseShmem ) {
        PgShmemDestroy( memory );
    }
    else {
        free( memory );
    }
    return NULL;
}
void *warning( char *msg )
{
    printf( "%s\n", msg );
    return NULL;
}

void *error( char *msg )
{
    printf( "%s\n", msg );
    PtExit( EXIT_FAILURE );
    return NULL;
}

void *progress( int percent )
{
    printf( "Load Status: %d.%d percent\n",
            percent >> 16, percent & 0xffff );
    return NULL;
}

Classification:
Photon

Safety

Interrupt handler  No
Signal handler      No
Thread              No

See also:
PgDrawPhImage*( ), PgDrawPhImageRect*( ),
PgDrawRepPhImage*( ), PgSetPalette(), PgSetFillColor(),
PgSetTextColor(), PgShmemCleanup(), PhCreateImage(),
PhImage_t, PhMakeGhostBitmap(), PhMakeTransBitmap(),
PhMakeTransparent(), PhReleaseImage(), PtCRC(), PtCRCValue(),
PxTerminate()
"Images" in the Raw Drawing and Animation chapter of the Photon
Programmer’s Guide
**PxTerminalBuildCharsets()**

Create character set tables based on translation tables

**Synopsis:**

```c
#include <photon/PxTerminal.h>

PtTerminalCharsets_t *PxTerminalBuildCharsets(
    PxTerminalCsNames_t const *names);
```

**Library:**

`phexlib`

**Description:**

This function is an alternative to creating charset tables by hand. It creates a `PtTerminalCharsets_t` structure (see the Photon Widget Reference) based on Photon character translation files (see `PxTranslateSet()`).

The `PxTerminalCsNames_t` structure is defined as follows:

```c
typedef struct {
    char const *AnsiCharsetName;
    char const *InternalCharsetName;
    char const *FontCharsetName;
    ...
} PxTerminalCsNames_t;
```

The `AnsiCharsetName` and `InternalCharsetName` members can be either NULL or the name of a supported character set. A NULL maps directly to a NULL in the resulting `PtTerminalCharsets_t` structure.

The `FontCharsetName` can be one of:

- NULL — no font translation
- The name of an 8-bit character encoding
- The special value Px_TERMINAL_UNICODE_FONT.

This function puts the resulting structure and all the tables in a single allocated block of memory. After it’s no longer needed, you can simply `free()` it.
Returns:
A pointer to the resulting \texttt{PtTerminalCharsets\_t} structure.

Classification:
Photon

\begin{tabular}{l|c}
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\end{tabular}

See also:
\begin{itemize}
\item \texttt{PxTerminalLoadCharsets()}, \texttt{PxTerminalSaveCharsets()}
\item \texttt{PtTerminal, PtTerminalCharsets\_t} in the Photon Widget Reference
\end{itemize}
**PxTerminalLoadCharsets()**

Load character-set information from a file

**Synopsis:**
```
#include <photon/PxTerminal.h>

PtTerminalCharsets_t *PxTerminalLoadCharsets(
    const char *filename,
    PtTerminalCsNames_t *names);
```

**Library:**
phexlib

**Description:**

`PxTerminalLoadCharsets()` loads character-set information from the given file. The `names` argument, if not NULL, points to a structure that the names of the character sets will be stored in.

This function puts the resulting structure and all the tables and strings in a single allocated block of memory. After it’s no longer needed, you can simply `free()` it. Note that this invalidates the strings stored in `*names`.

**Returns:**

A pointer to a `PtTerminalCharsets_t` structure ready for use with a `PtTerminal` widget.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PxTerminalBuildCharsets(), PxTerminalSaveCharsets()

PtTerminal, PtTerminalCharsets_t in the Photon Widget Reference
**PxTerminalSaveCharsets()**

Save character-set information in a file

**Synopsis:**

```c
#include <photon/PxTerminal.h>

int PxTerminalSaveCharsets(
    PtTerminalCharsets_t *charsets,
    PxTerminalCsNames_t *names,
    const char *filename);
```

**Library:**

`phexlib`

**Description:**

This function saves the character-set information in the given file. It's your responsibility to make sure that the information in `charsets` is consistent with the information in `names` — generating both with the same call to `PxTerminalBuildCharsets()` is a good way to ensure consistency.

**Returns:**

0  Success.

-1  An error occurred; `errno` is set.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PxTerminalBuildCharsets(), PxTerminalLoadCharsets()

PtTerminal, PtTerminalCharsets_t in the Photon Widget Reference
**PxTerminate()**

*Terminate a PxLoadImage() call*

**Synopsis:**

```c
#include <photon/PxImage.h>

void PxTerminate( PhImage_t *image );
```

**Arguments:**

- `image` The image that `PxLoadImage()` is trying to load or query.

**Library:**

- `phexlib`

**Description:**

This function terminates a call to `PxLoadImage()`. You can call `PxTerminate()` in the progress callback for `PxLoadImage()` to abort if something goes wrong with the image load or query.

**Classification:**

- Photon

**Safety**

- Interrupt handler No
- Signal handler No
- Thread No

**See also:**

- `PxLoadImage()`
- “Images” in the Raw Drawing and Animation chapter of the Photon Programmer’s Guide
Synopsis:

```
#include <photon/PxProto.h>

int PxTranslateFromUTF( struct PxTransCtrl *ctrl,
                        const char *src,
                        int maxsrc,
                        int *srctaken,
                        char *dst,
                        int maxdst,
                        int *dstmade );
```

Library:

`phexlib`

Description:

`PxTranslateFromUTF()` is used to translate a block of characters from UTF-8 (the internal Photon character set). The `ctrl` parameter specifies the encoding of the destination characters and must be the pointer returned by a previous call to `PxTranslateSet()` to install this encoding.

The parameters are as follows:

- `src` is a pointer to the buffer containing the source UTF-8 characters. If this is NULL, no translation is performed, and the function returns the worst-case number of bytes required to hold a character in the current encoding.

- `maxsrc` is the length of the contents in the source buffer (in bytes). No more than this number of bytes are read.

- `dst` is a pointer to the buffer where the encoded characters should be placed. If this is NULL, no data is copied but the translation is still performed to calculate the length required to store the converted data.

- `maxdst` is the length of the destination buffer (in bytes). No more than this number of bytes will be written; if this is 0, the
**PxTranslateFromUTF()**

buffer is assumed to be large enough to hold the entire encoding.

src
taken
this must point to an integer, which will be updated to reflect the number of bytes consumed from the source buffer src. This value may be smaller than maxsrc (if the given destination buffer would overflow or if the final input character of a multibyte UTF-8 sequence is incomplete).

dst
made
this must point to an integer, which will be updated to reflect the number of bytes produced (or would be produced) into the destination buffer dst. This value may be smaller than maxdst (if the given source buffer is exhausted or if the final output character of a multibyte sequence would be incomplete).

**Returns:**

0 Success.

-1 An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

 PxTranslateList(), PxTranslateSet(), PxTranslateStateFromUTF(), PxTranslateStateToUTF(), PxTranslateToUTF(), PxTranslateUnknown()  

 Unicode Multilingual Support in the Photon Programmer’s Guide  

 mbtowc(), wctomb() in the QNX Neutrino Library Reference
**PxTranslateList()**

Create a list of all supported character translations

**Synopsis:**
```
#include <photon/PxProto.h>

int PxTranslateList( PtWidget_t *widget,  
                    char const *none );
```

**Library:**
phexlib

**Description:**
This function may be used to create a list or combobox of all supported character translations. It takes as a parameter a pointer to a list-type widget and sets its `Pt_ARG_ITEMS` resource to be the list of translations. These translations are read from the file `/usr/photon/translations/charsets`, using the `Description` entry as the item text.

If non-NULL, the `none` parameter points to a string to be added to the top of the list. This allows you to specify an entry such as None or Default. Your application will need to know how to handle this entry when it’s chosen.

This list may then be used at run time to alter the current translation dynamically. The program should call `PxTranslateSet()` with the selected description text to install the new encoding.

**Returns:**
The number of items placed in the list, or -1 on error.

**Classification:**
Photon

**Safety**

Interrupt handler No

continued...
**PxTranslateList()**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

*PxTranslateFromUTF(), PxTranslateSet(),
PxTranslateStateFromUTF(), PxTranslateStateToUTF(),
PxTranslateToUTF(), PxTranslateUnknown()*

Unicode Multilingual Support in the Photon *Programmer’s Guide*
**PxTranslateSet()**
*Install a new character set translation*

**Synopsis:**
```
#include <photon/PxProto.h>

struct PxTransCtrl *PxTranslateSet(
    struct PxTransCtrl *ctrl,
    const char *charset);
```

**Library:**

```
phexlib
```

**Description:**

*PxTranslateSet()* installs a new character set translation. The *ctrl*
argument, if non-NULL, is a pointer to the control structure for a
character set translation returned from a previous call to
*PxTranslateSet()*. The translation it specifies is disabled, and any
resources it uses are released.

*PxTranslateSet()* searches the
`/usr/photon/translations/charsets` configuration file for the
translation specified by *charset*; it may be an entry section name, one
of the *Alias* entries or the *Description* entry. The *charset* name is
usually selected by the user (see *PxTranslateList()*)) or from an
external specification (for example, the *charset=* field of the
Content-type MIME / HTTP header).

If the specified *charset* is found, resources are allocated as required,
and any necessary data files are loaded into memory. The following
special values of *charset* are recognized:

- **NULL** — release any resources used by the existing encoding
  scheme without installing a new translation

- an empty string (**"***) — install a simple byte-copy scheme

The translation routines are provided in the Photon library *phexlib*,
with prototypes in `<photon/PxProto.h>`. 

1852  Chapter 14  •  Px—Extended
**Returns:**

A pointer to a translation control structure, which should be passed to subsequent translation routines and to future calls to `PxTranslateSet()`.

**Examples:**

This sample program converts characters from stdin (encoded in a character set specified to the program as its only argument) to stdout (in UTF-8). Note that a 256-byte buffer is allocated for input and a MB_LEN_MAX * 256 bytes (the worst-case UTF-8 encoding for that number of input bytes) created for output. Alternatively, a call to `PxTranslateToUTF()` with a NULL source buffer could be used to work out the bytes-per-character requirements (we exploit the fact that we already know this number for UTF-8 encoding).

```c
#include <stdio.h>
#include <stdlib.h>
#include <photon/PxProto.h>

#define BUFFER 256

int main(int argc, char *argv[]) {
    struct PxTransCtrl *trans;
    char *code, *utf;
    int srclen, dstlen;

    if (argc < 2) {
        fprintf(stderr, "specify translation charset\n");
        return(1);
    }

    if ((trans = PxTranslateSet(NULL, argv[1])) == NULL) {
        fprintf(stderr, "unknown translation charset '%s'\n",
            argv[1]);
        return(1);
    }

    if ((code = malloc(BUFFER)) == NULL ||
        (utf = malloc(BUFFER * MB_LEN_MAX)) == NULL) {
        fprintf(stderr,
            "unable to allocate %d-byte translation buffers\n",
            BUFFER);
        return(1);
    }

    while ((srclen = fread(code, sizeof(char), BUFFER, stdin)) {
        ...
    }

    return(0);
}
```
`PxTranslateSet()`

```c
if ((dstlen = PxTranslateStateToUTF(trans, code,
   srclen, NULL, utf,
   BUFFER * MB_LEN_MAX)) == -1) {
   fprintf(stderr, "invalid encoding sequence\n");
   return(1);
}
fwrite(utf, dstlen, sizeof(char), stdout);

return EXIT_SUCCESS;
```

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

`PxTranslateList()`, `PxTranslateFromUTF()`,
`PxTranslateStateFromUTF()`, `PxTranslateStateToUTF()`,
`PxTranslateToUTF()`, `PxTranslateUnknown()`

Unicode Multilingual Support in the Photon *Programmer’s Guide*
Synopsis:

```c
#include <photon/PxProto.h>

int PxTranslateStateFromUTF(
    struct PxTransCtrl *ctrl,
    const char *src,
    int maxsrc,
    int *consumed,
    char *dst,
    int maxdst);
```

Library:

phexlib

Description:

This function is similar to `PxTranslateFromUTF()` except that it uses an internal state buffer.

Since many encodings are multibyte, it’s possible (in cases where input data is being provided from a file or socket) for a conversion to end in the middle of a multibyte sequence or for the output buffer to be too small to hold the complete encoding of the final character. This routine buffers any partial encoding, using those bytes as the start of a character sequence for the next `PxTranslateStateFromUTF()` call.

This routine uses an appropriately sized temporary overflow buffer, allocated by the `PxTranslateSet()` routine.

The parameters `src` and `maxsrc` specify the input UTF-8 buffer; `dst` and `maxdst` specify the output buffer. These have the same meaning as in the `PxTranslateFromUTF()` function. The `consumed` parameter is updated with the number of bytes converted from the source buffer; this may be NULL if this information isn’t required (i.e. if the source is always correctly encoded and the destination buffer is always sufficiently large).
**PxTranslateStateFromUTF()**

**Returns:**

The number of bytes produced in the destination buffer, or -1 on error.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PxTranslateList(), PxTranslateFromUTF(), PxTranslateSet(), PxTranslateStateToUTF(), PxTranslateToUTF(), PxTranslateUnknown()*

Unicode Multilingual Support in the Photon *Programmer’s Guide*
**PxTranslateStateToUTF()**

_Synopsis:_

```
#include <photon/PxProto.h>

int PxTranslateStateToUTF( struct PxTransCtrl *ctrl,
   const char *src,
   int maxsrc,
   int *consumed,
   char *dst,
   int maxdst );
```

_**Library:**_

_phexlib_

_Description:_

This function is similar to `PxTranslateToUTF()` except that it uses an internal state buffer. Since many encodings are multibyte, it’s possible (in cases where input data is being provided from a file or socket) for a conversion to end in the middle of a multibyte sequence or for the output buffer to be too small to hold the complete encoding of the final character. This routine buffers any partial encoding, using those bytes as the start of a character sequence for the next `PxTranslateStateToUTF()` call.

This routine uses an appropriately sized temporary overflow buffer, allocated by the `PxTranslateSet()` routine.

The parameters `src` and `maxsrc` specify the input buffer; the parameters `dst` and `maxdst` specify the output UTF-8 buffer. These have the same meaning as in the `PxTranslateToUTF()` function. The `consumed` parameter will be updated with the number of bytes converted from the source buffer; this may be NULL if this information isn’t required (i.e. if the source is always correctly encoded and the destination buffer is always sufficiently large).
**PxTranslateStateToUTF()**

**Returns:**

The number of bytes produced in the destination buffer, or -1 on error.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PxTranslateList(), PxTranslateFromUTF(), PxTranslateSet(), PxTranslateStateFromUTF(), PxTranslateToUTF(), PxTranslateUnknown()

Unicode Multilingual Support in the Photon *Programmer’s Guide*
**PxTranslateToUTF()**

*Translate characters to UTF-8*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxTranslateToUTF( struct PxTransCtrl *ctrl,
                     const char *src,
                     int maxsrc,
                     int *srctaken,
                     char *dst,
                     int maxdst,
                     int *dstmade );
```

**Library:**

`phexlib`

**Description:**

This function translates a block of characters to Unicode UTF-8 (the internal Photon character set). The `ctrl` parameter specifies the encoding of the source characters and must be the pointer returned by a previous call to `PxTranslateSet()` to install this encoding.

The parameters are as follows:

- `src` A pointer to the buffer containing the source characters. If this is NULL, no translation is performed and the function returns the worst-case number of bytes required to hold a character in UTF-8 (MB_LEN_MAX == 3).

- `maxsrc` The length of the contents in the source buffer (in bytes). No more than this number of bytes are read.

- `dst` A pointer to the buffer where the UTF-8 characters should be placed. If this is NULL, no data is copied but the translation is still performed to calculate the length required to store the converted data.

- `maxdst` The length of the destination buffer (in bytes). No more than this number of bytes will be written; if this is 0, the buffer is assumed to be large enough to hold the entire encoding.
**PxTranslateToUTF()**

*src* 
This must point to an integer, which will be updated to reflect the number of bytes consumed from the source buffer *src*. This value may be smaller than *maxsrc* (if the given destination buffer would overflow or if the final input character of a multibyte sequence is incomplete).

*dst* 
This must point to an integer, which will be updated to reflect the number of bytes produced (or would be produced) in the destination buffer *dst*. This value may be smaller than *maxdst* (if the given source buffer is exhausted or if the final output character of a multibyte UTF-8 sequence would be incomplete).

**Returns:**

- 0  Success.
- -1  An error occurred.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

*PxTranslateList(), PxTranslateFromUTF(), PxTranslateSet(), PxTranslateStateFromUTF(), PxTranslateStateToUTF(), PxTranslateUnknown()*

Unicode Multilingual Support in the Photon *Programmer’s Guide*
In the QNX Neutrino Library Reference, the functions `mbtowc()` and `wctomb()` are described.
**PxTranslateUnknown()**

*Control how unknown encodings are handled*

**Synopsis:**

```c
#include <photon/PxProto.h>

int PxTranslateUnknown( struct PxTransCtrl *ctrl,
                        uint16_t to,
                        uint16_t from );
```

**Library:**

`phexlib`

**Description:**

This function controls the behavior of the encoding routines when they encounter an invalid byte sequence or a character that can’t be represented in the current encoding scheme.

The `ctrl` argument is a pointer to the control structure for a character set translation returned from a previous call to `PxTranslateSet()`.

The `to` argument is used when converting to UTF-8 by calling `PxTranslateToUTF()` or `PxTranslateStateToUTF()`. If `to` is 0 (the default) and an invalid encoding is encountered, the translation is halted and returns an error. If `to` is nonzero, it’s the Unicode character to insert into the translation instead of the invalid one.

The `from` argument is similar, but is used when converting from UTF-8 by calling `PxTranslateFromUTF()` or `PxTranslateStateFromUTF()`.

**Returns:**

0  Success.

-1  An error occurred.

**Classification:**

Photon
**PxTranslateUnknown()**

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PxTranslateFromUTF()`, `PxTranslateSet()`, `PxTranslateStateFromUTF()`, `PxTranslateStateToUTF()`, `PxTranslateToUTF()`

Unicode Multilingual Support in the Photon *Programmer’s Guide*
The functions in this chapter deal with realtime timers.
RtTimerCreate()
Create a realtime timer

Synopsis:
#include <photon/realtime/RtTimer.h>

RtTimer_t *RtTimerCreate(clockid_t clock_id, int prio, RtTimerCbF_t *cb, void *data);

Arguments:

clock_id The clock source; see the documentation for
timer_create() in the QNX Neutrino Library Reference
for the possible values.

prio The priority of the Photon pulse that will be used when
the timer expires. If priority is -1, the pulse’s priority is
the same as that of the calling process.

cb A pointer to the function that you want called when the
timer expires. The RtTimerCbF_t type is:

typedef int RtTimerCbF_t(RtTimer_t *timer,
        void *data);

data A pointer to a block of data that’s passed to the cb
function as the data argument.

Library:

ph

Description:

This function creates a realtime timer. The timer is disabled when
created; it isn’t enabled until you call RtTimerSetTime().
**RtTimerCreate()**

**Returns:**
A pointer to a `RtTimer_t` structure to be passed to the other routines dealing with realtime timers, or NULL if an error occurred.

**Classification:**
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**
- `RtTimerDelete()`, `RtTimerGetTime()`, `RtTimerSetTime()`
- “Timers” in the Working with Code chapter of the Photon *Programmer’s Guide*
- `timer_create()` in the QNX Neutrino *Library Reference*
**RtTimerDelete()**

*Delete a realtime timer*

**Synopsis:**

```c
#include <photon/realtime/RtTimer.h>

void RtTimerDelete( RtTimer_t *timer );
```

**Arguments:**

- `timer` A pointer to a `RtTimer_t` that was created by a call to `RtTimerCreate()`.

**Library:**

`ph`

**Description:**

This function deletes the realtime timer identified by the structure pointed to by `timer`.

**Classification:**

Photon

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`RtTimerCreate()`, `RtTimerGetTime()`, `RtTimerSetTime()`

“Timers” in the Working with Code chapter of the Photon Programmer’s Guide

`timer_delete()` in the QNX Neutrino Library Reference
**RtTimerGetTime()**

Get the time remaining on a realtime timer

**Synopsis:**

```c
#include <photon/realtime/RtTimer.h>

int RtTimerGetTime( RtTimer_t *timer,
                    struct itimerspec *value );
```

**Arguments:**

- `timer`: A pointer to a `RtTimer_t` that was created by a call to `RtTimerCreate()`.
- `value`: A pointer to a `itimerspec` structure in which to store the remaining time.

**Library:**

`ph`

**Description:**

This function gets the time remaining on the realtime timer identified by the structure pointed to by `timer`. The time remaining is put into the structure pointed to by `value`.

**Returns:**

- `0`: Success.
- `-1`: An error occurred; `errno` is set.

**Errors:**

- `EINVAL`: The given timer isn’t attached to the calling process.

**Classification:**

`Photon`
**RtTimerGetTime()**

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*RtTimerCreate(), RtTimerDelete(), RtTimerSetTime()*

“Timers” in the Working with Code chapter of the Photon
*Programmer’s Guide*

timer_gettime() in the QNX Neutrino *Library Reference*
RtTimerSetTime()
Set the expiration time for a realtime timer

Synopsis:
#include <photon/realtime/RtTimer.h>

int RtTimerSetTime(RtTimer_t *timer,
                    int flags,
                    struct itimerspec *value,
                    struct itimerspec *ovalue);

Arguments:

	timer   A pointer to a RtTimer_t that was created by a call to
               RtTimerCreate().
	nflags   See timer_settime() in the QNX Neutrino Library
               Reference.

tvalue   A pointer to a itimerspec structure that defines the new
               expiration time.
	novalue  If non-NULL, a pointer to a itimerspec structure in
               which to store the old expiration time.

Library:

   ph

Description:
This function sets the expiration time for the timer identified by the
structure pointed to by timer to the time specified by value.

Returns:

   0     Success.

   -1    An error occurred; errno is set.
**RtTimerSetTime()**

**Errors:**

See *timer_settime()* in the QNX Neutrino Library Reference.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*RtTimerCreate(), RtTimerDelete(), RtTimerGetTime()*

*timer_settime()* in the QNX Neutrino Library Reference

“Timers” in the Working with Code chapter of the Photon Programmer’s Guide
Chapter 16

utf8—UTF-8 Character
The Photon libraries provide multibyte string functions that are useful if you’re using UTF-8 character strings. Use the functions described in this chapter instead of the usual 8-bit-character functions such as `strlen()` that are described in the QNX Neutrino Library Reference.
utf8len()
Count the bytes in a UTF-8 character

Synopsis:

```c
#include <utf8.h>

int utf8len( const char *s,
             size_t n );
```

Arguments:

- `s` A pointer to a UTF-8 character.
- `n` The maximum number of bytes to count.

Library:

ph

Description:

The `utf8len()` function counts the number of bytes in the UTF-8 character pointed to by `s`, to a maximum of `n` bytes, if `n` is nonzero.

This function is similar to `mblen()`, except that:

- `utf8len()` isn’t affected by the current locale.
- The `s` argument isn’t allowed to be NULL.
- You can pass 0 for `n` if you know that `s` points to a null-terminated string (i.e. 0 is equivalent to, but more efficient than, `strlen(s)`).
- `utf8len()` returns -1 if `s` points to an invalid byte sequence. If `n` is nonzero and the `n` bytes pointed to by `s` look like an incomplete but potentially valid character, the function returns the negative total length of that (complete) character (this is in the range from -2 to -UTF8_LEN_MAX).

Returns:

- `0` `s` points to the null character.
- `> 0` The number of bytes that comprise the multibyte character (if the next `n` or fewer bytes form a valid multibyte character).
utf8len()

-1 The n-byte sequence that s points to isn’t a valid (beginning of a) UTF-8-encoded character.

Other negative value

The n bytes pointed to by s could be the initial bytes of a valid UTF-8 sequence.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

utf8strbilen(), utf8strlen(), utf8strnlen()

Unicode Multilingual Support in the Photon Programmer’s Guide

mblen() in the QNX Neutrino Library Reference
utf8strblen()

Find the number of UTF-8 characters in part of a string

Synopsis:

```c
#include <utf8.h>

int utf8strblen( char const *text,
                 int max_bytes,
                 int *bytes );
```

Arguments:

- `text` A UTF-8 string.
- `max_bytes` The maximum number of bytes to count.
- `bytes` A pointer to a location where `utf8strblen()` stores the number of characters in the specified portion of the string.

Library:

```
ph
```

Description:

The `utf8strblen()` function returns the number of UTF-8 characters made up of `max_bytes` bytes in the string `text`, and sets `bytes` to the number of bytes used to compose the number of UTF-8 characters returned.

The `bytes` argument won’t equal `max_bytes` if there are fewer than `max_bytes` bytes in the string, or if the last byte doesn’t fall at the end of a UTF-8 character.

Returns:

The number of UTF-8 characters made up of `max_bytes` bytes in the string `text`. 
utf8strblen()

Classification:
Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:
utf8len(), utf8strlen(), utf8strnlen()

Unicode Multilingual Support in the Photon Programmer’s Guide
utf8strchr()
Search for a UTF-8 character in a string

Synopsis:

```
#include <utf8.h>

char * utf8strchr( char const *string,
                  char const *mbchar,
                  int *count );
```

Arguments:

- `string` A pointer to the string to search.
- `mbchar` The character to look for.
- `count` A pointer to the location where utf8strchr() stores the number of UTF-8 characters the matching character is from the start of the string.

Library:

`ph`

Description:

The utf8strchr() function searches for a character in `string` that matches `mbchar`. If such a match occurs in `string`, `count` (if provided) is set to the number of UTF-8 characters the matching character is from the beginning of the string. For example, if `mbchar` matches the first character in `string`, `count` is set to 0. If `mbchar` matches the second character in `string`, `count` is set to 1.

A pointer to the beginning of the matching character within `string` is returned. If no match is found, the function returns NULL and doesn’t change `count`.

Returns:

A pointer to the matching character, if found, or NULL if not found.
utf8strchr()

Examples:

```c
#include <Pt.h>

int main()
{
    char string[] = "Hello there: äîêü found";
    char mbchar[] = "é";
    int count;
    char *p;

    if( (p = utf8strchr( string, mbchar, &count ) ) )
        printf("Character found: character offset %d\n byte offset %d.\n", count, p - string);
    else
        printf("Not found.\n");
    return EXIT_SUCCESS;
}
```

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

utf8stricr(), utf8strnchr(), utf8strnichr(), utf8strstrchr(), utf8strrchr(), utf8strirchr()

Unicode Multilingual Support in the Photon *Programmer's Guide*
utf8strichr()
Search for a UTF-8 character in a string, ignoring case

Synopsis:

```c
char *utf8strichr( char const *string,
    char const *mbchar,
    int *count );
```

Arguments:

- `string` A pointer to the string to search.
- `mbchar` The character to look for.
- `count` A pointer to the location where `utf8strrchr()` stores the number of UTF-8 characters the matching character is from the start of the search.

Library:

```
ph
```

Description:

The `utf8strichr()` searches for a character in `string` that matches `mbchar` disregarding the case. If such a match occurs in `string`, then `count` (if provided) is set to the number of characters spanned to find the match.

Returns:

A pointer to the beginning of the matching character within `string`, or NULL if no match is found.

Classification:

Photon

Safety

Interrupt handler  No

continuing...
## utf8strichr()

### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### See also:

- `utf8strchr()`, `utf8strnchr()`, `utf8strnichr()`, `utf8strrchr()`, `utf8strirchr()`

Unicode Multilingual Support in the Photon *Programmer’s Guide*
UTF8STRIRCHR()
Search backwards for a UTF-8 character in a string, ignoring case

Synopsis:

```
char *utf8strirchr( char const *string_base,
    char const *start_char,
    char const *mbchar,
    int *count);
```

Arguments:

- `string_base` A pointer to the beginning of the string to search.
- `start_char` A pointer to the character in the string from which to search.
- `mbchar` The character to look for.
- `count` A pointer to the location where `utf8strirchr()` stores the number of characters spanned to find `mbchar`.

Library:

`ph`

Description:

The `utf8strirchr()` function searches backwards from `start_char` to `string_base`, inclusive, for a character that matches `mbchar`, regardless of case. If such a match occurs, `count` (if provided) is set to the number of characters spanned to find the match.

Returns:

A pointer to the beginning of the matching character within `string_base`, or NULL if no match is found.

Classification:

Photon
utf8strirchr()

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

utf8strchr(), utf8strichr(), utf8strnchr(), utf8strnichr(), utf8strrrchr()

Unicode Multilingual Support in the Photon *Programmer’s Guide*
utf8strlen()
Find the length of a UTF-8 character string

Synopsis:
```c
#include <utf8.h>

int utf8strlen( char const *text,
                int *bytes );
```

Arguments:
- `text` A UTF-8 string.
- `bytes` A pointer to a location where `utf8strlen()` stores the number of bytes in the string.

Library:
```
ph
```

Description:
The `utf8strlen()` function returns the number of UTF-8 characters in the string `text`, and sets `bytes` to the number of bytes in `text`.

Returns:
The number of UTF-8 characters in the string.

Classification:
```
Photon
```

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1888  Chapter 16  utf8—UTF-8 Character
utf8strlen()
utf8strnchr()
Search for a UTF-8 character in part of a string

Synopsis:
```c
#include <utf8.h>

char * utf8strnchr( char const *string, 
            char const *mbchar, 
            int num, 
            int *count );
```

Arguments:
- `string` A pointer to the string to search.
- `mbchar` The character to look for.
- `num` The maximum number of characters to search in the string.
- `count` A pointer to the location where `utf8strnchr()` stores the number of UTF-8 characters the matching character is from the start of the string.

Library:
- `ph`

Description:
This function searches for a UTF-8 character in `string` that matches `mbchar`. The match must occur in the first `num` UTF-8 characters. If such a match is found, `count` (if provided) is set to the number of UTF-8 characters the matching character is from the beginning of the string. For example, if `mbchar` matches the first character in `string`, `count` is set to 0. If `mbchar` matches the second character in `string`, `count` is set to 1.

Returns:
A pointer to the beginning of the matching character within `string`. If no match is found within `num` characters, the function returns NULL and doesn’t change `count`.

1890 Chapter 16 ● utf8—UTF-8 Character
utf8strnchr()

Examples:

Search the first 5 UTF-8 characters for a match to the provided UTF-8 character:

```c
#include <Pt.h>

int main()
{
    char string[] = "Hello there: äöéû found";
    char mbchar[] = "é";
    int count;
    char *p;

    if( (p = utf8strnchr( string, mbchar, 5, &count ) ) )
        printf( "Character found: character offset %d\n byte offset %d.\n", count, p - string );
    else
        printf( "Not found.\n" );
    return EXIT_SUCCESS;
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

utf8strchr(), utf8strichr(), utf8strnichr(), utf8strrchr(), utf8strirchr()

Unicode Multilingual Support in the Photon Programmer’s Guide
utf8strncmp()

Compare part of a UTF-8 character string

Synopsis:

```c
#include <utf8.h>

int utf8strncmp( char const *str1,
                 char const *str2,
                 int len );
```

Arguments:

- `str1, str2` The UTF-8 strings to compare.
- `len` The number of UTF-8 characters to compare in the strings.

Library:

- ph

Description:

Compare `len` UTF-8 characters from `str1` with `str2`.

The `char_width` parameter must be set to the maximum number of bytes used to represent a single character.

Returns:

An integer less than, equal to, or greater than zero, indicating that `str1` is less than, equal to, or greater than `str2`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
</tbody>
</table>

continued...
utf8strncmp()

**Safety**

| Thread | Yes |

**See also:**

Unicode Multilingual Support in the Photon *Programmer’s Guide*
**utf8strndup()**

Create a copy of part of a UTF-8 character string

**Synopsis:**

```c
char *utf8strndup( char const *text,
                  int count,
                  int *bytes );
```

**Arguments:**

- `text` The UTF-8 character string to be copied.
- `count` The number of character to copy from the string.
- `bytes` A pointer to a location where `utf8strndup()` stores the number of bytes in the copy of the string.

**Library:**

ph

**Description:**

The `utf8strndup()` function creates a copy of the first `count` characters of the given UTF-8 character string, `text`. It sets `bytes` to the number of bytes in the resulting string, not including the terminating `\0`.

**Returns:**

A pointer to the new string.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Chapter 16 • utf8—UTF-8 Character
utf8strndup()

See also:

Unicode Multilingual Support in the Photon *Programmer’s Guide*
utf8strnichr()

Search for a UTF-8 character in part of a string, ignoring case

Synopsis:

```
char *utf8strnichr( char const *string,
                   char const *mbchar,
                   int num,
                   int *count);
```

Arguments:

- `string`: A pointer to the string to search.
- `mbchar`: The character to look for.
- `num`: The maximum number of characters to search in the string.
- `count`: A pointer to the location where `utf8strnichr()` stores the number of UTF-8 characters the matching character is from the start of the search.

Library:

- `ph`

Description:

The `utf8strnichr()` function searches for a character in `string` that matches `mbchar`, regardless of case. If such a match occurs within `num` characters in `string`, `count` (if provided) is set to the number of characters spanned to find the match.

Returns:

A pointer to the beginning of the matching character within `string`, or NULL if no match is found within `num` characters.

Classification:

- Photon
utf8strnichr()  

**Safety**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also:

utf8strchr(), utf8strichr(), utf8strnchr(), utf8strrchr(), utf8strirchr()

Unicode Multilingual Support in the Photon Programmer's Guide
utf8strnlen()

Find the number of bytes used by a UTF-8 character string

Synopsis:

```c
#include <utf8.h>

int utf8strnlen( char const *text,
                 int max_len,
                 int *num );
```

Arguments:

- `text`: A UTF-8 string.
- `max_len`: The maximum number of characters to count.
- `num`: A pointer to a location where `utf8strnlen()` stores the number of characters formed by the number of bytes occupied by `max_len` characters in the string `text`. This will be different from `max_len` if there are fewer than `max_len` UTF-8 characters in `text`.

Library:

ph

Description:

The `utf8strnlen()` function returns the number of bytes occupied by `max_len` characters in the string `text`.

Returns:

The number of bytes occupied by `max_len` characters in the string `text`.

Classification:

Photon
utf8strnlen()  

**Safety**  

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See also:**

`utf8len()`, `utf8strblen()`, `utf8strlen()`  

Unicode Multilingual Support in the Photon *Programmer’s Guide*
utf8strrchr()
Search backwards for a UTF-8 character in a string

Synopsis:

```c
#include <utf8.h>

char *utf8strrchr( char const *string_base, char const *start_char, char const *mbchar, int *count );
```

Arguments:

- `string_base` A pointer to the beginning of the string to search.
- `start_char` A pointer to the character in the string from which to search.
- `mbchar` The character to look for.
- `count` A pointer to the location where `utf8strrchr()` stores the number of UTF-8 characters the matching character is from the start of the search.

Library:

```
ph
```

Description:

This function searches backwards from `start_char` to `string_base` (inclusive) for a UTF-8 character that matches `mbchar`. If such a match is found, `count` (if provided) is set to the number of UTF-8 characters the matching character is from the start of the search. For example, if `mbchar` matches the `start_char` character in `string`, `count` is set to 0. If `mbchar` matches the previous character in `string`, `count` is set to 1.

A pointer to the beginning of the matching character within `string_base` is returned. If no match is found, the function returns NULL and doesn’t change `count`.

Note that `start_char` doesn’t need to be on a character boundary.
utf8strrchr()

Returns:
A pointer to the matched character, or NULL if none was found.

Examples:
Search from the end of a string for a match to the provided UTF-8 character:

```c
#include <Pt.h>

int main()
{
    char string[] = "Hello there: äöü found";
    char mbchar[] = "é";
    int count;
    char *p;

    if( (p = utf8strrchr( string, string + strlen( string ) - 1, mbchar, &count ) ) )
    {
        printf( "Character found: \n%d characters from the end.\n", count );
        printf( "Byte offset %d.\n", p - string );
    }
    else
        printf( "Not found.\n" );
}
```

Classification:
Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>Yes</td>
</tr>
<tr>
<td>Thread</td>
<td>Yes</td>
</tr>
</tbody>
</table>
utf8strrchr()

See also:

utf8strchr(), utf8strichr(), utf8strnchr(), utf8strnichr(), utf8strirchr()

Unicode Multilingual Support in the Photon Programmer’s Guide
utf8towc()

Convert a UTF-8 character to a wide-character code

Synopsis:

```c
int utf8towc( wchar_t *pwc,
              const char *s,
              size_t n );
```

Arguments:

- `pwc` A pointer to a location where `utf8towc()` stores the wide-character representation of `s`.
- `s` A pointer to a UTF-8 character.
- `n` The maximum number of bytes to convert.

Library:

```ph```

Description:

The `utf8towc()` function converts a single UTF-8 character pointed to by `s` into a wide-character code pointed to by `pwc`, to a maximum of `n` bytes.

This function is similar to `mbtowc()`, except:

- `utf8towc()` isn’t affected by the current locale.
- Neither `pwc` nor `s` is allowed to be NULL.
- You can pass 0 for `n` if you know that `s` points to a null-terminated string (i.e. 0 is equivalent to, but more efficient than, `strlen(s)`).
- `utf8towc()` doesn’t set `errno`.

Returns:

- 0 The `s` argument points to the null character.
- > 0 The number of bytes that comprise the multibyte character, to a maximum of UTF8_LEN_MAX (if the next `n` or fewer bytes form a valid multibyte character).
utf8towc()

-1 The next n bytes don’t form a valid (complete) multibyte character.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

* wctouf8()
  Unicode Multilingual Support in the Photon *Programmer’s Guide*

* mbtowc() in the QNX Neutrino *Library Reference*
Chapter 17

wc—Wide-Character
The Photon libraries provide wide-character functions that are useful if you’re working with Unicode characters.
**wctolower()**

Return the lowercase equivalent of a wide character

**Synopsis:**

```c
wchar_t wctolower(wchar_t wc);
```

**Arguments:**

- `wc`: The wide character that you want to get the lowercase equivalent of.

**Library:**

`ph`

**Description:**

The `wctolower()` function returns the lowercase equivalent of `wc`, or `wc` itself if there’s no lowercase equivalent. It’s similar to `tolower()`, except that it knows about Unicode characters.

This function is optimized for size rather than speed. If speed is important, use `wctolower()` to create a full Unicode conversion table and then index into it directly.

**Returns:**

The equivalent lowercase character, or the given character itself if there’s no lowercase equivalent.

**Classification:**

UNIX

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

1908 Chapter 17 • wc—Wide-Character
See also:

tolower() in the QNX Neutrino Library Reference

Unicode Multilingual Support in the Photon Programmer’s Guide
**wctoutf8()**

*Convert a wide-character code into a UTF-8 character*

**Synopsis:**

```c
int wctoutf8( char *s,
              wchar_t wc );
```

**Arguments:**

- `s` A pointer to a location where `wctoutf8()` stores the UTF-8 character corresponding to `wc`.
- `wc` The wide character to be converted.

**Library:**

`ph`

**Description:**

The `wctoutf8()` function stores the multibyte representation corresponding to the code contained in `wc` in the array pointed to by `s`. This function stores at most MB_CUR_MAX characters.

The `wctoutf8()` function is similar to `wctomb()`, except:

- `wctoutf8()` isn’t affected by the current locale.
- The `s` argument isn’t allowed to be `NULL`.

**Returns:**

- `-1` The value of `wc` doesn’t correspond to a valid multibyte character.
- `> 0` The number of bytes that comprise the multibyte character corresponding to the value of `wc`.

**Classification:**

Photon
wctoutf8()

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

utf8towc()

Unicode Multilingual Support in the Photon Programmer’s Guide

wctomb() in the QNX Neutrino Library Reference
accelerator

See hotkey.

activate

A widget is usually activated when you release a mouse button while pointing at an armed widget.

active window

The window that currently has focus.

anchor offset

The distance between the edges of a widget and the parent widget it’s anchored to.

anchor

A constraint mechanism used to manage what happens to a widget when its parent is expanded or contracted. For example, a pane that’s anchored to the sides of a window expands or contracts as the window’s size is changed.

application region

A region that belongs to a Photon application (as opposed to a Photon system process, such as the window manager, graphics drivers, etc.). An application region is usually placed behind the device region. Also called a window region.

argument list

An array of type PtArg_t used when setting and getting widget resources.

arm

A widget is usually armed when you press a mouse button while pointing at it.
backdrop

An image that’s displayed as a background on your screen.

backdrop region

A region placed behind all windows to display a background image.

balloon

A small box that pops up to define or explain part of the user interface. A balloon is displayed when the pointer pauses over a widget.

bitmap

A color picture consisting of one or more bitplanes.

bitplane

An array of bits representing pixels of a single color in a bitmap.

blit

An operation that moves an area of a graphics context (e.g. the screen) to another area on the same or a different context.

callback

A callback function or a callback resource.

callback function

Code connecting an application’s user interface to its code. For example, a callback is invoked when you press a button.

callback resource

A resource that specifies a list of functions and their client data to be called when a certain action occurs.

canvas

The part of a widget that’s used for drawing. For PtWidget, this is the area inside the widget’s borders. For PtBasic and its descendants, the canvas is the area inside the widget’s border and
**margins.** Other widgets, such as `PtLabel`, may define additional margins.

**class**

See `widget class`.

**class hierarchy**

The relationships between all of the widget classes.

**client data**

Any arbitrary data the application may need to provide to a callback function.

**clipping list**

An array of rectangles used to restrict output to a particular area.

**clipping rectangle**

A rectangle used to restrict output to a particular area.

**CMY value**

A color expressed as levels of cyan, magenta, and yellow.

**CMYK value**

A color expressed as levels of cyan, magenta, yellow, and black.

**code-type link callback**

In a PhAB application, an application function that’s called when a widget’s callback list is invoked.

**color depth**

The number of bits per pixel for a screen or pixmap.
Common User Access

See CUA.

compose sequence

A sequence of key presses that can be used to type a character that might not appear on the keyboard.

console

One of nine virtual screens on the desktop. Also called a workspace.

consume

When a widget has processed an event and prevents another widget from interacting with the event, the first widget is said to have consumed the event.

container

A widget that can have other widgets as children. For example, PtWindow, PtGroup, and PtOSContainer.

cooked event

A key or pointer event that has been assigned a location in the Photon event space. Also called a focused event.

CUA

Common User Access — a standard that defines how you can change focus by using the keyboard.

current item

The item in a list or tree widget that will be selected (or perhaps unselected) when you press Enter or Space. It’s typically drawn with a blue dotted line around it when its widget has focus.
**cursor**

An indicator of a position on a screen, such as a **pointer** or an insertion point in a text field.

**damaged**

Whenever a widget needs to be redisplayed due to a change in the window (e.g. the widget is changed, moved, or **realized**), it’s said to be **damaged**.

**dead key**

A key that, when pressed, doesn’t produce a symbol, but initiates a **compose sequence**.

**default placement**

The placement of a region when no siblings are specified. The opposite of **specific placement**.

**desktop**

The virtual screen, consisting of nine **consoles** or **workspaces**.

**device region**

The **region** located in the middle of the **event space**, with **application regions** behind it and **driver regions** in front of it (from the user’s point of view).

**dialog module**

A PhAB **module** similar to a **window module**, except that a dialog module can have only one instance per process.

**direct-color**

A color scheme in which each pixel is represented by an RGB value. Contrast **palette-based**.
**disjoint parent**

A disjoint widget that’s the ancestor of another widget.

**disjoint widget**

A widget that can exist without a parent. If a disjoint widget has a parent, it can exist outside its parent’s canvas. For example, `PtWindow`, `PtMenu`, and `PtRegion` are disjoint widgets, but `PtButton`, `PtBkgd`, and `PtRect` aren’t.

A disjoint widget owns regions that aren’t children of its parent’s regions. Any clipping set by the parent of a disjoint widget isn’t applied to the disjoint widget. The regions of disjoint widgets are sensitive and opaque to expose events.

**dithering**

A process whereby pixels of two colors are combined to create a texture or a blended color.

**draw context**

A structure that defines the flow of the draw stream. The default draw context emits draw events to graphics drivers. **Print contexts** and **memory contexts** are types of draw contexts.

**draw stream**

A series of tokens that are dispatched via draw events and can be collected by a rendering engine such as a graphics driver.

**driver region**

A region created by a driver, usually placed in front of the device region.

**encapsulation driver**

A program that displays Photon graphical output inside another windowing system such as the X Window System.
event

A data structure that represents an interaction between you and an application or between applications. Events travel through the event space either toward you or away (i.e. toward the root region).

event compression

The merging of events such that the application sees only their latest values. The application doesn’t have to process many unnecessary events.

event handler

A callback function that lets an application respond directly to Photon events, such as dragging events.

event mask

A set of event types that are of interest to an event handler. When one of these events occurs, the event handler is invoked.

event space

An abstract, three-dimensional space that contains regions — from the root region at the back to the graphics region at the front. You sit outside the event space, looking in from the front. Events travel through the event space either toward the root region or toward you.

exported subordinate child

A widget created by a container widget (as opposed to an application) whose resources you can access only through the parent.

exposure

Typically occurs when a region is destroyed, resized, or moved. Expose events are sent to applications to inform them when the contents of their regions need to be redisplayed.
extent

A rectangle that describes the outermost edges of a widget.

File Manager

The Photon File Manager (PFM), an application used to maintain and organize files and directories.

focus

A widget that has focus will receive any key events collected by its window.

focus region

A region placed just behind the device region by the Photon Window Manager that lets it intercept key events and direct them to the active window.

focused event

A key or pointer event that has been assigned a location in the Photon event space. Also called a cooked event.

folder

In the Photon File Manager, a metaphor for a directory.

GC

See graphics context.

generate negotiation

The process of determining the layout for a widget and its descendants, which depends on the widget’s layout policy, any size set for the widget, and the dimensions and desired positions of each of the widget’s children.
global header file
A header file that’s included in all code generated by PhAB for an application. The global header file is specified in PhAB’s Application Startup Information dialog.

graphics driver
A program that places a region that’s sensitive to draw events on the user’s side of the device region, collects draw events, and renders the graphical information on the screen.

graphics context (GC)
A data structure that defines the characteristics of primitives, including foreground color, background color, line width, clipping, etc.

Helpviewer
A Photon application for viewing online documentation.

hotkey
A special key or keychord that invokes an action (such as a menu item) without actually selecting a widget. Also called an accelerator. Contrast keyboard shortcut.

hotspot
The part of the pointer that corresponds to the coordinates reported for the pointer (e.g. the intersection of crosshairs, or the tip of the arrow of the basic pointer).

HSB
Hue-Saturation-Brightness color model.

HSV
Hue-Saturation-Value color model.
icon module

A PhAB module that associates icons with an application.

image

A rectangular array of color values, where each element represents a single pixel. See also direct-color and palette-based.

initialization function

In a PhAB application, a function that’s called before any widgets are created.

input driver

A program that emits, and is the source of, key and/or pointer events.

input group

A set of input and output devices. There’s typically one input group per user.

input handler (or input-handling function)

A function that’s hooked into Photon’s main event-processing loop to handle messages and pulses sent to the application by other processes.

instance

A concrete example of an abstract class; for example, “Lassie” is an instance of the class “dog.” In Photon, an instance is usually a widget instance; for example, a pushbutton is an instance of the PtButton widget class. When an instance of a widget is created, the initial values of its resources are assigned.

instance name

In PhAB, a string that identifies a particular instance of a widget so that you can access the instance in your application’s code.
instantiation

The action of creating an **instance** of a widget class in an application.

internal link

A PhAB mechanism that lets a developer access a PhAB module directly from an application’s code.

**Image Viewer**

A Photon application (pv) that displays images.

**key modifier**

A flag in a key event that indicates the state of the corresponding modifier key when another key was pressed.

**keyboard driver**

A program that gets information from the keyboard hardware, builds Photon key events, and emits them towards the root region.

**keyboard shortcut**

A key that selects a menu item. The shortcut works only if the menu is displayed. Contrast **hotkey**.

**language database**

A file that contains the text strings used in a PhAB application; a language database makes it easier to create multilingual applications with PhAB’s language editor.

**link callback**

A mechanism that connects different parts of a PhAB application. For example, a link callback can be invoked to display a dialog when a button is pressed.
**margin**

The area between a widget’s border and canvas.

**memory context**

A draw context in which Photon draw events are directed to memory for future displaying on the screen, as opposed to a printer (print context) or to the screen directly (the default draw context).

**menu module**

A PhAB module used to create a menu.

**method**

A function that’s internal to a widget class and invoked under specific conditions (e.g. to draw the widget). Methods are provided as pointers to functions in widget class records.

**modifier key**

A key (such as Shift, Alt, or Ctrl) used to change the meaning of another key.

**module**

An object in PhAB that holds an application’s widgets. PhAB modules include windows, menus, icons, pictures, and dialogs.

**module-type link callback**

A link callback that displays a PhAB module.

**mouse driver**

A program that gets information from the pointer hardware, builds Photon raw pointer events, and emits them towards the root region.

**opaque**

The state of a region with regard to events. If a region is opaque to an event type, any event of that type that intersects with the region has its
rectangle set adjusted to clip out the intersecting area. The region prevents the event from passing through.

**palette**

An array of colors. A hard palette is in hardware; a soft palette is in software.

**palette-based**

A color scheme in which each pixel is represented by an index into a palette. Contrast direct-color.

**PDR**


**PFM**

See Photon File Manager.

**PhAB**

Photon Application Builder. Visual design tool that generates the code required to implement a user interface.

**phditto**

A utility that accesses the Photon workspace on a remote node. See also ditto.

**Phindows**

Photon in Windows. An application that accesses a Photon session from a Microsoft Windows environment.

**Photon File Manager (PFM)**

An application used to maintain and organize files and directories.
Photon Manager or server

The program that maintains the Photon event space by managing regions and events.

Photon Terminal

An application (pterm) that emulates a character-mode terminal in a Photon window.

Photon Window Manager (PWM)

An application that manages the appearance of window frames and other objects on the screen. For example, the window manager adds the resize bars, title bar, and various buttons to an application’s window. The window manager also provides a method of focusing keyboard events.

picture module

A PhAB module that contains an arrangement of widgets that can be displayed in another widget or used as a widget database.

pixmap

A bitmap or image.

plane mask

A mask used to restrict graphics operations to affect only a subset of color bits.

point source

A single-point rectangle set used as the source of an event.

pointer

An object on the screen that tracks the position of a pointing device (e.g. a mouse, tablet, track-ball, or joystick). Photon has several pointers indicating various states: Basic, Busy, Help, Move, Resize, I-beam, No-input.
Press-drag-release (PDR)

A method of selecting a menu item by pressing down a mouse button while pointing to a menu button, dragging until the desired item is highlighted, and releasing the mouse button.

print context

A draw context in which Photon draw events are directed to a file, as opposed to the screen (the default draw context) or to memory (memory context).

printer driver

A program that converts Photon draw stream format into a format suitable for a printer, including PostScript, Hewlett-Packard PCL, and Canon.

procreated widget

A widget created by another widget (as opposed to an application), such as the PtList and PtText created by a PtComboBox. Also known as a subordinate child.

pterm

A Photon Terminal; an application that emulates a character-mode terminal in a Photon window.

pulse

A small message that doesn’t require a reply; used for asynchronous communication with a Photon application.

pv

See Image Viewer.

PWM

See Photon Window Manager.
**raw event**

An input event that hasn’t been assigned a location in the Photon event space. Also called an **unfocused event**.

**raw callback**

A function that lets an application respond directly to Photon events such as dragging events. Also called an **event handler**.

**realize**

To display a widget and its descendants, possibly making them interactive.

**rectangle set**

An array of nonoverlapping rectangles associated with an event.

**region**

A rectangular area within the Photon event space that’s used by an application for collecting and emitting events.

**resize policy**

A rule that governs how a widget resizes itself when its contents change.

**resource**

An attribute of a widget, such as fill color, dimensions, or a callback list.

**root region**

The region at the very back of the Photon event space.

**sensitive**

The state of a region with regard to events. If a region is sensitive to a particular type of event, the region’s owner collects a copy of any such event that intersects with the region.
setup function

A function that’s called after a PhAB module is created.

shelf

An application that attaches areas to the outside edge of the screen. You can add plugins to customize these areas, such as a taskbar, launcher, clock, and magnifier.

Snapshot

A Photon application for capturing images of the screen.

specific placement

The placement of a region when one or more siblings are specified. The opposite of default placement.

subordinate child

A widget created by another widget (as opposed to an application), such as the PtList and PtText created by a PtComboBox. Also known as a procreated widget.

table-of-contents (TOC) file

In the Photon Helpviewer, a file that describes a hierarchy of help topics.

taskbar

A shelf plugin that displays icons representing the applications that are currently running.

tile

A data structure used to build linked lists of rectangles, such as a list of the damaged parts of an interface.
**topic path**
Help information identified by a string of *titles* that are separated by slashes.

**topic root**
A topic path that’s used as a starting point for locating help topics.

**topic tree**
A hierarchy of help information.

**translation file**
A file containing translated strings for a PhAB application. There’s one translation file per language supported by the application.

**unfocused event**
See **raw event**.

**Unicode**
The ISO/IEC 10646 16-bit encoding scheme for representing the characters used in most languages.

**UTF-8**
The encoding for **Unicode** characters, where each character is represented by one, two, or three bytes.

**widget**
A component (e.g. a pushbutton) in a graphical user interface.

**widget class**
A template for widgets that perform similar functions and provide the same public interface. For example, **PtButton** is a widget class.
widget database

In PhAB, a module containing widgets that can be copied at any time into a window, dialog, or other container.

widget family

A hierarchy of widget instances. For example, a window and the widgets it contains.

widget instance

See instance.

window frame region

A region that PWM adds to a window. It lets you move, resize, iconify, and close the window.

Window Manager

See Photon Window Manager.

window module

A PhAB module that’s instantiated as a PtWindow widget.

window region

A region that belongs to an application window.

work procedure

A function that’s invoked when there are no Photon events pending for an application.

workspace

See console.

workspace menu

A configurable menu that’s displayed when you press or click the right mouse button while pointing at the background of the desktop.
Index

AbGetABW() 90
ABLPATH 118, 169, 202
ABR_CANCEL 165
ABR_CODE 165
ABR_DONE 165
ABR_POST_REALIZE 165
ABR_PRE_REALIZE 165
AL_ACCELERATOR 99
AL_ISMESSAGE 99
AL_MULTILINE 99
AlClearTranslation() 96
AlCloseDBase() 98
alert dialog 1182
AlGetEntry() 99
AlGetSize() 101
AlOpenDBase() 103
alpha attributes
  resetting 490
alpha blending
  map 677
  setting parameters 677, 685
  starting 438
  stopping 437
AlReadTranslation() 105
AlSaveTranslation() 107
AlSetEntry() 109

Index 1935
Index

AlTextEntry_t  99
Ap_MODAL_BLOCK_WINDOWS  171
ApAddClass()  114
ApAddContext()  116
ApAppendTranslation()  118
ApCloseDBase()  121
ApCloseMessageDB()  123
ApCopyDBWidget()  124
ApCreateDBWidget()  127
ApCreateDBWidgetFamily()  130
ApCreateModule()  132
ApCreateWidget()  137
ApCreateWidgetFamily()  140
ApCreateInfo_t  148
ApDeleteDBWidget()  143
ApError()  145
ApGetDBWidgetInfo()  148
ApGetInstance()  154
ApGetItemText()  156
ApGetMessage()  158
ApGetTextRes()  160
ApGetWidgetPtr()  162
ApInfo_t  165
ApInstanceName()  167
ApLoadMessageDB()  169
ApModalWait()  171
ApModifyItemAccel()  173
ApModifyItemState()  175
ApModifyItemText()  177
ApModuleFunction()  179
ApModuleLocation()  182
ApModuleParent()  183
ApName()  185
ApOpenDBase()  187
ApOpenDBaseFile()  189
application callbacks
  adding  1186
  removing  1225
application event handlers
  adding  1188, 1196
  removing  1227
application filter callbacks
  adding  1193
  remove  1231
application hotkey handlers
  removing  1233
application resources
  adding  1244
  getting  1216, 1219
applications
  exiting  1402
    allowing  1184
    preventing  1557
    initializing  1221
    sending to another Photon server  1591
ApRemoveClass()  192
ApRemoveContext()  194
ApResClose()  196
ApSaveDBaseFile()  198
ApSetContext()  200
ApSetTranslation()  202
ApWidget()  205
arcs, drawing  500
area
  data structure  797
  rectangle, converting from  990
  rectangle, converting to  798
  widget canvas, setting based on  1619
  widget, getting  1705
argument lists  1252, 1253, 1621
Index

arrows, drawing
  beveled 511
  to fit in a rectangle 505
attributes
drawing
  plane mask, setting 731
fill
color 701
dithering 703
transparency 708
XOR color 710
stroke
caps 735
color 737
dash 739
dithering 742
joints 744
transparency 747
width 749
XOR color 751
text
color 753
dithering 755
font 712
transparency 757
underline 763
XOR color 759

B
Bézier curve, drawing 516
background processing See threads, work procedures
balloons, inflating 1512
beveled arrow, drawing 511
beveled box, drawing 507
beveled box, drawing with gradients 447, 478, 533
beveled rectangle, drawing 511
bitmaps
color 753
cursor 805, 835
dithering 755
drawing 520
  repeatedly 569
transparency 757
XOR color 759
blitting 449
  within a region 803
  within a widget 1257
  within a widget, with clipping 1286
box, beveled, drawing 507

callbacks
adding
general case 1154, 1157
  Pt_CB_FILTER 1168, 1170
  Pt_CB_HOTKEY 1172
  Pt_CB_RAW 1163, 1166
link, determining the
  initiator 205
PhAB, information
  structure 165
removing
general case 1595, 1597
Index

Pt_CB_FILTER 1605, 1607
Pt_CB_HOTKEY 1609
Pt_CB_RAW 1601, 1603

canvas, determining for a widget 1266
caps, stroke (line) 735

capabilities

Neutrino, attaching 810
Photon
attaching 799, 1514
connection information 906
detaching 847
events, arming for 881
events, checking for 891
reattaching 984
use by widget library 1276

character
cursor 814

capabilities

multibyte  See UTF-8
positions, calculating 323, 332
translating
from UTF-8 1847, 1855
installing 1852
listing supported 1850
tables, building 1840
tables, loading 1842
tables, saving 1844
to UTF-8 1857, 1859
unknown 1862

wide
converting to lowercase 1908
converting to UTF-8 1910
extent, calculating 347

Chinese characters, entering 1408

chroma attributes
resetting 491
chroma key
setting parameters 687
starting 456
stopping 455
class, widget 1714, 1728, 1730
clipboard
copying into 816
header data structure 818
pasting from 819
reading from 821
writing to 823

cleaning
list of rectangles, setting to 726
PtClipAdd() 1284
PtClipRemove() 1288
setting 689, 765
widget, visible extent 1754

colors
alpha component 439
best matches 467
bitmaps 753
blue component 451
CMY, converting to RGB 460
composite, data type 462
fill 701
gray
RGB, converting from 638
RGB, converting to 636
green component 640
HSV
data structure 466
RGB, converting from 668
RGB, converting to 642, 645
model
getting 611
setting 691
palette
current, getting 622
current, setting 728
red component 664
RGB
composite format, converting
to 441, 666
data type 462
gray, converting from 636
gray, converting to 638
HSV, converting from 642, 645
HSV, converting to 668
shading, calculating top and
bottom 443
stroke 737
text 753
text underline 763
XOR
bitmaps 759
fill 710
stroke 751
text 759
configuration files
closing 1762
entries
deleting 1764
getting as string 1772, 1776
opening 1779
parameters, reading and writing
Booleans 1784, 1806
characters 1786, 1809
integers 1788, 1790, 1812, 1815
long integers 1793, 1796, 1821
long long integers 1818
short integers 1799, 1824
strings 1802, 1827
sections
creating 1770
deleting 1766
seeking 1803
seeking first 1768
seeking next 1774
connections
clients
connecting to servers 1320
destroying 1298
error handler 1300
sending to server 1312
server event handlers,
adding 1293
user data, getting 1299
user data, setting 1302
event buffer, resizing 1311
ID 905
information 906
servers
destroying 1314
error handler 1316
message handlers,
adding 1295
notifications, flushing 1305
notifications, sending 1307
replying to clients 1309
temporary name,
creating 1319
user data, getting 1315, 1318
connectors
creating 1323
destroying 1325
finding
  by ID 1303
  by name 1304
getting ID of 1326

consoles
  current, getting
    coordinates 1059
switching 1327
containers
  flux count
    decrementing 1347, 1395
    incrementing 1343, 1652
  flux, determining if in 1522
focus
  finding 1331
  next 1332
  nullifying 1345
  previous 1334
updates
  holding off 1343
  permitting 1347
context
  PhAB
    adding 116, 194
    setting 200
contexts
  data, copying to another
    context 472, 475
draw
  creating 840
  getting current 841
  releasing 842
  setting current 844
  graphical
    alpha, resetting 490
    attributes, resetting 493
    chroma, resetting 491
    creating 483
    current, getting 613
    current, setting 714
    destroying 497
    draw buffer, clearing 457
    draw mode, resetting 494
    fill, resetting 492
    plane mask, resetting 494
    stroke (line) attributes,
      resetting 495
    text attributes, resetting 496
memory
  activating 1097
  creating 1084
  deactivating 1099
  draw buffer, incremental size
    for 1093
  draw buffer, maximum size
    for 1094
  flushing 1089
  releasing 1091
  type, setting 1095
printing
  activating 1104
  attributes 1124
  attributes, modifying 1143
  attributes, querying 1114
  canvas 1112, 1141
  creating 1107
  deactivating 1109, 1149
  initializing 1119, 1121, 1147
  page break 1134
  printers, list of
    available 1111, 1123
    releasing 1139
widgets 1137
control flags, getting 1456
control surfaces
  action ID, getting 1653
  active, getting 1704
  backmost 1667
  bounding box 1658, 1673
  brother behind 1656
  brother in front 1657
  creating 1353, 1359
  damaging 1369, 1371
  data
    adding 1654
    getting 1662
    removing 1675
  destroying 1374–1376
  disabling 1380, 1382, 1669
  enabling 1391, 1393, 1670
  events, matching with 1277
  extent, calculating 1660
  finding 1442, 1444
  finding at a point 1664
  frontmost 1668
  geometry, calculating 1268, 1270, 1272
  hiding 1506, 1508, 1671
  ID, getting 1666
  inserting 1520
  moving to back 1678
  moving to front 1680
  point within, testing for 1677
  showing 1639, 1641, 1672
conventions
  typographical lxxvi
coordinates
  translation
    clearing 459
rectangles 849, 1037
setting 761
CRC (cyclic redundancy check),
  generating 1349, 1351
cursors
  bitmaps 805, 835
  characters 814
  description 837
  information about 976
  moving
    absolute position 961
    relative position 962
custom widgets
  summary of functions 55
cyclic redundancy check (CRC),
  generating 1349, 1351

D
dash, stroke (line) 739
data chains
  adding to 1161
  removing from 1599
  searching 1431, 1440
  unlinking 1698
devices
  currently available 231
  targeting 247
dialogs
  alert 1182
  file selection 1413
  notice 1548
  prompt 1553, 1580
dimensions
  data structure 852
Index

widget, getting 1718

direct mode
  context
    creating 214
    releasing 243
  entering 224
  leaving 226

display power-saving mode,
  setting 693

dithering
  bitmaps 755
  fill 703
  stroke 742
  text 755

drag and drop  See also transport
  data
    canceling 1274
    data_array 1457
    initiating 1517
  PtDndFetch_t 1384
  selecting data 1388

dragging
  cancelling 808
  starting 930

draw buffer
  clearing 457
  flushing 609
  resizing 695

draw contexts
  creating 840
  getting current 841
  releasing 842
  setting current 844

draw events
  region, emitting 624, 733

draw mode
  resetting 494

setting 697

draw streams
  waiting until emitted 785

drawing attributes  See also fill, stroke, and text attributes
  plane mask, setting 731

drawing primitives
  arcs 500
  arrows 505, 511
  Bézier curve 516
  beveled box 507
  beveled rectangle 511
  bitmaps 520, 569
  ellipses 524
  grids 535
  images 538, 550, 553, 574,
        576, 599
  lines 542
  pixels 555, 557
  polygons 559
  rectangles 564
  rectangles, rounded 579
  spans 583
  text 590, 595
  trend 602

driver, graphics 480

E

ellipses, drawing 524

environment variables
  ABLPATH 118, 169, 202
  HOME 1119, 1121
  PATH 1644
  PHFONT 267
PHFONTMEM 267
PHOTON 799, 1591
errno, displaying messages for 145
  events
    arming the Photon channel 881
    asynchronous notification 894
    buffer size, getting 909
    buffer, resizing 1615
    checking for 891
    data
drag (PhDragEvent_t) 853
    getting 908
    key (PhKeyEvent_t) 936
    pointer
      (PhPointerEvent_t) 973
    data structure 863
device
  adding a handler 1191
  removing a handler 1229
emitting 857, 860, 883, 886
forwarding to window
  manager 1451, 1453
key
  ISO8859-1 value 1036
  UTF-8 value 940
main loop 1529
  exiting 1587
modal processing 1540, 1541, 1576
Photon, processing
  outstanding 1255
  processing 1576
rectangle set, getting 915
regions
data structure 898
synchronous notification 889
timer, arming 1034, 1683
types 863
widget, sending to 1617
widgets involved in 1401
extent
  multiline text, calculating 605
  recalculating 1404, 1405
text, calculating 607
visible 1058, 1754
widget, getting 1719

F
family hierarchy
  child of a given class, finding 1423, 1426
  container parent, finding 1430
file descriptors, reducing number of 196
file-descriptor (FD) functions
  adding 1191
  changing modes 1240
  prototype 1406
  removing 1229, 1240
files, selecting 1413
fill attributes
  color 701
  dithering 703
  resetting 492
  transparency 708
  XOR color 710
flags, control, getting 1456
flux count
  decrementing 1347, 1395
Index

incrementing 1343, 1652
focus
  branch 1523
  child widget, closest
    focusable 1435
container
  finding 1331
  next 1332
  nullifying 1345
  previous 1334
  giving to a widget 1336, 1478
  next after focused widget, giving to 1480
  next after given widget, giving to 1484
  next before focused widget, giving to 1486
  next before given widget, giving to 1490
  container’s widget, giving to 1482
  previous container’s widget, giving to 1488
  leaf 1523
  next widget 1436
  previous widget 1437
  widget, determining degree of 1523
FontDetails 399
FontQueryInfo 395
FontRender 404
fonts
  base stem 355
  character positions, calculating 323, 332
  characters, obtaining 389
  extent, calculating 311, 319, 326, 341, 347, 367
  flags, getting 359
  foundry name, getting 357
  fractional scaling
    extent, calculating 367
    glyph outlines, getting 380
  glyphs, obtaining 389
  IDs
    building from stem name 290
    converting to name 283
    generating 351
  information, querying 395
  list of 399
  loading dynamically 299
  metrics
    loading 393
    unloading 424
  name, generating 372
  point size, getting 361
  rendering 404
  resources, freeing 369
  selecting 1448
  server
    attaching 267
    detaching 293
  preloading fonts 392
  setting for subsequent draws 712
  stem names, converting to IDs 290
  unloading dynamically 306
  width, calculating 419, 422, 428, 430
  frames, determining size 1756
front-end processor, controlling from an application 1408

deleting 497
draw buffer, clearing 457
draw mode, resetting 494
fill, resetting 492
plane mask, resetting 494
stroke (line) attributes, resetting 495
text attributes, resetting 496
graphics driver, creating region for 480
graphics hardware capabilities, getting 615
grids, drawing 535

generation
area 797
dimensions 852
point 972
rectangle 986
widgets
area, based on canvas 1619
area, getting 1705
dimensions, getting 1718
extent, recalculating 1404, 1405
extent, visible 1754
offset, getting 1734
positions, absolute 1263
glyphs
bitmap, obtaining 389
outlines, getting 380
gradients
beveled box, drawing 447, 478, 533
colors
light and dark 453
rendering 529
graphical contexts
alpha, resetting 490
attributes, resetting 493
chroma, resetting 491
creating 483
current, getting 613
current, setting 714
hardware capabilities, getting 615
help
quitting 1492
searching 1494
topic path 1496
topic path root 1498
topic tree 1500
URL 1502
URL root 1504
widget with help topic, finding first 1724
HELP_SEARCH_METHOD_EXACT 1493
HELP_SEARCH_METHOD_SUBSTRING 1494
HELP_SEARCH_METHOD_SUBSTRING_CASE 1494
HELP_SEARCH_METHOD_WORD 1493
Index

HELP_SEARCH_MODE_DISPLAYED 1493
HELP_SEARCH_MODE_TEXT 1493
HELP_SEARCH_MODE_TITLE 1493
HELP_SEARCH_SCOPE_ALL 1493
HELP_SEARCH_SCOPE_SELECTED 1493

hierarchy
  family
    child of a given class, finding 1423, 1426
    container parent, finding 1430
    widget class 1714, 1728, 1730

hold count
decrementing 1593, 1701
ignoring 1446
increasing 1510

HOME 1119, 1121

hotkey handlers
  local
    adding 1172
    removing 1609

I

images
  creating 832
cropping 1064
cyclic redundancy check (CRC) 922, 1349, 1351
data structure 922
drawing 550
  part of 553
  primitive functions 538, 599
  repeatedly 574, 576
duplicating 1066
extensions 1829
extracting from databases 151
ghosting 550, 553, 577, 946
inverting 1068
loading 1831, 1846
memory allocation 1833
pixels
  getting 1070, 1074
  setting 1076
reading from screen 659
memory requirements,
  determining 662
releasing members 1018
tag 922
transparency 923
  chroma 953, 1832
  transparency mask 948
types 924

input groups
  determining 933
  regions, information
    about 1024, 1026
input handler
  adding 1197, 1203
  prototype 1519
  removing 1235
internal links 132
interprocess communication (IPC)
  channels
    use by widget library 1276
    channels, attaching 810
    node descriptor, getting from receive ID 1465
    process ID, getting from receive ID 1463, 1465
ionotify() 1198

J
Japanese characters, entering 1408
joints, stroke (line) 744

K
key events
UTF-8 value 940
keys, modifier 854, 936

L
language databases
closing 98
getting entries 99
number of records 101
opening 103
setting translation 109
translations
clearing 96
reading 105
saving 107
layers
arguments
setting 717
capabilities 647
capabilities, querying 619
locking 655, 781

M
main loop 1529
exiting 1587
MAX_FONT_TAG 371
MB_CUR_MAX 1910
mbstr* See utf*
memory contexts
activating 1097
creating 1084
deactivating 1099
draw buffer
incremental size 1093
maximum size 1094
flushing 1089
releasing 1091
Index

type, setting 1095
memory, shared
  attaching 769
cleaning up 771
creating 773
destroying 775
detaching 777
menus
  items
    state, setting 175
text, getting 156
text, setting 173, 177
position, setting 1555
message databases
  closing 98, 123
getting entries 99
loading 169
number of records 101
opening 103
retrieving messages 158
setting translation 109
translations
  clearing 96
  reading 105
  saving 107
messages
  displaying 1531
errors, displaying 145
modal dialogs 171, 1530, 1576
modal loop
  starting 1533
  stopping 1543
modal processing
  ending 1540
  starting 1541
modifier keys 854, 936
modules
  creating 132
location, specifying 182
parent, specifying 183
setup function, specifying 179
setup functions
  information structure 165
widget instance pointer,
  getting 154
widget instance pointer, getting
  within 162
widgets, determining which
  initiated a link callback
  205
MsgReceive() 894
multibyte characters  See UTF-8

N

name_attach()  1199, 1204
nodes
  descriptor, getting from receive
    ID (RCVID)  1465
notice dialog 1548

O

offscreen contexts
  layers
    creating for 486
displaying 723
translation, setting 245
P

palette
  current, getting 622
  current, setting 728
passwords, prompting for 1553
PATH 1644
pathname delimiter in QNX
Momentics documentation lxxviii

PdCreateDirectContext() 214
PdCreateOffscreenContext() 218
PdCreateOffscreenLock() 220
PdDestroyOffscreenLock() 222
PdDirectStart() 224
PdDirectStop() 226
PdDupOffscreenContext() 229
PdGetDevices() 231
PdGetOffscreenContextPtr() 233
PdGetOffscreenSurface() 237
PdIsOffscreenLocked() 238
PdLockOffscreen() 240
PdOffscreenContext_t 242
PdOSCCreateLockParams_t 220
PdReleaseDirectContext() 243
PdSetOffscreenTranslation() 245
PdSetTargetDevice() 247
PdUnlockOffscreen() 250
PF_BITMAP 359
PF_CHAR_DRAW_POSITIONS 322, 331
PF_FRACTIONAL 310, 311, 330, 404
PF_RECT 311
PF_SCALABLE 359

PF_STYLE_ANTIALIAS 350, 359, 371
PF_STYLE_BI 359
PF_STYLE_BOLD 350, 359, 371
PF_STYLE_DULINE 350, 371
PF_STYLE_ITALIC 350, 359, 371
PF_STYLE_ULINE 350, 371
PF_WIDE_CHAR32 310, 403
PF_WIDE_CHARS 310, 322, 331, 403
PfAttach() 267
PfAttachCx() 267
PfConvertFontID() 283
PfConvertFontIDCx() 283
PfDecomposeStemToID() 290
PfDecomposeStemToIDCx() 290
PfDetach() 293
PfDetachCx() 293
PfDynamicLoad() 299
PfDynamicLoadCx() 299
PfDynamicUnload() 306
PfDynamicUnloadCx() 306
PfExtent() 311
PfExtentComponents() 319
PfExtentComponentsCx() 319
PfExtentCx() 311
PfExtentFractTextCharPositions() 323
PfExtentText() 326
PfExtentTextCharPositions() 332
PfExtentTextCharPositionsCx() 332
PfExtentToRect() 341
PfExtentWideText() 347
PfFindFont() 351
PfFindFontCx() 351
PfFontBaseStem() 355
PfFontBaseStemCx() 355

Index 1949
Index

PfFontDescription() 357
PfFontDescriptionCx() 357
PfFontFlags() 359
PfFontFlagsCx() 359
PfFontSize() 361
PfFontSizeCx() 361
PfFractionalExtentText() 367
PfFreeFont() 369
PfFreeFontCx() 369
PfGenerateFontName() 372
PfGenerateFontNameCx() 372
PfGetOutline() 380
PfGetOutlineCx() 380
PfGlyph() 389
PfGlyphCx() 389
PfLoadFont() 392
PfLoadMetrics() 393
PfLoadMetricsCx() 393
PfQueryFontInfo() 395
PfQueryFontInfoCx() 395
PfQueryFonts() 399
PfQueryFontsCx() 399
PfRender() 404
PfRenderCx() 404
PfTextWidthBytes() 419
PfTextWidthChars() 422
PfUnloadMetrics() 424
PfWideTextWidthBytes() 428
PfWideTextWidthChars() 430
Pg_2D_ACCELERATOR 617
Pg_ALPHA_OP_DST_GLOBAL 678
Pg_ALPHA_OP_SRC_GLOBAL 678
Pg_ALPHA_OP_SRC_MAP 678
Pg_ALPHA_OP_TEST 678
Pg_ARC 500
Pg_ARC_CHORD 500
Pg_ARC_PIE 500
Pg_BACK_FILL 520, 545, 569, 590, 591, 595
Pg_BEVEL_ADOWN 511
Pg_BEVEL_ALEFT 512
Pg_BEVEL_ARIGHT 512
Pg_BEVEL_AUP 511
Pg_BEVEL_CLIP 511
Pg_BEVEL_JOIN 744
Pg_BEVEL_MAX 507
Pg_BEVEL_ROUND 511
Pg_BEVEL_SET 512
Pg_BEVEL_SQUARE 511
Pg_BITMAP_BACKFILL 554, 924, 927, 1071
Pg_BITMAP_TRANSPARENT 554, 924, 927, 1071
Pg_BLACK 463
Pg_BLEND_DST_0 680
Pg_BLEND_DST_1 680
Pg_BLEND_DST_1mA0Ad 682
Pg_BLEND_DST_1mA0As 682
Pg_BLEND_DST_1mA1Ad 682
Pg_BLEND_DST_1mA1As 681
Pg_BLEND_DST_1mAd 681
Pg_BLEND_DST_1mA 681
Pg_BLEND_DST_1mS 681
Pg_BLEND_DST_A0Ad 682
Pg_BLEND_DST_A0As 682
Pg_BLEND_DST_A1Ad 681
Pg_BLEND_DST_A1As 681
Pg_BLEND_DST_Ad 681
Pg_BLEND_DST_As 681
Pg_BLEND_DST_ONE_MINUS_SRC_ALPHA 680
Pg_BLEND_DST_S 681
Pg_BLEND_SRC_0  679
Pg_BLEND_SRC_1  679
Pg_BLEND_SRC_1mA0Ad  680
Pg_BLEND_SRC_1mA0As  680
Pg_BLEND_SRC_1mA1Ad  680
Pg_BLEND_SRC_1mA1As  679
Pg_BLEND_SRC_1mAd  679
Pg_BLEND_SRC_1mD  679
Pg_BLEND_SRC_A0Ad  680
Pg_BLEND_SRC_A0As  680
Pg_BLEND_SRC_A1Ad  680
Pg_BLEND_SRC_A1As  679
Pg_BLEND_SRC_Ad  679
Pg_BLEND_SRC_As  679
Pg_BLEND_SRC_D  679
Pg_BLEND_SRC_SRC_ALPHA  679, 680
Pg_BLUE  463
Pg_BROWN  463
Pg_BUTT_CAP  735
Pg_BUTT_JOIN  744
Pg_BVB_DRAW_ALL  447, 478, 534
Pg_BVB_DRAW_ALL_BV  534
Pg_BVB_DRAW_ALL_INLINE  533
Pg_BVB_DRAW_ALL_OUTLINE  533
Pg_BVB_DRAW_BITS  447, 478, 534
Pg_BVB_DRAW_BOTTOM  447, 478, 534
Pg_BVB_DRAW_BOTTOM_INLINE  533
Pg_BVB_DRAW_BOTTOM_OUTLINE  533
Pg_BVB_DRAW_LEFT  447, 478, 534
Pg_BVB_DRAW_LEFT_INLINE  533
Pg_BVB_DRAW_LEFT_OUTLINE  533
Pg_BVB_DRAW_RIGHT  447, 478, 534
Pg_BVB_DRAW_RIGHT_INLINE  533
Pg_BVB_DRAW_RIGHT_OUTLINE  533
Pg_BVB_DRAW_TOP  447, 478, 534
Pg_BVB_DRAW_TOP_INLINE  533
Pg_BVB_DRAW_TOP_OUTLINE  533
Pg_BVB_FILL  446, 477, 533
Pg_BVB_FULL_GRADIENTS  447, 478, 533
PgCELERON  463
Pg_CHROMA_DEST_MATCH  687
Pg_CHROMA_DRAW  687
Pg_CHROMA_NODRAW  687
Pg_CHROMA_SRC_MATCH  687
Pg_CLOSED  516, 560
Pg_CM_ARGB  439, 441, 462
Pg_CM_DOUBLE_BUFFERED  616
Pg_CM_PRGB  462
Pg_CM_RGB  462
Pg_CYAN  463
Pg_DBLUE  463
Pg_DCYAN  463
Pg_DEVICE_COLOR  464
Pg_DGRAY  463
Index

Pg_DGREEN 463
Pg_DRAW_FILL 500, 507, 512, 516, 523, 560, 564, 579, 584
Pg_DRAW_FILL_STROKE 500, 507, 512, 516, 523, 560, 564, 579, 584
Pg_DRAW_STROKE 500, 507, 512, 516, 523, 560, 564, 579, 584
Pg_DRAWMODE_AND 697
Pg_DRAWMODE_OPAQUE 472, 475, 479, 697
Pg_DRAWMODE_OR 697
Pg_DRAWMODE_XOR 464, 697
Pg_DrawModePSN 472, 475, 479
Pg_DrawModeS1 697
Pg_EXTENT_BASED 523
Pg_GHOST 550, 553, 577, 946
Pg_GRAD_4POINT 528
Pg_GRAD_BOX_4POINT 528
Pg_GRAD_BOX_DIAGB 528
Pg_GRAD_BOX_DIAGF 528
Pg_GRAD_DIAGB 528
Pg_GRAD_DIAGF 528
Pg_GRAD_EXP 529
Pg_GRAD_HILL 528
Pg_GRAD_HILL2 529
Pg_GRAD_HORZ 528
Pg_GRAD_LINEAR 528
Pg_GRAD_TABLE 529
Pg_GRAD_VERTICAL 528
Pg_GRAY 463
Pg_GREEN 463
Pg_IMAGE_CLASS_DIRECT 927
Pg_IMAGE_CLASS_GRADIENT 927
Pg_IMAGE_CLASS_MASK 927
Pg_IMAGE_CLASS_PALETTE 927
Pg_IMAGE_DIRECT_1555 925, 949, 1070
Pg_IMAGE_DIRECT_444 925, 949, 1070
Pg_IMAGE_DIRECT_4444 925, 949, 1070
Pg_IMAGE_DIRECT_555 925, 949, 1070, 1085
Pg_IMAGE_DIRECT_565 925, 949, 1070, 1085
Pg_IMAGE_DIRECT_888 926, 949, 1070, 1085
Pg_IMAGE_DIRECT_8888 925, 949, 1070, 1085
Pg_IMAGE_GRADIENT_BYTE 926, 948, 1070
Pg_IMAGE_GRADIENT_NIBBLE 554, 926, 948, 1070
Pg_IMAGE_PALETTE_BYTE 926, 948, 1070, 1085
Pg_IMAGE_PALETTE_NIBBLE 554, 926, 948, 1070
Pg_INDEX_COLOR 464, 729, 948, 953
Pg_INVERT_COLOR 464, 698
Pg_LAYER_ARG_ACTIVE 717
Pg_LAYER_ARG_ALPHA 719
Pg_LAYER_ARG_BRIGHTNESS 718
Pg_LAYER_ARG_CHROMA 719
Pg_LAYER_ARG_CONTRAST 719
Pg_LAYER_ARG_DST_VIEWPORT 718
Pg_LAYER_ARG_EDGE_MODE 720
Index

Pg_LAYER_ARG_FILTER_MODE 720
Pg_LAYER_ARG_FORMAT_INDEX 718
Pg_LAYER_ARG_LIST_BEGIN 721
Pg_LAYER_ARG_LIST_END 721
Pg_LAYER_ARG_SATURATION 719
Pg_LAYER_ARG_SRC_VIEWPORT 718
Pg_LAYER_CAP_* 648
Pg_LAYER_FORMAT_* 647
Pg_LINEAR_FRAME_BUFFER_CAPABLE 618
Pg_MAGENTA 463
Pg_MGRAY 463
Pg_MITER_JOIN 744
Pg_OFFSCREEN 617
Pg_OSC_ALREADY_UNLOCKED 250
Pg_OSC_CREATE_LOCK_FAILED 221
Pg_OSC_CRTC_SAFE 216, 228, 779
Pg_OSC_LOCK_ALREADY_CREATED 221
Pg_OSC_LOCK_DEADLOCK 241
Pg_OSC_LOCK_INVALID 222, 238, 241, 250
Pg_OSC_LOCK_SIG 220
Pg_OSC_LOCK_TIMED_OUT 240, 241
Pg_OSC_LOCKED 238
Pg_OSC_MAIN_DISPLAY 217
Pg_OSC_MEM_2D_READABLE 217, 229, 485
Pg_OSC_MEM_2D_WRITABLE 217, 228, 485
Pg_OSC_MEM_HINT_CPU_READ 217, 229, 485
Pg_OSC_MEM_HINT_CPU_WRITE 218, 229, 486
Pg_OSC_MEM_PAGE_ALIGN 217, 228, 233, 485
Pg_OSC_MEM_SYS_ONLY 217, 228
Pg_OSC_NOT_LOCKED 238, 250
Pg_PALSET_FORCE_EXPOSE 730
Pg_PALSET_GLOBAL 729
Pg_PALSET_HARD 728
Pg_PALSET_HARDACTIVE 729
Pg_PALSET_HARDLOCKED 729
Pg_PALSET_SOFT 729
Pg_PAT_DEFAULT 708, 747, 757
Pg_POINT_CAP 735
Pg_PURPLE 463
Pg_QROUND_JOIN 744
Pg_RED 463
Pg_RELATIVE 516, 560, 761
Pg_REPBM_ALTERNATE 569, 574, 577
Pg_ROUND_CAP 735
Pg_ROUND_JOIN 744
Pg_SCALER_CAP_BRIGHTNESS_ADJUST 671
Pg_SCALER_CAP_CONTRAST_ADJUST 671
Pg_SCALER_CAP_DOUBLE_BUFFER 671
Pg_SCALER_CAP_DST_CHROMA_KEY 671
Pg_SCALER_PROP_CHROMA_ENABLE 673

Index 1953
<table>
<thead>
<tr>
<th>Function/Fetch/Method</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pg_SCALER_PROP_CHROMA_SPECIFY_KEY_MASK</td>
<td>673</td>
</tr>
<tr>
<td>Pg_SCALER_PROP_DISABLE_FILTERING</td>
<td>674</td>
</tr>
<tr>
<td>Pg_SCALER_PROP_DOUBLE_BUFFER</td>
<td>674</td>
</tr>
<tr>
<td>Pg_SCALER_PROP_DRAW_TARGETABLE</td>
<td>674</td>
</tr>
<tr>
<td>Pg_SCALER_PROP_SCALER_ENABLE</td>
<td>674</td>
</tr>
<tr>
<td>Pg_SCALER_PROP_TO_BACK</td>
<td>674</td>
</tr>
<tr>
<td>Pg_SCALER_PROP_TO_FRONT</td>
<td>674</td>
</tr>
<tr>
<td>Pg_SQUARE_CAP</td>
<td>735</td>
</tr>
<tr>
<td>Pg_SWAP_BLIT</td>
<td>779</td>
</tr>
<tr>
<td>Pg_TEXT_BOTTOM</td>
<td>545, 591, 596</td>
</tr>
<tr>
<td>Pg_TEXT.CENTER</td>
<td>545, 591, 596</td>
</tr>
<tr>
<td>Pg_TEXT_ELLIPSS</td>
<td>596</td>
</tr>
<tr>
<td>Pg_TEXT_ELLIPSS_INVERT</td>
<td>596</td>
</tr>
<tr>
<td>Pg_TEXT_ELLIPSS_MIDDLE</td>
<td>596</td>
</tr>
<tr>
<td>Pg_TEXT_LEFT</td>
<td>545, 591, 596</td>
</tr>
<tr>
<td>Pg_TEXT_MIDDLE</td>
<td>545, 591, 596</td>
</tr>
<tr>
<td>Pg_TEXT_RIGHT</td>
<td>545, 591, 596</td>
</tr>
<tr>
<td>Pg_TEXT_TOP</td>
<td>545, 591, 596</td>
</tr>
<tr>
<td>Pg_TEXT_WIDECHAR</td>
<td>590, 591, 595</td>
</tr>
<tr>
<td>Pg_TRANSPARENT</td>
<td>464, 763</td>
</tr>
<tr>
<td>Pg_TREND_HORIZ</td>
<td>602</td>
</tr>
<tr>
<td>Pg_TREND_VERT</td>
<td>602</td>
</tr>
<tr>
<td>Pg_VGA0 - Pg_VGAF</td>
<td>463</td>
</tr>
<tr>
<td>Pg_VIDEO_CHANNEL_SCALER</td>
<td>488</td>
</tr>
<tr>
<td>Pg_VIDEO_FORMAT_*</td>
<td>670</td>
</tr>
<tr>
<td>Pg_VIDEO_OVERLAY</td>
<td>617</td>
</tr>
<tr>
<td>Pg_WHITE</td>
<td>463</td>
</tr>
<tr>
<td>Pg_YELLOW</td>
<td>463</td>
</tr>
<tr>
<td>PgAlphaOff(), PgAlphaOffCx()</td>
<td>437</td>
</tr>
<tr>
<td>PgAlphaOn(), PgAlphaOnCx()</td>
<td>438</td>
</tr>
<tr>
<td>PgAlphaValue()</td>
<td>439</td>
</tr>
<tr>
<td>PgARGB()</td>
<td>441</td>
</tr>
<tr>
<td>PgBackgroundShadings()</td>
<td>443</td>
</tr>
<tr>
<td>PgBevelBox(), PgBevelBoxCx()</td>
<td>447</td>
</tr>
<tr>
<td>PgBlit(), PgBlitCx()</td>
<td>449</td>
</tr>
<tr>
<td>PgBlueValue()</td>
<td>451</td>
</tr>
<tr>
<td>PgCalcColorContrast()</td>
<td>453</td>
</tr>
<tr>
<td>PgChroma_t</td>
<td>719</td>
</tr>
<tr>
<td>PgChromaOff(), PgChromaOffCx()</td>
<td>455</td>
</tr>
<tr>
<td>PgChromaOn(), PgChromaOnCx()</td>
<td>456</td>
</tr>
<tr>
<td>PgClearDrawBuffer(), PgClearDrawBufferCx()</td>
<td>457</td>
</tr>
<tr>
<td>PgClearTranslation(), PgClearTranslationCx()</td>
<td>459</td>
</tr>
<tr>
<td>PgCMY()</td>
<td>460</td>
</tr>
<tr>
<td>PgColor_t</td>
<td>462</td>
</tr>
<tr>
<td>PgColorHSV_t</td>
<td>466</td>
</tr>
<tr>
<td>PgColorMatch()</td>
<td>467</td>
</tr>
<tr>
<td>PgConfigScalerChannel()</td>
<td>469</td>
</tr>
<tr>
<td>PgContextBlit(), PgContextBlitCx()</td>
<td>472</td>
</tr>
<tr>
<td>PgContextBlitArea(), PgContextBlitAreaCx()</td>
<td>475</td>
</tr>
<tr>
<td>PgContrastBevelBox(), PgContrastBevelBoxCx()</td>
<td>478</td>
</tr>
<tr>
<td>PgCreateDriverRegion()</td>
<td>480</td>
</tr>
<tr>
<td>PgCreateGC()</td>
<td>483</td>
</tr>
<tr>
<td>PgCreateLayerSurface()</td>
<td>486</td>
</tr>
</tbody>
</table>
Index

PgCreateVideoChannel()  488
PgDefaultAlpha()  490
PgDefaultFill()  491, 492
PgDefaultGC()  493
PgDefaultMode()  494
PgDefaultStroke()  495
PgDefaultText()  496
PgDestroyGC()  497
PgDestroyVideoChannel()  498
PgDisplaySettings_t  628, 767
PgDrawArc()  500
PgDrawArcCx()  500
PgDrawArrow()  505
PgDrawArrowCx()  505
PgDrawBevelBox()  507
PgDrawBevelBoxCx()  507
PgDrawBeveled()  511
PgDrawBeveledCx()  511
PgDrawBezier(), PgDrawBezierv()  516
PgDrawBezierCx(), PgDrawBezierCxv()  516
PgDrawBitmap(), PgDrawBitmapv()  520
PgDrawBitmapCx(), PgDrawBitmapCxv()  520
PgDrawEllipse()  524
PgDrawEllipseCx()  524
PgDrawGradient(), PgDrawGradientCx()  529
PgDrawGradientBevelBox(), PgDrawGradientBevelBoxCx()  533
PgDrawGrid(), PgDrawGridCx()  535
PgDrawIBevelBox()  507
PgDrawIBevelBoxCx()  507
PgDrawILine(), PgDrawILineCx()  542
PgDrawImage(), PgDrawImagev()  538
PgDrawImageCx(), PgDrawImageCxv()  538
PgDrawIPixel(), PgDrawIPixelCx()  555
PgDrawIRect(), PgDrawIRectCx()  564
PgDrawLine(), PgDrawLineCx()  542
PgDrawMultiTextArea(), PgDrawMultiTextAreaCx()  546
PgDrawPhImage(), PgDrawPhImagev()  550
PgDrawPhImageCx(), PgDrawPhImageCxv()  550
PgDrawPhImageRectv(), PgDrawPhImageRectCx()  553
PgDrawPixel(), PgDrawPixelCx()  555
PgDrawPixelArray(), PgDrawPixelArrayv()  557
PgDrawPixelArrayCx(), PgDrawPixelArrayCxv()  557
PgDrawPolygon(), PgDrawPolygonv()  559
Index

PgDrawPolygonCx(),
    PgDrawPolygonCxv() 559
PgDrawRect(), PgDrawRectCx() 564
PgDrawRepBitmap(),
    PgDrawRepBitmapv() 569
PgDrawRepBitmapCx(),
    PgDrawRepBitmapCxv() 569
PgDrawRepImage(),
    PgDrawRepImagev() 574
PgDrawRepImageCx(),
    PgDrawRepImageCxv() 574
PgDrawRepPhImage(),
    PgDrawRepPhImagev() 576
PgDrawRepPhImageCx(),
    PgDrawRepPhImageCxv() 576
PgDrawRoundRect(),
    PgDrawRoundRectCx() 579
PgDrawSpan(), PgDrawSpanv() 583
PgDrawSpanCx(),
    PgDrawSpanCxv() 583
PgDrawString(), PgDrawStringv() 587
PgDrawStringCx(),
    PgDrawStringCxv() 587
PgDrawText(), PgDrawTextCx() 590
PgDrawTextArea(),
    PgDrawTextAreaCx() 595
PgDrawTextChars(),
    PgDrawTextCharsCx() 590
PgDrawTextv(), PgDrawTextCxv() 590
PgDrawTImage(),
    PgDrawTImagev() 599
PgDrawTImageCx(),
    PgDrawTImageCxv() 599
PgDrawTrend(), PgDrawTrendv() 602
PgDrawTrendCx(),
    PgDrawTrendCxv() 602
PgExtentMultiText() 605
PgExtentText() 607
PgFlush(), PgFFlush() 609
PgFlushCx(), PgFFlushCx() 609
PgGetColorModel(),
    PgGetColorModelCx() 611
PgGetGC(), PgGetGCCx() 613
PgGetGraphicsHWCaps() 615
PgGetLayerCaps() 619
PgGetOverlayChromaColor() 621
PgGetPalette() 622
PgGetRegion(), PgGetRegionCx() 624
PgGetScalerCapabilities() 626
PgGetVideoMode() 628
PgGetVideoModeInfo() 630
PgGetVideoModeList() 634
PgGray() 636
PgGrayValue() 638
PgGreenValue() 640
Index

PgHSV() 642
PgHSV2RGB() 645
PgHWCaps_t 615
PgLayerAlpha_t 719
PgLayerCaps_t 647
PgLockLayer() 655
PgMap_t 677
PgNextVideoFrame() 658
PgReadScreen() 659
PgReadScreenSize() 662
PgRedValue() 664
PgRGB() 666
PgRGB2HSV() 668
PgScalerCaps_t 670
PgScalerProps_t 673
PgSetAlpha(), PgSetAlpha() 677
PgSetAlphaBlend(),
  PgSetAlphaBlendCx() 685
PgSetChroma(), PgSetChromaCx() 687
PgSetClipping(),
  PgSetClippingCx() 689
PgSetColorModel(),
  PgSetColorModelCx() 691
PgSetDPMSMode() 693
PgSetDrawBufferSize(),
  PgSetDrawBufferSizeCx() 695
PgSetDrawMode(),
  PgSetDrawModeCx() 697
PgSetFillColor(),
  PgSetFillColorCx() 701
PgSetFillDither(),
  PgSetFillDitherCx() 703
PgSetFillTransPat(),
  PgSetFillTransPatCx() 708
PgSetFillXORColor(),
  PgSetFillXORColorCx() 710
PgSetFont(), PgSetFontCx() 712
PgSetGC(), PgSetGCCx() 714
PgSetLayerArg() 717
PgSetLayerSurface() 723
PgSetMultiClip(),
  PgSetMultiClipCx() 726
PgSetPalette(), PgSetPaletteCx() 728
PgSetPlaneMask(),
  PgSetPlaneMaskCx() 731
PgSetRegion(), PgSetRegionCx() 733
PgSetStrokeCap(),
  PgSetStrokeCapCx() 735
PgSetStrokeColor(),
  PgSetStrokeColorCx() 737
PgSetStrokeDash(),
  PgSetStrokeDashCx() 739
PgSetStrokeDither(),
  PgSetStrokeDitherCx() 742
PgSetStrokeFWidth(),
  PgSetStrokeFWidthCx() 749
PgSetStrokeJoin(),
  PgSetStrokeJoinCx() 744
PgSetStrokeTransPat(),
  PgSetStrokeTransPatCx() 747

Index  1957
Index

`PgSetStrokeWidth()`, `PgSetStrokeWidthCx()` 749
`PgSetStrokeXORColor()`, `PgSetStrokeXORColorCx()` 751
`PgSetTextColor()`, `PgSetTextColorCx()` 753
`PgSetTextDither()`, `PgSetTextDitherCx()` 755
`PgSetTextTransPat()`, `PgSetTextTransPatCx()` 757
`PgSetTextXORColor()`, `PgSetTextXORColorCx()` 759
`PgSetTranslation()`, `PgSetTranslationCx()` 761
`PgSetUnderline()`, `PgSetUnderlineCx()` 763
`PgSetUserClip()`, `PgSetUserClipCx()` 765
`PgSetUserClipAbsolute()`, `PgSetUserClipAbsoluteCx()` 765
`PgSetVideoMode()` 767
`PgShmemAttach()` 769
`PgShmemCleanup()` 771
`PgShmemCreate()` 773
`PgShmemDestroy()` 775
`PgShmemDetach()` 777
`PgSpan_t` 583
`PgSwapDisplay()`, `PgSwapDisplayCx()` 779
`PgUnlockLayer()` 781
`PgVideoChannel_t` 783

`PgVideoModeInfo_t` 630
`PgVideoModes_t` 634
`PgVM_MODE_CAP1_*` 631
`PgVM_MODE_CAP2_*` 632
`PgWaitDrawComplete()` 785
`PgWaitHWIdle()` 786
`PgWaitVSync()`, `PgWaitVSyncCx()` 787
`Ph_ACTIVATE_DC` 839
`Ph_AUXPTR_REGION` 980
`Ph_BACK_EVENT` 864
`Ph_BUTTON_ADJUST` 855, 938, 974, 977
`Ph_BUTTON_MENU` 855, 938, 974, 977
`Ph_BUTTON_SELECT` 855, 938, 974, 977
`Ph_CAPTURE_EXPOSE` 609, 872
`Ph_CLIPBOARD_TYPE_TEXT` 818
`Ph_CONSUMED` 864
`Ph_CREATED_DC` 839
`Ph_CURSOR_INHERIT` 992
`Ph_CURSOR_NO_INHERIT` 992
`Ph_CURSOR_NOINPUT` 1259, 1261
`Ph_DEACTIVATE_DC` 839
`Ph_DESTROYING_DC` 839
`Ph_DEV_RID` 898
`Ph_DIRECTED_FOCUS` 864
`Ph_DONE_DRAW` 609
`Ph_DONT_COPY` 1049, 1178, 1687, 1695
`Ph_DRAG_KEY_MOTION` 855, 929
`Ph_DRAG_NOBUTTON` 855, 929
`Ph_DRAG_TRACK` 855, 928
Ph_DRAW_TO_MEMORY_CONTEXT 838
Ph_DRAW_TO_OFFSCREEN_MEMORY 838
Ph_DRAW_TO_PHOTON 838
Ph_DRAW_TO_PRINT_CONTEXT 838
Ph_DRAW_TO_SERVICE 838
Ph_DYNAMIC_BUFFER 800, 889, 891, 909
Ph_EMIT_TOWARD 865
Ph_EV_BOUNDARY 866, 980
Ph_EV_BUTPRESS 868, 973, 977, 980, 1704
Ph_EV_BUT_RELEASE 868, 973, 980
Ph_EV_BUT_REPEAT 868, 973, 980
Ph_EV_DND_ACK 870
Ph_EV_DND_CANCEL 869
Ph_EV_DND_COMPLETE 869
Ph_EV_DND_DELIVERED 869
Ph_EV_DND_DROP 870
Ph_EV_DND_ENTER 869
Ph_EV_DND_INIT 869
Ph_EV_DND_LEAVE 870
Ph_EV_DND_MOTION 869
Ph_EV_DND_NAK 870
Ph_EV_DND_REPEAT 870
Ph_EV_DNDROP 869, 980
Ph_EV_DRA G 808, 870, 929, 930, 980
Ph_EV_DRA G_BOUNDARY 870
Ph_EV_DRA G_COMPLETE 870
Ph_EV_DRA G_INIT 871
Ph_EV_DRA G_KEY_EVENT 855, 871
Ph_EV_DRA G_MOTION_EVENT 871
Ph_EV_DRA G_MOVE 871
Ph_EV_DRA G_START 871
Ph_EV_DRAW 871, 980
Ph_EV_EXPOSE 872, 980, 997
Ph_EV_FEP 873, 877
Ph_EV_INFO 873, 980
Ph_EV_INVALIDATE_SYSINFO 873
Ph_EV_KEY 874, 929, 980
Ph_EV_PTR_MOTION_BUTTON 875, 929, 973, 980
Ph_EV_PTR_MOTION_NOBUTTON 875, 929, 973, 980
Ph_EV_PTR_STEADY 867
Ph_EV_PTR_UNSTEADY 867
Ph_EV_RAW 875, 980
Ph_EV_RELEASE_ENDCLICK 868
Ph_EV_RELEASE_OUTBOUND 868
Ph_EV_RELEASE_PHANTOM 868
Ph_EV_RELEASE_REAL 868
Ph_EV_REMOTE_WM 876
Ph_EV_SERVICE 875, 980
Ph_EV_SYSTEM 878, 980
Ph_EV_TIMER 879, 980, 1683
Ph_EV_WM 880, 981, 1054
Ph_EV_WM_EVENT 880, 1054
Ph_EVENT_ABSOLUTE 865, 994
Ph_EVENT_DIRECT 865
Ph_EVENT_INCLUSIVE 865
Ph_EVENT_MSG 889, 891
Ph_EXPOSE_FAMILY 997
Ph_EXPOSE_REGION 997
Ph_FAKE_EVENT 864
Index

Ph_FEP_ACTIVATE 878
Ph_FEP_broadcast 873, 877
Ph_FEP_change_mode 1407
Ph_FEP_Chinese 874
Ph_FEP_deactivate 878
Ph_FEP_DEREGISTER 873
Ph_FEP_HELP 1407
Ph_FEP_JAPANESE 874
Ph_FEP_KOREAN 874
Ph_FEP_norect 878
Ph_FEP_rect 878
Ph_FEP_REGISTER 873, 877
Ph_FEP_TOGGLE_1 1407
Ph_FOCUS_BRANCH 864
Ph_FORCE_BOUNDARY 980, 993
Ph_FORCE_FRONT 980, 993
Ph_GEN_INFO_BANDWIDTH 1025
Ph_GEN_INFO_CAPABILITIES 1025
Ph_GEN_INFO_NUM_GFX 1025
Ph_GEN_INFO_NUM_IG 1025
Ph_GEN_INFO_NUM_KBD 1025
Ph_GEN_INFO_NUM_PTR 1025
Ph_GRAFX_REGION 980, 993
Ph_GRAPHIC_EXPOSE 873
Ph_INLINE_SHMEM_OBJECTS 838
Ph_INPUTGROUP_REGION 980, 993
Ph_KBD_REGION 980, 993
Ph_LIB_VERSION 941
Ph_NO_HOLD 800
Ph_NORMAL_DRAW 609
Ph_NORMAL_EXPOSE 872
Ph_NOT_CUAKEY 875
Ph_NOT_HOTKEY 875
Ph_PACK_RAW 1044, 1178
Ph_PACK_STRING 1044, 1178
Ph_PACK_STRUCT 1044, 1178
Ph_PRINT_REGION 980
Ph_PTR_REGION 980, 994
Ph_QUERY_CONSOLE 1058
Ph_QUERY_EXACT 1059
Ph_QUERY_GRAPHICS 1058
Ph_QUERY_INPMGRINFO 1004
Ph_QUERY_INPUT_GROUP 1058
Ph_QUERY_WORKSPACE 1059
Ph_RDATA_CLIPBOARD 1004
Ph_RDATA_CURSOR 1004
Ph_RDATA_GFXDETAIL 1004
Ph_RDATA_GFXINFO 1004
Ph_RDATA_KBDINFO 1004
Ph_RDATA_INPUT_GROUP 1004
Ph_RDATA_POINTERINFO 1004
Ph_RDATA_PTRINFO 1004
Ph_RDATA_USER 1004
Ph_RDATA_WINDOW 1004
Ph_RDATA_WMCONFIG 1004
Ph_REGION_BEHIND 1009
Ph_REGION_CURSOR 1009
Ph_REGION_DATA 1009
Ph_REGION_EV_OPAQUE 1009
Ph_REGION_EV_SENSE 1009
Ph_REGION_FLAGS 1009
Ph_REGION_HANDLE 481, 1009
Ph_REGION_IN_FRONT 1009
Ph_REGION_ORIGIN 1009
Ph_REGION_OWNER 1009
Ph_REGION_PARENT 1009
Ph_REGION_RECT 1009
Ph_RELEASE_GHOST_BITMAP 923, 1018

1960 Index
<table>
<thead>
<tr>
<th>Key</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ph_RELEASE_IMAGE</td>
<td>923, 1018</td>
</tr>
<tr>
<td>Ph_RELEASE_IMAGE_ALL</td>
<td>923, 1018</td>
</tr>
<tr>
<td>Ph_RELEASE_PALETTE</td>
<td>923, 1018</td>
</tr>
<tr>
<td>Ph_RELEASE_TRANSPARENCYMASK</td>
<td>923, 949, 1018</td>
</tr>
<tr>
<td>Ph_RESIZE_MSG</td>
<td>889, 891, 909</td>
</tr>
<tr>
<td>Ph_RIDQUERY_IG_POINTER</td>
<td>979</td>
</tr>
<tr>
<td>Ph_RIDQUERY_TOWARD</td>
<td>979</td>
</tr>
<tr>
<td>Ph_ROOT_RID</td>
<td>898, 1009</td>
</tr>
<tr>
<td>Ph_START_DRAW</td>
<td>609</td>
</tr>
<tr>
<td>Ph_SUPRESS_PARENT_CLIP</td>
<td>839</td>
</tr>
<tr>
<td>Ph_SYNC_GCS</td>
<td>839</td>
</tr>
<tr>
<td>Ph_SYSTEM_REGION_CHANGE</td>
<td>878</td>
</tr>
<tr>
<td>Ph_TEXT_EXTENTS</td>
<td>838</td>
</tr>
<tr>
<td>Ph_TRACK_BOTTOM</td>
<td>855, 928</td>
</tr>
<tr>
<td>Ph_TRACK_DRAG</td>
<td>855, 928</td>
</tr>
<tr>
<td>Ph_TRACK_LEFT</td>
<td>855, 928</td>
</tr>
<tr>
<td>Ph_TRACK_RIGHT</td>
<td>855, 928</td>
</tr>
<tr>
<td>Ph_TRACK_TOP</td>
<td>855, 928</td>
</tr>
<tr>
<td>Ph_TRANSPORT_ENDIANIAN_OK</td>
<td>1690</td>
</tr>
<tr>
<td>Ph_TRANSPORT_FILE_STREAM</td>
<td>965, 969, 1178, 1688, 1695</td>
</tr>
<tr>
<td>Ph_TRANSPORT_FILEREF</td>
<td>795, 964, 968, 1048, 1177, 1688, 1694</td>
</tr>
<tr>
<td>Ph_TRANSPORT_INLINE</td>
<td>795, 964, 968, 1047, 1177, 1688, 1694</td>
</tr>
<tr>
<td>Ph_TRANSPORT_NAMED_STREAM</td>
<td>965, 969, 1177, 1688, 1694</td>
</tr>
<tr>
<td>Ph_TRANSPORT_SHMEM</td>
<td>795, 965, 968, 1048, 1177, 1688, 1694</td>
</tr>
<tr>
<td>Ph_TRANSPORT_STREAM</td>
<td>965, 969, 1177, 1688, 1694</td>
</tr>
<tr>
<td>Ph_TYPE_SPECIFIC</td>
<td>864</td>
</tr>
<tr>
<td>Ph_USE_TRANSPARENCY</td>
<td>923</td>
</tr>
<tr>
<td>Ph_USER_RSVDBITS</td>
<td>864</td>
</tr>
<tr>
<td>Ph_WINDOW_REGION</td>
<td>979, 994</td>
</tr>
<tr>
<td>Ph_WM_BACKDROP</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_CLOSE</td>
<td>1054</td>
</tr>
<tr>
<td>Ph_WM_CONSWITCH</td>
<td>1055</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_FFRONT</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_FFRONT_DISABLE</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_FOCUS</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_FOCUSLOST</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_HIDE</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_INVERSE</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_MENU</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_MENU_FINISH</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_PERFORM</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_EVSTATE_UNHIDE</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_FFRONT</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_FOCUS</td>
<td>1054, 1056</td>
</tr>
<tr>
<td>Ph_WM_HELP</td>
<td>1055</td>
</tr>
<tr>
<td>Ph_WM_HIDE</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_MAX</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_MENU</td>
<td>1054, 1056</td>
</tr>
<tr>
<td>Ph_WM_MOVE</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_RESIZE</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_RESTORE</td>
<td>1055, 1056</td>
</tr>
<tr>
<td>Ph_WM_STATE_ISBACKDROP</td>
<td>1056</td>
</tr>
<tr>
<td>Ph_WM_STATE_ISHIDDEN</td>
<td>1055</td>
</tr>
</tbody>
</table>
Index

Ph_WM_STATE_ISICONIFIED 1055
Ph_WM_STATE_ISMAX 1055
Ph_WM_STATE_ISNORMAL 1055
Ph_WM_STATEognito 1055
Ph_WM_SUPERSELECT 1453
Ph_WM_TOBACK 1055
Ph_WM_TOFRONT 1054
Ph_WND_MGRREGION 980, 994
PhAB
  application as DLL 116, 194
PhAB callback information
  structure 165
PhAddMergeTiles() 792
PhAllocPackType() 795
PhArea_t 797
PhAreaToRect() 798
PhAttach() 799
PhBitmapCursorDescription_t 805
PhBlit() 803
PhCancelDrag() 808
PhChannelAttach() 810
PhChannelParms_t 800
PhCharacterCursorDescription_t 814
PhClipboardCopyString() 816
PhClipboardHdr 818, 1004
PhClipboardPasteString() 819
PhClipboardRead() 821
PhClipboardWrite() 823
PhClipTilings() 826
PhCoalesceTiles() 828
PhConnectInfo_t 906
PhCopyTiles() 830
PhCreateImage() 832
PhCreateTransportCtrl() 834
PhCursorDef_t 835, 1004
PhCursorDescription_t 837
PhCursorInfo_t 976
PhDCCreate() 840
PhDCGetCurrent() 841
PhDCRelease() 842
PhDCSetCurrent() 844
PhDetach() 847
PhDeTranslateRect() 849
PhDeTranslateTiles() 850
PhDim_t 852
PhDragEvent_t 853, 870
PhDrawEvent_t 871
PhEmit() 857
PhEmitmx() 860
PhEvent_t 863
PhEventArm() 881
PhEventEmit() 883
PhEventEmitmx() 886
PhEventNext() 889
PhEventPeek() 891
PhEventRead() 894
PhEventRegion_t 898
PhFEPInfo_t 873
PhFEPIService_t 877
PhFindTransportType() 900
PHFONT 267
PHFONT_ALL_FONTS 398, 1449
PHFONT_ALL_SYMBOLS 398
PHFONT_BITMAP 398, 1449
PHFONT_DONT_SHOW_LEGACY 398
PHFONT_FIXED 398, 1449
PHFONT_INFO_ALIAS 396, 400
PHFONT_INFO_BLDITC 396, 400
PHFONT_INFO_BOLD 396, 400

1962  Index
Index

PHFONT_INFO_DECORATIVE 396, 400
PHFONT_INFO_FIXED 396, 400
PHFONT_INFO_ITALIC 396, 400
PHFONT_INFO_PLAIN 396, 400
PHFONT_INFO_PROP 396, 400
PHFONT_INFO_SANSERIF 396, 400
PHFONT_INFO_SERIF 396, 400
PHFONT_LOAD_IMAGES 391
PHFONT_LOAD_METRICS 391
PHFONT_PROP 398, 1449
PHFONT_SCALABLE 398, 1449
PHFONTMEM 267
PhFreeTiles() 901
PhFreeTransportType() 902
PhGetAllTransportHdtrs() 904
PhGetConnectId() 905
PhGetConnectInfo() 906
PhGetData() 908
PhGetMsgSize() 909
PhGetNextInlineData() 911
PhGetNextTransportHdr() 913
PhGetRects() 915
PhGetTile() 917
PhGetTransportHdr() 919
PhGetTransportVectors() 920
PhGrafxDetail_t 1004
PhGrafxInfo_t 1024
PhGrafxRegionData_t 1004
PhInfo_t 1024
PhImage_t 922
PhInitDrag() 930
PhInputGroup() 933
PhIntersectTilings() 934
PhKbdInfo_t 1024
PhKbdRegionData_t 1004
PhKeyEvent_t 936
PhKeyToMb() 940
PhLibVersion() 941
PhLinkTransportData() 942
PhLocateTransHdr() 944
PhMakeGhostBitmap() 946
PhMakeTransBitmap() 948
PhMakeTransparent() 953
PhMallocUnpack() 957
PhMergeTiles() 959
PhMoveCursorAbs() 961
PhMoveCursorRel() 962
PHOTON 799, 1591
Photon channels
  arming for events 881
  attaching 799, 1514
  connection information 906
  detaching 847
  events
    checking for 891
    reattaching 984
Photon server, sending an application
  to another 1591
PhPackEntry() 963
PhPackType() 967
PhPoint_t 972
PhPointerEvent_t 973
PhPtrInfo_t 1024
PhPtrRegionData_t 1004
PhQueryCursor() 976
PhQueryRids() 979
PhQuerySystemInfo() 982
PhReattach() 984
PhRect_t 986
PhRectIntersect() 987
Index

PhRectsToTiles() 988
PhRectToArea() 990
PhRectUnion() 991
PhRegion_t 992
PhRegionChange() 997
PhRegionClose() 1001
PhRegionDataFindType() 1002
PhRegionDataHdr_t 1004
PhRegionInfo() 1006
PhRegionOpen() 1009
PhRegionQuery() 1014
PhRegisterTransportType() 1016
PhReleaseImage() 1018
PhReleaseTransportCtrl() 1020
PhReleaseTransportHdres() 1021
PhRemoteWMEvent_t 876
phs-to-* 1127
PhSortTiles() 1022
PhSysInfo_t 1024
PhTile_t 1029
PhTilesBoundingRect() 1030
PhTilesToRects() 1032
PhTimerArm() 1034
PhTo8859_1() 1036
PhTranslateRect() 1037
PhTranslateTiles() 1039
PhTransportCtrl_t 1041
PhTransportFindLink() 1042
PhTransportLink_t 1043
PhTransportRegEntry_t 1044
PhTransportType() 1047
PhUnlinkTransportHdr() 1051
PhUnpack() 1052
PhWindowEvent_t 1054
PhWindowInfo_t 1004
PhWindowQueryVisible() 1058
Pi_FREE 1064, 1066, 1068
Pi_HORIZONTAL 1068
Pi_SHMEM 1064, 1066, 1068
Pi_VERTICAL 1068
PiCropImage() 1064
picture modules, creating 132
PiDuplicateImage() 1066
PiFlipImage() 1068
PiGetPixel() 1070
PiGetPixelFromData() 1072
PiGetPixelRGB() 1074
pipes, FD handler 1191, 1229, 1240
PiSetPixel() 1076
PiSetPixelInData() 1078
pixels
drawing 555, 557
getting from a block of data 1072
getting in an image 1070, 1074
setting in a block of data 1078
setting in an image 1076
Pk_KF_Cap_Valid 937
Pk_KF_Compose 937
Pk_KF_Key_Down 937
Pk_KF_Key_Repeat 937
Pk_KF_Scan_Valid 937
Pk_KF_Sym_Valid 937
Pk_KM_Alt 854, 936, 974
Pk_KM_Alt_Lock 854, 936, 975
Pk_KM_AltGr 854, 936, 974
Pk_KM_AltGr_Lock 854, 936, 975
Pk_KM_Caps_Lock 854, 936, 975
Pk_KM_Ctrl 854, 936, 974
Pk_KM_Ctrl_Lock 854, 936, 975
Pk_KM_Mod6 854, 936, 974
Pk_KM_Mod6_Lock 854, 936, 975
Pk_KM_Mod7 854, 936, 974
Index

Pk_KM_Mod7_Lock  854, 936, 975
Pk_KM_Mod8  854, 936, 974
Pk_KM_Num_Lock  854, 936, 975
Pk_KM_Scroll_Lock  854, 937, 975
Pk_KM_Shift  854, 936, 974
Pk_KM_Shift_Lock  854, 936, 975
Pk_KM_Sh13  854, 936, 974
Pk_KM_Sh13_Lock  854, 936, 975
PkKeyDef.h  938
plane mask
   resetting  494
   setting  731
Pm_IMAGE_CONTEXT  1095
Pm_PHS_CONTEXT  1095
PmMemCreateMC()  1084
PmMemFlush()  1089
PmMemReleaseMC()  1091
PmMemSetChunkSize()  1093
PmMemSetMaxBufSize()  1094
PmMemSetType()  1095
PmMemStart()  1097
PmMemStop()  1099
point, data structure  972
pointers
   information about  976
   moving
      absolute position  961
      relative position  962
polygons, drawing  559
power-saving mode, setting  693
Pp_clearbit()  1125
Pp_PC_COLLATING_MODE  1114, 1124, 1144
Pp_PC_COLOR_MODE  1114, 1124, 1144
Pp_PC_CONTROL  1114, 1125, 1144
Pp_PC_COPIES  1114, 1126, 1144
Pp_PC_DATE  1114, 1126, 1144
Pp_PC_DEVICE  1114, 1126, 1144, 1147
Pp_PC_DITHERING  1114, 1127, 1144
Pp_PC_DO_PREVIEW  1114, 1127, 1144
Pp_PC_DRIVER  1114, 1127, 1144
Pp_PC_DUPLEX  1114, 1127, 1144
Pp_PC_FILENAME  1114, 1128, 1144, 1147
Pp_PC_INKTYPE  1114, 1128, 1144
Pp_PC_INTENSITY  1114, 1128, 1144
Pp_PC_JOB_NAME  1114, 1128, 1144
Pp_PC_MARGINS  1114, 1128, 1144
Pp_PC_MAX_DEST_SIZE  1114, 1129, 1144
Pp_PC_NAME  1114, 1129, 1144, 1147
Pp_PC_NONPRINT_MARGINS  1114, 1129, 1144
Pp_PC_ORIENTATION  1114, 1129, 1144
Pp_PC_PAGE_NUM  1114, 1129, 1130, 1144
Pp_PC_PAGE_RANGE  1114, 1129, 1144
Pp_PC_PAPER_SIZE  1114, 1130, 1144

Index 1965
Index

Pp_PC_PAPER_SOURCE 1114, 1130, 1144
Pp_PC_PAPER_TYPE 1114, 1131, 1144
Pp_PC_PREVIEW_APP 1114, 1131, 1144
Pp_PC_PRINTER_RESOLUTION 1114, 1131, 1144
Pp_PC_PROP_APP 1114, 1131, 1144
Pp_PC_REVERSED 1114, 1131, 1144
Pp_PC_SCALE 1114, 1132, 1144
Pp_PC_SOURCE_COLORS 1114, 1132, 1144
Pp_PC_SOURCE_OFFSET 1114, 1132, 1144
Pp_PC_SOURCE_RESOLUTION 1114, 1133, 1144
Pp_PC_SOURCE_SIZE 1114, 1133, 1144
Pp_PC_USER_ID 1114, 1133, 1144
Pp_resetbits() 1125
Pp_setbit() 1125
Pp_testbit() 1125
PpContinueJob() 1104
PpCreatePC() 1107
PpEndJob() 1109
PpFreePrinterList() 1111
PpGetCanvas() 1112
PpGetPC() 1114
PpLoadDefaultPrinter() 1119
PpLoadPrinter() 1121
PpLoadPrinterList() 1123
PpPageRange_t 1129
PpPCControl_t 1125
PpPrintContext_t 1124
PpPrintNewPage() 1134
PpPrintWidget() 1137
PpReleasePC() 1139
PpSetCanvas() 1141
PpSetPC() 1143
PpStartJob() 1147
PpSuspendJob() 1149
printing
  canvas
    getting 1112
    setting 1141
collating mode 1124
color mode 1124
completing 1109
contexts
  activating 1104
  attributes 1124
  attributes, modifying 1143
  attributes, querying 1114
  creating 1107
deactivating 1149
initializing 1119, 1121
releasing 1139
control (PpPCControl_t) 1125
date 1126
dithering 1127
duplex 1127
filter application 1127
initializing 1147
ink type 1128
intensity 1128
job name 1128
margins 1128
nonprintable margins 1129
number of copies 1126
Index

orientation 1129
deleating 1648
page break 1134
prompts 1553, 1580
page number 1129, 1130
Pt_ARG_FLAGS 1163, 1332,
paper size 1130
1334, 1336, 1478, 1589
parameters, modifying 1559
paper source 1130
preview 1127, 1131
paper type 1131
printer name 1129
parameters, modifying 1559
printer properties 1131
preview 1127, 1131
printers, list of available
printer resolution 1131
freecing 1111
printers, list of available
loading 1123
source
scale 1132
selecting options and
initiating 1569, 1571

source

colors 1132
offset 1132
resolution 1133

size 1133

spooler 1126
temporary file size 1129
to a file 1128
user ID 1133
widgets 1137

processes
ID, getting from receive ID
(RCVID) 1463, 1465
spawning 1645
spawning and waiting 1650
termination callback
adding 1649
Pt_ARG_FLAGS 1163, 1332,
adding 1163, 1166
removing 1601, 1603
Pt_CENTER 1181, 1547, 1552, 1580
Pt_CLEAN_RESOURCES 1716
Pt_CLEAR 1395
Pt_CONNECTION_CLIENT_BROKEN 1300
Pt_CONNECTION_MSGREAD_FAILED 1316
Pt_CONNECTION_NOTIFY_FLUSH 1307
Pt_CONNECTION_NOTIFY_NOFLUSH 1307
Pt_CONNECTION_NOTIFY_RESIZE 1307
Pt_CONNECTION_REALLOC_RECEIVE 1316
Pt_CONNECTION_REALLOC_REPLY 1300
Pt_CONNECTION_REPLY_FAILED 1316
Pt_CONNECTION_SEND_FAILED 1300
Pt_CONNECTION_SERVER_BROKEN 1316
Pt_CONSUME 1361
Pt_CONTAINER 1716
Pt_DAMAGE_SURFACE 1380, 1382, 1391, 1393
Pt_DEFAULT_PARENT 1364, 1625
Pt_DELAY_EXIT 1396, 1525
Pt_DELAY_REALIZE 1589
Pt_DESTROYED 1378
Pt_DISJOINT 1433, 1716
Pt_DND_CMD_PROVIDE_DATA 1690
Pt_DND_LOCAL 1274, 1517
Pt_DND_REQUEST_DATA 1689, 1690
Pt_DND_SELECT_DATA_DUP 1384
Pt_DND_SELECT_DUP_DATA 1384
Pt_DND_SELECT_MOTION 869, 1385
Pt_DND_SELECT_MULTIPLE 1385
Pt_DND_SILENT 1274, 1517
Pt_ESC_DISABLE 1182, 1547, 1552, 1580
Pt_EVENT_PROCESS_ALLOW 171, 1396, 1525, 1533
Pt_EVENT_PROCESS_PREVENT 171, 1396, 1525, 1533
Pt_FD_OBAND 1190, 1240
Pt_FD_READ 1190, 1240
Pt_FD_WRITE 1190, 1240
Pt_FEP_PRESENT 1456
Pt_FEP_QUERIED 1456
Pt_FORCE_UNREALIZE 1716
Pt_FSDIALOG_BTN1 1410, 1415
Pt_FSDIALOG_BTN2 1410, 1415
Pt_FSR_CASE_INSENSITIVE 1412
Pt_FSR_CONFIRM_EXISTING 1411
Pt_FSR_CREATE_PATH 1411
Pt_FSR_DONT_SHOW_DIRS 1412
Pt_FSR_DONT_SHOW_FILES 1412
Pt_FSR_FREE_ON_COLLAPSE 1412
Pt_FSR_MULTIPLE 1410
Pt_FSR_NO_CONFIRM_CREATE_PATH 1411
Pt_FSR_NO_CONFIRM_DELETE 1411
Pt_FSR_NO_DELETE 1411
Pt_FSR_NO_ERROR_POPUP 1413
Pt_FSR_NO_FCHECK 1410
Pt_FSR_NO_FSPEC 1410
Pt_FSR_NO_NEW 1411
Pt_FSR_NO_NEW_BUTTON 1411
Pt_FSR_NO_ROOT_DISPLAY 1412
Pt_FSR_NOSEEK_KEY 1412
Pt_FSR_NO_SELECT_FILES 1411
Pt_FSR_NO_UP_BUTTON 1411
Pt_FSR_RECURSIVE_DELETE 1411
Pt_FSR_SELECT_DIRS 1411, 1414
Pt_FSR_SHOW_ERRORS 1412
Pt_FSR_SHOW_HIDDEN 1412
Pt_FSR_TREE 1412
Pt_GETS_FOCUS 1163, 1332, 1334, 1336, 1478
Pt_GHOST 550, 553, 577, 946
Pt_HOTKEY_CHAINED 1173
Pt_HOTKEY_IGNORE_Mods 1172
Pt_HOTKEY_SYM 1172
Pt_IMMEDIATE_CHILD 1279
Pt_INDEX_EXPOSE 1456
Pt_INDEX_RESOURCES 1716
Pt_LEFT 1181, 1547, 1552, 1580
Pt_MODAL 1182, 1547, 1552, 1580
Pt_MULTITEXT 1580
Pt_NO_INHERITED_RESOURCES 1717
Pt_NO_PARENT 1364, 1625
Pt_OBSCURED 1395
Pt_OCCUSIVE 1717
Pt_PP_NO_RESIZE 1136
Pt_PP_RESIZE_PC 1137
Pt_PP_RESIZE_WIDGET 1136
Pt_PRINTSEL_ALL_PANES 1574
Pt_PRINTSEL_CANCEL 1569, 1574
Pt_PRINTSEL_DFLT_LOOK 1574
Pt_PRINTSEL_ERROR 1570
Pt_PRINTSEL_FILE_PANE 1573
Pt_PRINTSEL_NO_COPIES 1573
Pt_PRINTSEL_NO_PAGE_RANGE 1573
Pt_PRINTSEL_NO_PRINTSELECT 1573
Pt_PRINTSEL_NO_SELECT_RANGE 1573
Pt_PRINTSEL_PREFERENCES 1573
Pt_PRINTSEL_PREVIEW 1569, 1574
Pt_PRINTSEL_PRINT 1569, 1574
Pt_PRINTSEL_SETTINGS_PANE 1574
Pt_PROCREATED 1438
Pt_PSP_CANCEL 1566
Pt_PSP_DONE 1566
Pt_PSP_ERROR 1565
Pt_PWD_ACCEPT 1552, 1553
Pt_PWD_CANCEL 1553
Pt_PWD_REJECT 1552, 1553
Pt_PWD_RETRY 1552
Pt_RECTANGULAR 1717
Index

Pt_RELATIVE  1182, 1547, 1552, 1580
Pt_RESIZE_XY_ALWAYS  1738
Pt_RIGHT  1181, 1547, 1552, 1580
Pt_STYLE_ACTIVATE  1473, 1632
Pt_STYLE_CALC_BORDER  1474, 1632
Pt_STYLE_CALC_OPAQUE  1474, 1632
Pt_STYLE_DATA  1474, 1633
Pt_STYLE_DEACTIVATE  1474, 1632
Pt_STYLE_DRAW  1473, 1631
Pt_STYLE_EXTENT  1473, 1631
Pt_STYLE_NAME  1474, 1633
Pt_SUBORDINATES_CHILD  1279
Pt_SURFACE_CONSUME_EVENTS  1354, 1360, 1380, 1382
Pt_SURFACE_DISABLED  1354, 1360
Pt_SURFACE_ELLIPSE  1355, 1360
Pt_SURFACE_HIDDEN  1354, 1359
Pt_SURFACE_PARENT_RELATIVE  1354, 1360
Pt_SURFACE_RECT  1355, 1360
Pt_SURFACE_RELEASE_POINTS  1354, 1359
Pt_SURFACE_USE_ORIGIN  1354, 1355, 1360
Pt_TRAVERSE_BACK  1751
Pt_TRAVERSE_DONE  1748
Pt_TRAVERSE_FORCE  1751
Pt_TRAVERSE_LAST  1750
Pt_TRAVERSE_ROOT  1750
Pt_TRAVERSE_START  1748, 1750
Pt_UNCLEAN_RESOURCES  1717
Pt_WAIT  1547
PtAddCallback()  1154
PtAddCallbacks()  1157
PtAddClassStyle()  1159
PtAddData()  1161
PtAddEventHandler()  1163
PtAddEventHandlers()  1166
PtAddFilterCallback()  1168
PtAddFilterCallbacks()  1170
PtAddHotkeyHandler()  1172
PtAddResponseType()  1176
PtAlert()  1182
PtAllowExit()  1184
PtAppAddCallback()  1186
PtAppAddEventHandlers()  1188, 1196
PtAppAddFilterCallback()  1193
PtAppAddInput()  1197
PtAppAddInputRemote()  1203
PtAppAddSignalProc()  1207
PtAppAddWorkProc()  1209
PtAppCreatePulse()  1212
PtAppDeletePulse()  1214
PtAppGetResource()  1216
PtAppGetResources()  1219
PtAppInit()  1221
PtAppPulseTrigger()  1223
PtAppRemoveCallback()  1225
PtAppRemoveEventHandlers()  1227
PtAppRemoveFd()  1229
PtAppRemoveFilterCallback()  1231
PtAppRemoveHotkeyHandler()  1233
PtAppRemoveInput()  1235
PtAppRemoveSignal()  1236
Index

PtAppRemoveWorkProc() 1238
PtAppSetFdMode() 1240
PtAppSetResources() 1244
PtArg_t 1252
PtBkgdHandlerProcess() 1255
PtBlit() 1257
PtBlockAllWindows() 1259
PtBlockWindow() 1261
PtCalcAbsPosition() 1263
PtCalcCanvas() 1266
PtCalcSurface() 1268
PtCalcSurfaceByAction() 1270
PtCalcSurfaceById() 1272
PtCancelDnd() 1274
PtChannelCreate() 1276
PtCheckSurfaces() 1277
PtChildType() 1279
PtClearWidget() 1281
PtClipAdd() 1284
PtClippedBlit() 1286
PtClipRemove() 1288
PtComboBox functions 67
PtCondTimedWait() 1289
PtCondWait() 1291
PtConnectionAddEventHandlers() 1293
PtConnectionAddMsgHandlers() 1295
PtConnectionClientDestroy() 1298
PtConnectionClientGetUserData() 1299
PtConnectionClientSetError() 1200, 1205, 1300
PtConnectionClientSetUserData() 1302
PtConnectionFindId() 1303
PtConnectionFindName() 1304
PtConnectionFlash() 1305
PtConnectionNotify() 1307
PtConnectionReply(), PtConnectionReplymx() 1309
PtConnectionResizeEventBuffer() 1311
PtConnectionSend(), PtConnectionSendmx() 1312
PtConnectionServerDestroy() 1314
PtConnectionServerGetUserData() 1315
PtConnectionServerSetUserData() 1318
PtConnectionTmpName() 1319
PtConnectionWaitForName() 1320
PtConnectorCreate() 1323
PtConnectorDestroy() 1325
PtConnectorGetId() 1326
PtConsoleSwitch() 1327
PtContainerBox() 1329
PtContainerFindFocus() 1331
PtContainerFocusNext() 1332
PtContainerFocusPrev() 1334
PtContainerGiveFocus() 1336
PtContainerHit() 1341
PtContainerHold() 1343
PtContainerNullFocus() 1345
PtContainerRelease() 1347
PtCRC() 1349
PtCRCValue() 1351
PtCreateActionSurface() 1353
PtCreateClassStyle() 1357
PtCreateSurface() 1359
PtCreateTransportCtrl() 1363
PtCreateWidget() 1364
Index

PtDamageExtent() 1367
PtDamageSurface() 1369
PtDamageSurfaceByAction() 1371
PtDamageSurfaceById() 1369
PtDamageWidget() 1372
PtDestroyAllSurfaces() 1374
PtDestroySurface() 1375
PtDestroySurfaceById() 1376
PtDestroyWidget() 1378
PtDisableSurface() 1380
PtDisableSurfaceByAction() 1382
PtDisableSurfaceById() 1380
PtDndFetch_t 869, 1384
PtDndSelect() 1388
PtDupClassStyle() 1389
PtEnableSurface() 1391
PtEnableSurfaceByAction() 1393
PtEnableSurfaceById() 1391
PtEndFlux() 1395
PtEnter() 1396
PtEventHandler() 1401
PtExit() 1402
PtExtentWidget() 1404
PtExtentWidgetFamily() 1405
PtFepCmd_t, PtFepProc_t 1406
PtFepCmd() 1408
PtFileSel functions 67
PtFileSelection() 1413
PtFileSelectionInfo_t 1415
PtFindChildClassInfo_t 1419
PtFindChildClass() 1423
PtFindChildClassMember() 1426
PtFindClassStyle() 1428
PtFindContainer() 1430
PtFindData() 1431
PtFindDisjoint() 1433
PtFindFocusChild() 1435
PtFindFocusNextFrom() 1436
PtFindFocusPrevFrom() 1437
PtFindGuardian() 1438
PtFindNextData() 1440
PtFindSurface() 1442
PtFindSurfaceByAction() 1444
PtFlush() 1446
PtFontSelection() 1448
PtForwardWindowEvent() 1451
PtForwardWindowTaskEvent() 1453
PtFSFreeInfo() 1419
PtGenList functions 69
PtGenTree functions 72
PtGetAbsPosition() 1455
PtGetControlFlags() 1456
PtGetDndFetchIndex() 1457
PtGetParent() 1460
PtGetParentWidget() 1462
PtGetRcvidPid() 1463
PtGetRcvidPidNd() 1465
PtGetResource() 1467
PtGetResources() 1470
PtGetStyleMember() 1473
PtGetWidgetStyle() 1476
PtGiveFocus() 1478
PtGlobalFocusNext() 1480
PtGlobalFocusNextContainer() 1482
PtGlobalFocusNextFrom() 1484
PtGlobalFocusPrev() 1486
PtGlobalFocusPrevContainer() 1488
PtGlobalFocusPrevFrom() 1490
PtHelpQuit() 1492
PtHelpSearch() 1494
PtHelpTopic() 1496
PtHelpTopicRoot() 1498
PtHelpTopicTree() 1500
Index

PtHelpUrl() 1502
PtHelpUrlRoot() 1504
PtHideSurface() 1506
PtHideSurfaceByAction() 1508
PtHideSurfaceById() 1506
PtHit() 1509
PtHold() 1510
PtInflateBalloon() 1512
PtInit() 1514
PtInitDnd() 1517
PtInputCallbackProcF_t, PtInputCallbackProc_t 1519
PtInsertSurface(), PtInsertSurfaceById() 1520
PtIsFluxing() 1522
PtIsFocused() 1523
PtLeave() 1526
PtList functions 74
PtMainLoop() 1529
PtMakeModal() 1530
PtMenuButton 1555
PtMessageBox() 1531
PtModalBlock() 1533
PtModalCtrl_t 1533
PtModalEnd() 1540
PtModalStart() 1541
PtModalUnblock() 1543
PtMTrend functions 75
PtMultiText functions 76
PtNextTopLevelWidget() 1545
PtNotice() 1548
PtPanelGroup functions 77
PtPassword() 1553
PtPositionMenu() 1555
PtPreventExit() 1557
PtPrintPropSelect() 1559
PtPrintPropSelectionInfo_t 1560
PtPrintSelect() 1569
PtPrintSelection() 1571
PtProcessEvent() 1576
PtProgress functions 77
PtPrompt() 1580
PtPulseArm() 1583
PtQuerySystemInfo() 1585
PtQuitMainLoop() 1587
PtRealizeWidget() 1589
PtReattach() 1591
PtRelease() 1593
PtReleaseTransportCtrl() 1594
PtRemoveCallback() 1595
PtRemoveCallbacks() 1597
PtRemoveData() 1599
PtRemoveEventHandler() 1601
PtRemoveEventHandlers() 1603
PtRemoveFilterCallback() 1605
PtRemoveFilterCallbacks() 1607
PtRemoveHotkeyHandler() 1609
PtReparentWidget() 1611
PtReqResponseHdr_t 1690
PtReRealizeWidget() 1613
PtResizeEventMsg() 1615
PtSendEventToWidget() 1617
PtSetAreaFromCanvas() 1619
PtSetArg() 1621
PtSetClassStyleMethods() 1623
PtSetParentWidget() 1625
PtSetResource() 1627
PtSetResources() 1629
PtSetStyleMember() 1631
PtSetStyleMembers() 1635
PtSetWidgetStyle() 1637
Index

PtShowSurface() 1639
PtShowSurfaceByAction() 1641
PtShowSurfaceById() 1639
PtSignalProcF_t
  PtSignalProc_t 1643
PtSpawn() 1645
PtSpawnDeleteCallback() 1648
PtSpawnOptions_t 1645
PtSpawnSetCallback() 1649
PtSpawnWait() 1650
PtStartFlux() 1652
PtStyleMethods_t 1623
PtSurfaceActionId() 1653
PtSurfaceAddData(),
  PtSurfaceAddDataById() 1654
PtSurfaceBrotherBehind() 1656
PtSurfaceBrotherInFront() 1657
PtSurfaceCalcBoundingBox(),
  PtSurfaceCalcBoundingBoxById() 1658
PtSurfaceExtent(),
  PtSurfaceExtentById() 1660
PtSurfaceGetData() 1662
PtSurfaceHit() 1664
PtSurfaceId() 1666
PtSurfaceInBack() 1667
PtSurfaceInFront() 1668
PtSurfaceIsDisabled() 1669
PtSurfaceIsEnabled() 1670
PtSurfaceIsHidden() 1671
PtSurfaceIsShown() 1672
PtSurfaceRect(),
  PtSurfaceRectById() 1673
PtSurfaceRemoveData(),
  PtSurfaceRemoveDataById() 1675
PtSurfaceTestPoint() 1677
PtSurfaceToBack(),
  PtSurfaceToBackById() 1678
PtSurfaceToFront(),
  PtSurfaceToFrontById() 1680
PtSyncWidget() 1682
PtTerminal functions 78
PtText functions 79
PtTimerArm() 1683
PtTransportCtrl_t 1685
PtTransportReqDataCB_t
  PtTransportRequestable() 1689
PtTransportType() 1695
PtTree functions 79
PtTrend functions 79
PtTty functions 81
PtUnblockWindows() 1697
PtUnlinkData() 1698
PtUnrealizeWidget() 1699
PtUpdate() 1701
PtValidParent() 1702
PtWidget
  canvas, determining 1266
PtWidgetActiveSurface() 1704
PtWidgetArea() 1705
PtWidgetBrotherBehind() 1706
PtWidgetBrotherInFront() 1709
PtWidgetChildBack() 1710
PtWidgetChildFront() 1712
PtWidgetClass() 1714
PtWidgetClassFlags() 1716
Index

PtWidgetDim() 1718
PtWidgetExtent() 1719
PtWidgetFamily() 1721
PtWidgetFlags() 1723
PtWidgetHelpHit() 1724
PtWidgetInsert() 1726
PtWidgetIsClass() 1728
PtWidgetIsClassMember() 1730
PtWidgetIsRealized() 1732
PtWidgetMinimumSize() 1733
PtWidgetOffset() 1734
PtWidgetParent() 1736
PtWidgetPreferredSize() 1738
PtWidgetRid() 1740
PtWidgetSkip() 1741
PtWidgetToBack() 1743
PtWidgetToFront() 1746
PtWidgetTree() 1748
PtWidgetTreeTraverse() 1750
PtWidgetVisibleExtent() 1754
PtWindow functions 82
PtWindowConsoleSwitch() 1755
PtWindowGetFrameSize() 1756
PtWorkProcF_t, PtWorkProc_t 1758

pulses, Photon
arming 1583
creating 1212
deleting 1214
delivering to yourself 1223
PX_DIRECT_COLOR 1832
PX_DODITHER 1832
PX_IMAGE 1833
PX_LOAD 1832
PX_NORMAL 1833
PX_PALETTE 1833
PX_QUERY 1832

PX_SUPPRESS_TAG 1832
PX_TRANSPARENT 1832
PX_USECOLORS 1832, 1834
PXCONFIG_CLEAR 1778
PXCONFIG_CREATE 1778
PXCONFIG_FMT_BOOL_ON 1805
PXCONFIG_FMT_BOOL_TRUE 1805
PXCONFIG_FMT_BOOL_YES 1805
PXCONFIG_FMT_CHAR_CHAR 1808
PXCONFIG_FMT_CHAR_HEX 1808
PXCONFIG_FMT_INT_DECIMAL 1814, 1817, 1820, 1823
PXCONFIG_FMT_INT_HEX 1814, 1817, 1820, 1823
PXCONFIG_FMT_STRING 1826
PXCONFIG_GLOBAL 1779
PXCONFIG_HOME 1779
PXCONFIG_READ 1778
PXCONFIG_UNLINK 1779
PXCONFIG_WRITE 1778
PxConfigClose(),
  PxConfigCloseCx() 1762
PxConfigDeleteEntry(),
  PxConfigDeleteEntryCx() 1764
PxConfigDeleteSection(),
  PxConfigDeleteSectionCx() 1766
PxConfigFirstSection(),
  PxConfigFirstSectionCx() 1768

Index 1975
<table>
<thead>
<tr>
<th>Function</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>PxConfigForceEmptySection()</td>
<td>1770</td>
</tr>
<tr>
<td>PxConfigForceEmptySectionCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigNextEntry()</td>
<td>1772</td>
</tr>
<tr>
<td>PxConfigNextEntryCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigNextSection()</td>
<td>1774</td>
</tr>
<tr>
<td>PxConfigNextSectionCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigNextString()</td>
<td>1776</td>
</tr>
<tr>
<td>PxConfigNextStringCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigOpen()</td>
<td>1779</td>
</tr>
<tr>
<td>PxConfigOpenCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadBool()</td>
<td>1784</td>
</tr>
<tr>
<td>PxConfigReadBoolCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadChar()</td>
<td>1786</td>
</tr>
<tr>
<td>PxConfigReadCharCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadDouble()</td>
<td>1788</td>
</tr>
<tr>
<td>PxConfigReadDoubleCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadInt()</td>
<td>1790</td>
</tr>
<tr>
<td>PxConfigReadIntCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadLLong()</td>
<td>1793</td>
</tr>
<tr>
<td>PxConfigReadLLongCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadLong()</td>
<td>1796</td>
</tr>
<tr>
<td>PxConfigReadLongCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadShort()</td>
<td>1799</td>
</tr>
<tr>
<td>PxConfigReadShortCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigReadString()</td>
<td>1802</td>
</tr>
<tr>
<td>PxConfigReadStringCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigSection()</td>
<td>1803</td>
</tr>
<tr>
<td>PxConfigSectionCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteBool()</td>
<td>1806</td>
</tr>
<tr>
<td>PxConfigWriteBoolCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteChar()</td>
<td>1809</td>
</tr>
<tr>
<td>PxConfigWriteCharCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteDouble()</td>
<td>1812</td>
</tr>
<tr>
<td>PxConfigWriteDoubleCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteInt()</td>
<td>1815</td>
</tr>
<tr>
<td>PxConfigWriteIntCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteLLong()</td>
<td>1818</td>
</tr>
<tr>
<td>PxConfigWriteLLongCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteLong()</td>
<td>1821</td>
</tr>
<tr>
<td>PxConfigWriteLongCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteShort()</td>
<td>1824</td>
</tr>
<tr>
<td>PxConfigWriteShortCx()</td>
<td></td>
</tr>
<tr>
<td>PxConfigWriteString()</td>
<td>1827</td>
</tr>
<tr>
<td>PxConfigWriteStringCx()</td>
<td></td>
</tr>
<tr>
<td>PxGetImageExtensions()</td>
<td>1829</td>
</tr>
<tr>
<td>PxLoadImage()</td>
<td>1831</td>
</tr>
<tr>
<td>PxMethods_t</td>
<td>1831</td>
</tr>
<tr>
<td>PxTerminalBuildCharsets()</td>
<td>1840</td>
</tr>
<tr>
<td>PxTerminalLoadCharsets()</td>
<td>1842</td>
</tr>
<tr>
<td>PxTerminalSaveCharsets()</td>
<td>1844</td>
</tr>
<tr>
<td>PxTerminate()</td>
<td>1846</td>
</tr>
</tbody>
</table>
PxTranslateFromUTF() 1847
PxTranslateList() 1850
PxTranslateSet() 1852
PxTranslateStateFromUTF() 1855
PxTranslateStateToUTF() 1857
PxTranslateToUTF() 1859
PxTranslateUnknown() 1862

R

realtime timers
  creating 1868
  deleting 1870
  getting time 1871
  setting expiration time 1873
receive ID (RCVID), getting node descriptor (ND) from
  1465
receive ID (RCVID), getting process ID (PID) from 1463,
  1465
rectangles
  area, converting from 798
  area, converting to 990
  data structure 986
  drawing 564
    beveled 511
    rounded 579
  intersection of 987
  union of 991
regions
  blitting 803
  changing definition of 997
  clipping 689, 726
  closing 1001
  console, switching to 1755
  cursor information 976
  data
    data structure 1004
    searching by type 1002
  data structure 992
dragging 808, 930
draw events, emitting 624, 733
events 863
data structure 898
extent, querying visible 1058
graphical contexts 483
graphics driver 480
opening 1009
querying 979, 1006, 1014
system information 982
  PhGeneralSysInfo_t 1024
  PhGrafxInfo_t 1024
  PhIgInfo_t 1024
  PhKbdInfo_t 1024
  PhPtrInfo_t 1024
timers 1034
  widgets 1740
  windows
    actions, data structure 1054
  REMOTE_FLAG_FIXED 876
  REMOTE_FLAG_INITIAL 876
  REMOTE_FLAG_IS_ORIGIN 877
  REMOTE_FLAG_NO_DIM 877
  REMOTE_WM_TITLE 876
  REMOTE_WM_WINDOW 876
resources
  argument lists 1252, 1253,
    1621
  getting 1467, 1470
Index

record file, closing 196
setting 1627, 1629
rounded rectangles, drawing 579
RtTimerCbF_t 1868
RtTimerCreate() 1868
RtTimerDelete() 1870
RtTimerGetTime() 1871
RtTimerSetTime() 1873

copying 1894
length, calculating 1880, 1888, 1898
searching 1882, 1890
searching backwards 1900
searching backwards, ignoring case 1886
searching, ignoring case 1884, 1896
width, calculating 419, 422, 428, 430

stroke attributes
caps 735
color 737
dash 739
dithering 742
joints 744
resetting 495
transparency 747
width 749
XOR color 751

styles
adding 1159
copying 1389
creating 1357
current
getting 1476
setting 1637
finding 1428
getting 1473
setting 1623, 1631, 1635

support lxxviii

synchronization
draw streams 785
vertical 787
video drivers 786

system information

S
screen, reading images from 659
shared memory
attaching 769
cleaning up 771
creating 773
destroying 775
detaching 777
SIGCHLD 1646
signal handlers
adding 1207
prototype 1643
removing 1236
spans, drawing 583
strings
clipboard, pasting from 819
drawing 587
extent, calculating 311, 319, 326, 341, 347, 367
rendering 404
UTF-8 character
bytes, counting 1878
comparing 1892
converting to wide character 1903

1978 Index
Index

connection 906
connection ID 905
PhGeneralSysInfo_t 1024
PhGrafxInfo_t 1024
PhIgInfo_t 1024
PhKbdInfo_t 1024
PhPtrInfo_t 1024
PhSysInfo_t 1024
region 982
widget 1585

T

technical support lxxviii
text
attributes
color 753
dithering 755
font 712
resetting 496
transparency 757
underline 763
XOR color 759
drawing 590, 595
multiline 546
extent, calculating 311, 319, 326, 341, 347, 367, 607
multiline 605
rendering 404
width, calculating 419, 422, 428, 430

threads
conditional variables 1289, 1291
Phonon library

locking 1396
unlocking 1526
tiles
clipping one list from another 826
combining 828
converting from rectangles 988
converting to rectangles 1030, 1032
copying a list 830
data structure 1029
determining intersection of two lists 934
freeing 901
getting 917
merging 959
merging two lists, eliminating overlap 792
sorting 1022
translating 850, 1039
timers
event, arming 1034, 1683
realtime
creating 1868
deleting 1870
getting time 1871
setting expiration time 1873
Tr_ENDIAN_ARRAY() 1045
Tr_ENDIAN_REF() 1045
Tr_ENDIAN() 1045
Tr_REF_ARRAY() 1045
Tr_REF_TYPE_ARRAY() 1045
Tr_REF_TYPE_REF_ARRAY() 1045
Tr_REF_TYPE() 1045
Tr_STRING() 1045
Tr_TYPE_ARRAY() 1045
Index

$Tr\_TYPE()$ 1045
translation
characters
from UTF-8 1847, 1855
installing 1852
listing supported 1850
tables, building 1840
tables, loading 1842
tables, saving 1844
to UTF-8 1857, 1859
unknown 1862
coordinates
  clearing 459
rectangles 849, 1037
setting 761
translation files
  appending 118
  changing to another 202
transparency
  bitmaps 757
  fill 708
  images 948, 953, 1832
  stroke 747
  text 757
transport data
  building I/O vector 920
control structure 1041, 1685
  creating 834, 1363
  freeing 1020
  releasing 1594
  getting next 911
headers
  extracting 904, 913, 919
  freeing 1021
  unlinking 1051
linked list entry 1043
linking into a list 942
packing 795, 963, 967, 1047, 1695
registry entries
  adding 1016
  data, freeing 902
  finding 900
  PhTransportRegEntry_t 1044
requestable, adding 1689
response chain, adding to 1176
searching 944, 1042
unpacking 1052
  using custom memory
  allocation 957
trend, drawing 602
typographical conventions lxxvi

U

UTF-8
  translating characters
    from 1847, 1855
    translating characters to 1857, 1859
UTF-8 character strings
  comparing 1892
  counting 1880, 1888, 1898
  searching 1882, 1890
    ignoring case 1884, 1896
    searching backwards 1900
    ignoring case 1886
  strings, copying 1894
UTF-8 characters
  bytes, counting 1878
converting to wide characters 1903
UTF8_LEN_MAX 1878
utf8len() 1878
utf8strblen() 1880
utf8strchr() 1882
utf8strichr() 1884
utf8strirchr() 1886
utf8strlen() 1888
utf8strnchr() 1890
utf8strncmp() 1892
utf8strndup() 1894
utf8strnichr() 1896
utf8strnlen() 1898
utf8strrchr() 1900
utf8towc() 1903
duplicating 229
locking 220, 222, 238, 240, 250
shared memory reference to 233
swapping 779
translation, setting 245
surfaces getting 237
video overlay
channels
creating 488
data structure 783
destroying 498
chroma key 621
data, getting next frame 658
scaler
capabilities, data
structure 670
capabilities, getting 626
configuring channels 469
properties 673

V
vertical synchronization 787
video drivers, waiting until idle 786
video memory
draw mode, setting 697
protecting 731
video modes
getting 628
information, getting 630
setting 767
supported, getting list of 634
video offscreen memory
contexts
creating 218
data structure 242
data, copying to another context 472, 475

W
wctolower() 1908
wctoutf8() 1910
wide characters
converting to lowercase 1908
converting to UTF-8 1910
extent, calculating 347
positions, calculating 323, 332
rendering 590, 595
widget databases
classes
adding 192
classes, adding 114
closing 121
creating widgets from 127,
130, 137, 140
images, extracting 151
opening 187, 189
saving 198
text, getting translated 160
widgets
copying 124
deleting 143
getting information 148
widgets
balloons, inflating 1512
blitting 1257, 1286
brother
behind 1706
in front 1709
child
backmost 1710
focusable, finding 1435
frontmost 1712
class 1714, 1728, 1730
classes
removing 192
classes, adding 114
control surface, active 1704
coordinates 1455
creating 1364
creating from a database 127,
130, 137, 140
custom
summary of functions 55
damage
flux count,
incrementing 1652
damaging
an area 1367
completely 1372
container, determining if in
flux 1522
default parent
creating children 1364
getting 1462
setting 1625
destroying 1378
all within a container 1281
determining relationship 1279
events, sending 1617
extent, getting 1719
family hierarchy
child of a given class,
finding 1423, 1426
container parent,
finding 1430
skipping 1741
traversing, front to
back 1748, 1750
traversing, top to
bottom 1721
finding
in an area 1329
the nth in a container 1509
the nth in an area 1341
flags 1723
class structure 1716
focus
container 1331, 1332, 1334,
1345
degree of, determining 1523
giving 1336, 1478
global, giving to next after focused widget 1480
global, giving to next after given widget 1484
global, giving to next before focused widget 1486
global, giving to next before given widget 1490
global, giving to next container’s widget 1482
global, giving to previous container’s widget 1488
next 1436
previous 1437
guardian 1438
guardian 1438
help topic, finding first with 1724
inserting 1726
instance name, getting 167
instance pointer within a module, getting 162
instance pointer, getting 90
library initialization 1514
menus
position, setting 1555
module instance pointer, getting 154
moving
to back 1743
to front 1746
parents
changing 1611
class, nearest 1460
getting 1736
valid, finding 1702
PhAB name, getting 185
position 1455
printing 1137
PtComboBox functions 67
PtFileSel functions 67
PtGenList functions 69
PtGenTree functions 72
PtList functions 74
PtMTrend functions 75
PtMultiText functions 76
PtPanelGroup functions 77
PtProgress functions 77
PtTerminal functions 78
PtText functions 79
PtTree functions 79
PtTrend functions 81
PtTty functions 81
PtWindow functions 82
realizing 1589
determining if 1732
region ID, getting 1740
reparenting 1611
rerealizing 1613
resources
argument lists 1252, 1253, 1621
getting 1467, 1470
setting 1627, 1629
size, minimum 1733
size, preferred 1738
Index

styles
  adding 1159
  copying 1389
  creating 1357
  current 1476, 1637
  finding 1428
  getting 1473
  setting 1623, 1631, 1635
system information 1585
top-level, finding 1545
unrealizing 1699
updates
  flux count, containers 1343, 1347
  forcing 1446
  holding off 1510
  permitting 1593, 1701
  synchronizing a widget 1682
window manager
  consoles, switching 1327
  forwarding events 1451, 1453
windows
  actions, data structure 1054
  blocking 1261, 1530
  blocking all 171, 1259
  cursor, setting 1259, 1261
  disjoint 1433
  frame size, determining 1756
  unblocking 1697
WmConfig_t 1004
work procedures
  adding 1209
  prototype 1758
  removing 1238

X

XOR color
  bitmaps 759
  fill 710
  stroke 751
text 759