QuickDraw Reference

(Legacy)

Developer

Contents

QuickDraw Reference (Legacy)

Framework	ApplicationServices/ApplicationServices.h
Companion guide	Quartz Programming Guide for QuickDraw Developers
Declared in	IOMacOSTypes.h IONDRVLibraries.h QDOffscreen.h QDPictToCGContext.h Quickdraw.h QuickdrawAPI.h QuickdrawTypes.h

Important: This document may not represent best practices for current development. Links to downloads and other resources may no longer be valid.

Overview

QuickDraw is the legacy 2D drawing engine for Macintosh computers. QuickDraw provides routines for drawing, manipulating, and displaying graphic objects such as lines, arcs, rectangles, ovals, regions, and bitmap images. Carbon supports most of the classic QuickDraw programming interface.

Note: QuickDraw has been deprecated for deployment targets Mac OS X version 10.4 and later. The replacement API is Quartz 2D. Because of the fundamental differences in the imaging models and design goals between QuickDraw and Quartz, there is no direct correspondence between QuickDraw and Quartz concepts and interfaces. For certain purposes, some QuickDraw functions may still be needed during a transition period; nevertheless, most of them have been deprecated to express the overriding goal of eliminating the use of QuickDraw in the future.

Functions by Task

Drawing QuickDraw Pictures in a Quartz Context

`QDPictCreateWithProvider` (page 461) Available in OS X v10.1 through OS X v10.6

Creates a QDPict picture, using QuickDraw picture data supplied with a Quartz data provider.

`QDPictCreateWithURL` (page 462) Available in OS X v10.1 through OS X v10.6

Creates a QDPict picture, using QuickDraw picture data specified with a Core Foundation URL.

`QDPictDrawToCGContext` (page 463) Available in OS X v10.1 through OS X v10.6

Draws a QuickDraw picture in a Quartz context.

`QDPictGetBounds` (page 464) Available in OS X v10.1 through OS X v10.6

Returns the intended location and size of a QDPict picture.

`QDPictGetResolution` (page 465) Available in OS X v10.1 through OS X v10.6

Returns the horizontal and vertical resolution of a QDPict picture.

`QDPictRelease` (page 465) Available in OS X v10.1 through OS X v10.6

Releases a QDPict picture.

`QDPictRetain` (page 466) Available in OS X v10.1 through OS X v10.6

Retains a QDPict picture.

Using Quartz 2D to Draw in a Graphics Port

`QDBeginCGContext` (page 59)

Returns a Quartz 2D drawing environment associated with a graphics port.

`QDEndCGContext` (page 61)

Terminates a Quartz 2D drawing environment associated with a graphics port.

`ClipCGContextToRegion` (page 158) Available in OS X v10.0 through OS X v10.6

Sets the clipping path in a Quartz 2D graphics context, using a clipping region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`CreateCGContextForPort` (page 175) Available in OS X v10.0 through OS X v10.6

Creates a Quartz 2D drawing environment associated with a graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SyncCGContextOriginWithPort` (page 447) Available in OS X v10.0 through OS X v10.6

Synchronizes the origin in a Quartz context with the lower-left corner of the associated graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Other Quartz-Related Functions in QuickDraw

`CreateNewPortForCGDisplayID` (page 177) Available in OS X v10.0 through OS X v10.6

Creates a graphics port associated with a display. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LockPortBits` (page 316) Available in OS X v10.0 through OS X v10.6

Acquires an exclusive lock on the back buffer for a Carbon window. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDFlushPortBuffer` (page 378) Available in OS X v10.0 through OS X v10.6

Calls the Quartz compositor to flush all new drawing in a Carbon window to the display. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UnlockPortBits` (page 453) Available in OS X v10.0 through OS X v10.6

Releases a previously acquired lock on the back buffer for a Carbon window. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDGetCGDirectDisplayID` (page 480) Available in OS X v10.3 through OS X v10.6

Returns the Quartz display ID that corresponds to a QuickDraw graphics device.

Calculating Black-and-White Fills

`CalcMask` (page 157) Available in OS X v10.0 through OS X v10.6

Determines where filling will not occur when filling from the outside of a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SeedFill` (page 399) Available in OS X v10.0 through OS X v10.6

Determines how far filling will extend from a seeding point. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Calculating Color Fills

`CalcCMask` (page 155) Available in OS X v10.0 through OS X v10.6

Determines where filling will not occur when filling from the outside of a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SeedCFill` (page 398) Available in OS X v10.0 through OS X v10.6

Determines how far filling will extend to pixels matching the color of a particular pixel. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing Black-and-White Cursors

`GetCursor` (page 232) Available in OS X v10.0 through OS X v10.6

Loads a cursor resource into memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetCursor` (page 405) Available in OS X v10.0 through OS X v10.6

Sets the current cursor. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing Color Cursors

`AllocCursor` (page 149) Available in OS X v10.0 through OS X v10.6

Reallocates cursor memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeCCursor` (page 185) Available in OS X v10.0 through OS X v10.6

Disposes of all structures allocated by the `GetCCursor` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetCCursor` (page 227) Available in OS X v10.0 through OS X v10.6

Loads a color cursor resource into memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetCCursor` (page 401) Available in OS X v10.0 through OS X v10.6

Specifies a color cursor for display on the screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing the Background Bit Pattern

`BackPat` (page 152) Available in OS X v10.0 through OS X v10.6

Changes the bit pattern used as the background pattern by the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing the Background Pixel Pattern

`BackPixPat` (page 153) Available in OS X v10.0 through OS X v10.6

Assigns a pixel pattern as the background pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Compressing and Decompressing Data

`PackBits` (page 352) Available in OS X v10.0 through OS X v10.6

Compresses a data buffer stored in RAM. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UnpackBits` (page 453) Available in OS X v10.0 through OS X v10.6

Decompresses a data buffer containing data compressed by `PackBits`. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Converting Between Angle and Slope Values

`AngleFromSlope` (page 150) Available in OS X v10.0 through OS X v10.6

Converts a slope value to an angle value.

`SlopeFromAngle` (page 434) Available in OS X v10.0 through OS X v10.6

Converts an angle value to a slope value.

Copying Images

`CopyBits` (page 165) Available in OS X v10.0 through OS X v10.6

Copies a portion of a bitmap or a pixel map from one graphics port or offscreen graphics world into another graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`CopyDeepMask` (page 168) Available in OS X v10.0 through OS X v10.6

Uses a mask when copying bitmaps or pixel maps between graphics ports (or from an offscreen graphics world into a graphics port). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`CopyMask` (page 170) Available in OS X v10.0 through OS X v10.6

Copies a bit or pixel image from one graphics port or offscreen graphics world into another graphics port only where the bits in a mask are set to 1. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating, Altering, and Disposing of Offscreen Graphics Worlds

`DisposeGWorld` (page 187) Available in OS X v10.0 through OS X v10.6

Disposes of all the memory allocated for an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeScreenBuffer` (page 191) Available in OS X v10.0 through OS X v10.6

Disposes an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewGWorld` (page 328) Available in OS X v10.0 through OS X v10.6

Creates an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewScreenBuffer` (page 337) Available in OS X v10.0 through OS X v10.6

Creates an offscreen `PixMap` structure and allocates memory for the base address of its pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewTempScreenBuffer` (page 338) Available in OS X v10.0 through OS X v10.6

Creates an offscreen `PixMap` structure and allocate temporary memory for the base address of its pixel image applications generally don't need to use `NewTempScreenBuffer`. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UpdateGWorld` (page 454) Available in OS X v10.0 through OS X v10.6

Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Disposing of Color Tables

`DisposeCTable` (page 186) Available in OS X v10.0 through OS X v10.6

Disposes a `ColorTable` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetCTable` (page 230) Available in OS X v10.0 through OS X v10.6

Obtains a color table stored in a `'clut'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Disposing of Pictures

`ClosePicture` (page 161) Available in OS X v10.0 through OS X v10.6

Completes the collection of drawing commands and picture comments that define your picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`KillPicture` (page 295) Available in OS X v10.0 through OS X v10.6

Releases the memory occupied by a picture not stored in a `'PICT'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpenCPicture` (page 345) Available in OS X v10.0 through OS X v10.6

Begins defining a picture in extended version 2 format. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpenPicture` (page 348) Available in OS X v10.0 through OS X v10.6

Creates a picture which allows you to specify resolutions for your pictures. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PicComment` (page 365) Available in OS X v10.0 through OS X v10.6

Inserts a picture comment into a picture that you are defining or into your printing code. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Disposing of Pixel Patterns

`CopyPixPat` (page 173) Available in OS X v10.0 through OS X v10.6

Copies the contents of one pixel pattern to another. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposePixPat` (page 189) Available in OS X v10.0 through OS X v10.6

Releases the storage allocated to a pixel pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPixPat` (page 251) Available in OS X v10.0 through OS X v10.6

Obtains a pixel pattern (`'ppat'`) resource stored in a resource file. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MakeRGBPat` (page 318) Available in OS X v10.0 through OS X v10.6

Creates the appearance of otherwise unavailable colors on indexed devices. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`NewPixPat` (page 334) Available in OS X v10.0 through OS X v10.6

Creates a new pixel pattern. Generally, however, your application should create a pixel pattern in a `'ppat'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

Creating and Managing Polygons

`ClosePoly` (page 162) Available in OS X v10.0 through OS X v10.6

Completes the collection of lines that defines a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`KillPoly` (page 296) Available in OS X v10.0 through OS X v10.6

Releases the memory occupied by a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`OffsetPoly` (page 341) Available in OS X v10.0 through OS X v10.6

Moves a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`OpenPoly` (page 349) Available in OS X v10.0 through OS X v10.6

Begins defining a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

Creating and Managing Rectangles

`EmptyRect` (page 195) Available in OS X v10.0 through OS X v10.6

Determines whether a rectangle is an empty rectangle.

`EqualRect` (page 197) Available in OS X v10.0 through OS X v10.6

Determines whether two rectangles are equal.

`InsetRect` (page 281) Available in OS X v10.0 through OS X v10.6

Shrinks or expands a rectangle.

`OffsetRect` (page 342) Available in OS X v10.0 through OS X v10.6

Moves a rectangle.

`Pt2Rect` (page 371) Available in OS X v10.0 through OS X v10.6

Determines the smallest rectangle that encloses two given points.

`PtInRect` (page 372) Available in OS X v10.0 through OS X v10.6

Determines whether a pixel below is enclosed in a rectangle.

`PtToAngle` (page 374) Available in OS X v10.0 through OS X v10.6

Calculates an angle between a vertical line pointing straight up from the center of a rectangle and a line from the center to a given point.

`SectRect` (page 395) Available in OS X v10.0 through OS X v10.6

Determines whether two rectangles intersect.

`SetRect` (page 426) Available in OS X v10.0 through OS X v10.6

Assigns coordinates to a rectangle.

`UnionRect` (page 449) Available in OS X v10.0 through OS X v10.6

Calculates the smallest rectangle that encloses two rectangles.

Creating and Managing Regions

`CloseRgn` (page 162) Available in OS X v10.0 through OS X v10.6

Organizes a collection of lines and shapes into a region definition. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`CopyRgn` (page 174) Available in OS X v10.0 through OS X v10.6

Makes a copy of a region.

`DiffRgn` (page 184) Available in OS X v10.0 through OS X v10.6

Subtracts one region from another.

`DisposeRgn` (page 190) Available in OS X v10.0 through OS X v10.6

Releases the memory occupied by a region.

`EmptyRgn` (page 195) Available in OS X v10.0 through OS X v10.6

Determines whether a region is empty.

`EqualRgn` (page 198) Available in OS X v10.0 through OS X v10.6

Determines whether two regions have identical sizes, shapes, and locations.

`InsetRgn` (page 283) Available in OS X v10.0 through OS X v10.6

Shrinks or expands a region.

`NewRgn` (page 336) Available in OS X v10.0 through OS X v10.6

Begins creating a new region.

`OffsetRgn` (page 343) Available in OS X v10.0 through OS X v10.6

Moves a region.

`OpenRgn` (page 350) Available in OS X v10.0 through OS X v10.6

Begins defining a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`PtInRgn` (page 373) Available in OS X v10.0 through OS X v10.6

Determines whether a pixel is within a region.

`RectInRgn` (page 383) Available in OS X v10.0 through OS X v10.6

Determines whether a rectangle intersects a region.

`RectRgn` (page 384) Available in OS X v10.0 through OS X v10.6

Changes the structure of an existing region to that of a rectangle.

`SectRgn` (page 396) Available in OS X v10.0 through OS X v10.6

Calculates the intersection of two regions.

`SetEmptyRgn` (page 407) Available in OS X v10.0 through OS X v10.6

Sets an existing region to be empty.

`SetRectRgn` (page 428) Available in OS X v10.0 through OS X v10.6

Changes the structure of an existing region to that of a rectangle.

`UnionRgn` (page 450) Available in OS X v10.0 through OS X v10.6

Calculates the union of two regions.

`XorRgn` (page 457) Available in OS X v10.0 through OS X v10.6

Calculates the difference between the union and the intersection of two regions.

Creating, Setting, and Disposing of GDevice Records

`DisposeGDevice` (page 187) Available in OS X v10.0 through OS X v10.6

Disposes of a `GDevice` structure, releases the space allocated for it, and disposes of all the data structures allocated for it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`InitGDevice` (page 280) Available in OS X v10.0 through OS X v10.6

Initializes a `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`NewGDevice` (page 327) Available in OS X v10.0 through OS X v10.6

Creates a new `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`SetDeviceAttribute` (page 406) Available in OS X v10.0 through OS X v10.6

Sets the attribute bits of a `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

SetGDevice (page 409) Available in OS X v10.0 through OS X v10.6

Sets a GDevice structure as the current device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating, Setting, and Disposing of Pixel Maps

CopyPixMap (page 172) Available in OS X v10.0 through OS X v10.6

Duplicates a PixMap structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposePixMap (page 188) Available in OS X v10.0 through OS X v10.6

Disposes a PixMap structure and its color table. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewPixMap (page 333) Available in OS X v10.0 through OS X v10.6

Creates a new, initialized PixMap structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPix (page 423) Available in OS X v10.0 through OS X v10.6

Sets the pixel map for the current color graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Customizing Color QuickDraw Operations

SetStdCProcs (page 429) Available in OS X v10.0 through OS X v10.6

Obtains a CQDProcs structure with fields that point to QuickDraw's standard low-level functions, which you can modify to change QuickDraw's standard low-level behavior. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Customizing QuickDraw Operations

SetStdProcs (page 430) Available in OS X v10.0 through OS X v10.6

Obtains a QDProcs structure with fields that point to basic QuickDraw's standard low-level functions, which you can modify to point to your own functions. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

StdArc (page 435) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing an arc or a wedge. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdBits` (page 436) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for transferring bits and pixels. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdComment` (page 437) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for processing a picture comment. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdGetPic` (page 438) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for retrieving information from the definition of a picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdLine` (page 439) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing a line. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdOval` (page 440) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdPoly` (page 441) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdPutPic` (page 442) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for saving information as the definition of a picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdRect` (page 442) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdRgn` (page 443) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdRRect` (page 444) Available in OS X v10.0 through OS X v10.6

QuickDraw's standard low-level function for drawing a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Determining Current Colors and Best Intermediate Colors

`GetBackColor` (page 226) Available in OS X v10.0 through OS X v10.6

Obtains the background color of the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetCPixel` (page 229) Available in OS X v10.0 through OS X v10.6

Determines the color of an individual pixel specified in the `h` and `v` parameters. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetForeColor` (page 234) Available in OS X v10.0 through OS X v10.6

Obtains the color of the foreground color for the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Determining the Characteristics of a Video Device

`DeviceLoop` (page 182) Available in OS X v10.0 through OS X v10.6

Draws images that are optimized for every screen they cross. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ScreenRes` (page 393) Available in OS X v10.0 through OS X v10.6

Determines the resolution of the main device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`TestDeviceAttribute` (page 448) Available in OS X v10.0 through OS X v10.6

Determines whether the flag bit for an attribute has been set in the `gdFlags` field of a `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Determining Whether QuickDraw Has Finished Drawing

`QDDone` (page 376) Available in OS X v10.0 through OS X v10.6

Determines whether QuickDraw has completed drawing in a given graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Arcs and Wedges

`EraseArc` (page 199) Available in OS X v10.0 through OS X v10.6

Erases a wedge. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillArc` (page 205) Available in OS X v10.0 through OS X v10.6

Fills a wedge with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameArc` (page 218) Available in OS X v10.0 through OS X v10.6

Draws an arc of the oval that fits inside a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertArc` (page 284) Available in OS X v10.0 through OS X v10.6

Inverts the pixels of a wedge. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintArc` (page 354) Available in OS X v10.0 through OS X v10.6

Paints a wedge of the oval that fits inside a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Lines

`Line` (page 297) Available in OS X v10.0 through OS X v10.6

Draws a line a specified distance from the graphics pen's current location in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LineTo` (page 298) Available in OS X v10.0 through OS X v10.6

Draws a line from the graphics pen's current location to a new location. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`Move` (page 324) Available in OS X v10.0 through OS X v10.6

Moves the graphics pen a particular distance. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MoveTo` (page 325) Available in OS X v10.0 through OS X v10.6

Moves the graphics pen to a particular location in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Ovals

`EraseOval` (page 200) Available in OS X v10.0 through OS X v10.6

Erases an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillOval` (page 212) Available in OS X v10.0 through OS X v10.6

Fills an oval with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameOval` (page 220) Available in OS X v10.0 through OS X v10.6

Draws an outline inside an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertOval` (page 286) Available in OS X v10.0 through OS X v10.6

Inverts the pixels enclosed by an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintOval` (page 355) Available in OS X v10.0 through OS X v10.6

Paints an oval with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Pictures

`DrawPicture` (page 192) Available in OS X v10.0 through OS X v10.6

Draws a picture on any type of output device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPicture` (page 246) Available in OS X v10.0 through OS X v10.6

Obtains a handle to a picture stored in a `'PICT'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Polygons

`ErasePoly` (page 201) Available in OS X v10.0 through OS X v10.6

Erases a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillPoly` (page 213) Available in OS X v10.0 through OS X v10.6

Fills a polygon with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FramePoly` (page 221) Available in OS X v10.0 through OS X v10.6

Draws the outline of a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertPoly` (page 287) Available in OS X v10.0 through OS X v10.6

Inverts the pixels enclosed by a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintPoly` (page 356) Available in OS X v10.0 through OS X v10.6

Paints a polygon with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

Drawing Rectangles

`EraseRect` (page 202) Available in OS X v10.0 through OS X v10.6

Erases a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`FillRect` (page 214) Available in OS X v10.0 through OS X v10.6

Fills a rectangle with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`FrameRect` (page 222) Available in OS X v10.0 through OS X v10.6

Draws an outline inside a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`InvertRect` (page 288) Available in OS X v10.0 through OS X v10.6

Inverts the pixels enclosed by a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`PaintRect` (page 357) Available in OS X v10.0 through OS X v10.6

Paints a rectangle with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

Drawing Regions

`EraseRgn` (page 203) Available in OS X v10.0 through OS X v10.6

Erases a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`FillRgn` (page 215) Available in OS X v10.0 through OS X v10.6

Fills a region with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`FrameRgn` (page 223) Available in OS X v10.0 through OS X v10.6

Draws an outline inside a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`InvertRgn` (page 289) Available in OS X v10.0 through OS X v10.6

Inverts the pixels enclosed by a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintRgn` (page 358) Available in OS X v10.0 through OS X v10.6

Paints a region with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Rounded Rectangles

`EraseRoundRect` (page 204) Available in OS X v10.0 through OS X v10.6

Erases a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillRoundRect` (page 216) Available in OS X v10.0 through OS X v10.6

Fills a rounded rectangle with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameRoundRect` (page 224) Available in OS X v10.0 through OS X v10.6

Draws an outline inside a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertRoundRect` (page 291) Available in OS X v10.0 through OS X v10.6

Inverts the pixels enclosed by a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintRoundRect` (page 359) Available in OS X v10.0 through OS X v10.6

Paints a rounded rectangle with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing With Color QuickDraw Colors

`FillCArc` (page 206) Available in OS X v10.0 through OS X v10.6

Fills a wedge with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCOval` (page 207) Available in OS X v10.0 through OS X v10.6

Fills an oval with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCPoly` (page 208) Available in OS X v10.0 through OS X v10.6

Fills a polygon with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCRect` (page 209) Available in OS X v10.0 through OS X v10.6

Fills a rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCRgn` (page 210) Available in OS X v10.0 through OS X v10.6

Fills a region with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCRoundRect` (page 211) Available in OS X v10.0 through OS X v10.6

Fills a rounded rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`HiliteColor` (page 278) Available in OS X v10.0 through OS X v10.6

Changes the highlight color for the current color graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpColor` (page 344) Available in OS X v10.0 through OS X v10.6

Sets the maximum color values for the `addPin` and `subPin` arithmetic transfer modes and the weight color for the `blend` arithmetic transfer mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RGBBackColor` (page 388) Available in OS X v10.0 through OS X v10.6

Changes the background color. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RGBForeColor` (page 389) Available in OS X v10.0 through OS X v10.6

Changes the color of the "ink" used for framing and painting. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetCPixel` (page 404) Available in OS X v10.0 through OS X v10.6

Sets the color of an individual pixel to the color that most closely matches the RGB color that you specify in the `cPix` parameter. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing With the Eight-Color System

`BackColor` (page 151) Available in OS X v10.0 through OS X v10.6

Changes a basic graphics port's background color. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ColorBit` (page 165) Available in OS X v10.0 through OS X v10.6

Sets the foreground color for all printing in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ForeColor` (page 217) Available in OS X v10.0 through OS X v10.6

Changes the color of the "ink" used for framing, painting, and filling on computers that support only basic QuickDraw. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Getting Pattern Resources

`GetIndPattern` (page 239) Available in OS X v10.0 through OS X v10.6

Obtains a pattern stored in a pattern list (`'PAT#'`) resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPattern` (page 243) Available in OS X v10.0 through OS X v10.6

Obtains a pattern (`'PAT'`) resource stored in a resource file. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Getting the Available Graphics Devices

`GetDeviceList` (page 233) Available in OS X v10.0 through OS X v10.6

Obtains a handle to the first `GDevice` structure in the device list. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetGDevice` (page 235) Available in OS X v10.0 through OS X v10.6

Obtains a handle to the `GDevice` structure for the current device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetMainDevice` (page 240) Available in OS X v10.0 through OS X v10.6

Obtains a handle to the `GDevice` structure for the main screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetMaxDevice` (page 241) Available in OS X v10.0 through OS X v10.6

Obtains a handle to the `GDevice` structure for the video device with the greatest pixel depth. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetNextDevice` (page 242) Available in OS X v10.0 through OS X v10.6

Returns a handle to the next `GDevice` structure in the device list. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Hiding and Showing Cursors

`HideCursor` (page 276) Available in OS X v10.0 through OS X v10.6

Hides the cursor if it is visible on the screen.

`InitCursor` (page 279) Available in OS X v10.0 through OS X v10.6

Sets the cursor to the standard arrow and makes the cursor visible.

`ObscureCursor` (page 340) Available in OS X v10.0 through OS X v10.6

Hides the cursor until the next time the user moves the mouse.

`ShieldCursor` (page 431) Available in OS X v10.0 through OS X v10.6

Hides the cursor in a rectangle.

`ShowCursor` (page 432) Available in OS X v10.0 through OS X v10.6

Displays a cursor hidden by the `HideCursor` or `ShieldCursor` functions.

Managing a Color Graphics Pen

`PenPixPat` (page 363) Available in OS X v10.0 through OS X v10.6

Sets the pixel pattern used by the graphics pen in the current color graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing an Offscreen Graphics World's Pixel Image

`AllowPurgePixels` (page 149) Available in OS X v10.0 through OS X v10.6

Makes the base address for an offscreen pixel image purgeable. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetGWorldPixMap` (page 238) Available in OS X v10.0 through OS X v10.6

Obtains the pixel map created for an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPixBaseAddr` (page 247) Available in OS X v10.0 through OS X v10.6

Obtains a pointer to an offscreen pixel map.

`GetPixelsState` (page 250) Available in OS X v10.0 through OS X v10.6

Saves the current information about the memory allocated for an offscreen pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LockPixels` (page 314) Available in OS X v10.0 through OS X v10.6

Prevents the base address for an offscreen pixel image from being moved while you draw into or copy from its pixel map. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`NoPurgePixels` (page 339) Available in OS X v10.0 through OS X v10.6

Prevents the Memory Manager from purging the base address for an offscreen pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`PixMap32Bit` (page 367) Available in OS X v10.0 through OS X v10.6

Determines whether a pixel map requires 32-bit addressing mode for access to its pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`SetPixelsState` (page 414) Available in OS X v10.0 through OS X v10.6

Restores an offscreen pixel image to the state that you saved with the `GetPixelsState` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`UnlockPixels` (page 451) Available in OS X v10.0 through OS X v10.6

Allows the Memory Manager to move the base address for the offscreen pixel map that you specify in the `pm` parameter. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

Managing Bitmaps, Port Rectangles, and Clipping Regions

`BitMapToRegion` (page 154) Available in OS X v10.0 through OS X v10.6

Converts a bitmap or pixel map to a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`ClipRect` (page 159) Available in OS X v10.0 through OS X v10.6

Changes the clipping region of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`GetClip` (page 228) Available in OS X v10.0 through OS X v10.6

Saves the clipping region of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`MovePortTo` (page 325) Available in OS X v10.0 through OS X v10.6

Changes the position of the port rectangle of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`PortSize` (page 370) Available in OS X v10.0 through OS X v10.6

Changes the size of the port rectangle of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers* .)

`ScrollRect` (page 394) Available in OS X v10.0 through OS X v10.6

Scroll the pixels of a specified portion of a basic graphics port's bitmap (or a color graphics port's pixel map). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetClip` (page 402) Available in OS X v10.0 through OS X v10.6

Changes the clipping region of the current graphics port (basic or color) to a region you specify. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetOrigin` (page 411) Available in OS X v10.0 through OS X v10.6

Changes the coordinates of the window origin of the port rectangle of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPortBits` (page 417) Available in OS X v10.0 through OS X v10.6

Sets the bitmap for the current basic graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing Color Tables

`GetCTSeed` (page 231) Available in OS X v10.0 through OS X v10.6

Obtains a unique seed value for a color table created by your application. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ProtectEntry` (page 371) Available in OS X v10.0 through OS X v10.6

Adds protection to or removes protection from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ReserveEntry` (page 385) Available in OS X v10.0 through OS X v10.6

Reserves or removes reservation from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RestoreEntries` (page 386) Available in OS X v10.0 through OS X v10.6

Restores a selection of color table entries. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SaveEntries` (page 390) Available in OS X v10.0 through OS X v10.6

Saves a selection of color table entries. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetEntries` (page 408) Available in OS X v10.0 through OS X v10.6

Sets a group of color table entries for the current `GDevice` data structure. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing Colors

`Color2Index` (page 164) Available in OS X v10.0 through OS X v10.6

Obtains the index of the best available approximation for a given color in the color table of the current `GDevice` data structure. This function is used only by system software. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetSubTable` (page 273) Available in OS X v10.0 through OS X v10.6

Searches one color table for the best matches to colors in another color table. Your application should not need to call this function; it is used by system software only. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`Index2Color` (page 279) Available in OS X v10.0 through OS X v10.6

Obtains the `RGBColor` data structure corresponding to an index value in the color table of the current `GDevice` data structure. Your application should not need to call this function; it is used by system software only. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertColor` (page 285) Available in OS X v10.0 through OS X v10.6

Finds the complement of an `RGBColor` data structure. This function is used only by system software. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MakeITable` (page 317) Available in OS X v10.0 through OS X v10.6

Generates an inverse table for a color table. Your application should not need to call this function; it is used by system software only. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RealColor` (page 382) Available in OS X v10.0 through OS X v10.6

Determines whether a given `RGBColor` data structure exists in the current device's color table. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing the Graphics Pen

`GetPen` (page 244) Available in OS X v10.0 through OS X v10.6

Determines the location of the graphics pen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPenState` (page 245) Available in OS X v10.0 through OS X v10.6

Determines the graphics pen's location, size, pattern, and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`HidePen` (page 277) Available in OS X v10.0 through OS X v10.6

Makes the graphics pen invisible, so that pen drawing doesn't show on the screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenMode` (page 360) Available in OS X v10.0 through OS X v10.6

Sets the pattern mode of the graphics pen in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenNormal` (page 361) Available in OS X v10.0 through OS X v10.6

Sets the size, pattern, and pattern mode of the graphics pen in the current graphics port to their initial values. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenPat` (page 362) Available in OS X v10.0 through OS X v10.6

Sets the bit pattern to be used by the graphics pen in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenSize` (page 364) Available in OS X v10.0 through OS X v10.6

Sets the dimensions of the graphics pen in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPenState` (page 413) Available in OS X v10.0 through OS X v10.6

Restores the state of the graphics pen that was saved with the `GetPenState` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ShowPen` (page 433) Available in OS X v10.0 through OS X v10.6

Changes the ink of a graphics pen from invisible to visible, making pen drawing appear on the screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Manipulating Points in Graphics Ports

`AddPt` (page 147) Available in OS X v10.0 through OS X v10.6

Adds the coordinates of two points.

`DeltaPoint` (page 181) Available in OS X v10.0 through OS X v10.6

Subtracts the coordinates of one point from another. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`EqualPt` (page 197) Available in OS X v10.0 through OS X v10.6

Determines whether the coordinates of two given points are equal.

`GetPixel` (page 249) Available in OS X v10.0 through OS X v10.6

Determines whether the pixel associated with a point is black or white. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GlobalToLocal` (page 274) Available in OS X v10.0 through OS X v10.6

Converts the coordinates of a point from global coordinates to the local coordinates of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LocalToGlobal` (page 313) Available in OS X v10.0 through OS X v10.6

Converts a point's coordinates from the local coordinates of the current graphics port (basic or color) to global coordinates. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPt` (page 424) Available in OS X v10.0 through OS X v10.6

Assigns two coordinates to a point.

`SubPt` (page 446) Available in OS X v10.0 through OS X v10.6

Subtracts the coordinates of one point from another.

Obtaining a Pseudorandom Number

`Random` (page 381) Available in OS X v10.0 through OS X v10.6

Obtains a pseudorandom integer. (Deprecated. Use the Standard C Library `random(3)` function instead.)

Operations on Search and Complement Functions

`AddComp` (page 147) Available in OS X v10.0 through OS X v10.6

Adds a function to the head of the current device data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`AddSearch` (page 148) Available in OS X v10.0 through OS X v10.6

Adds a function to the head of the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DelComp` (page 180) Available in OS X v10.0 through OS X v10.6

Removes a custom complement function from the current `GDevice` data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DelSearch` (page 180) Available in OS X v10.0 through OS X v10.6

Removes a custom search function from the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetClientID` (page 402) Available in OS X v10.0 through OS X v10.6

Sets the `gdID` field in the current `GDevice` data structure to identify this client program to its search and complement functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Reporting Data Structure Changes to QuickDraw

`CTabChanged` (page 178) Available in OS X v10.0 through OS X v10.6

Signals QuickDraw that the content of a `ColorTable` structure has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GDeviceChanged` (page 225) Available in OS X v10.0 through OS X v10.6

Notifies QuickDraw that the content of a `GDevice` structure has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PixPatChanged` (page 368) Available in OS X v10.0 through OS X v10.6

Notifies QuickDraw that the content of a `PixPat` structure, including its `PixMap` structure or the image in its `patData` field, has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PortChanged` (page 369) Available in OS X v10.0 through OS X v10.6

Notifies QuickDraw that the content of a `GrafPort` structure or `CGrafPort` structure, including any of the data structures specified by handles within the structure, has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Retrieving Color QuickDraw Result Codes

`QDError` (page 377) Available in OS X v10.0 through OS X v10.6

Obtains a result code from the last applicable QuickDraw function that you called. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Saving and Restoring Graphics Ports

`GetPort` (page 253) Available in OS X v10.0 through OS X v10.6

Saves the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPort` (page 415) Available in OS X v10.0 through OS X v10.6

Changes the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

`GetGWorld` (page 236) Available in OS X v10.0 through OS X v10.6

Saves the current graphics port (basic, color, or offscreen) and the current `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetGWorldDevice` (page 237) Available in OS X v10.0 through OS X v10.6

Obtains a handle to the `GDevice` structure associated with an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetGWorld` (page 410) Available in OS X v10.0 through OS X v10.6

Changes the current graphics port (basic, color, or offscreen). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Scaling and Mapping Points, Rectangles, Polygons, and Regions

`MapPoly` (page 319) Available in OS X v10.0 through OS X v10.6

Maps and scales a polygon within one rectangle to another rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MapPt` (page 320) Available in OS X v10.0 through OS X v10.6

Maps a point in one rectangle to an equivalent position in another rectangle.

`MapRect` (page 321) Available in OS X v10.0 through OS X v10.6

Maps and scales a rectangle within one rectangle to another rectangle.

`MapRgn` (page 322) Available in OS X v10.0 through OS X v10.6

Maps and scales a region within one rectangle to another rectangle.

`ScalePt` (page 391) Available in OS X v10.0 through OS X v10.6

Scales a height and width according to the proportions of two rectangles.

Miscellaneous

`IsValidRgnHandle` (page 482) Available in OS X v10.4 through OS X v10.6

`QDGlobalToLocalPoint` (page 473) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDGlobalToLocalRect` (page 473) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDGlobalToLocalRegion` (page 474) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDIsNamedPixMapCursorRegistered` (page 474) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDLocalToGlobalPoint` (page 475) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDLocalToGlobalRect` (page 475) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDLocalToGlobalRegion` (page 476) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDRegisterNamedPixMapCursor` (page 476) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDSetCursorScale` (page 477) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDSetNamedPixMapCursor` (page 477) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDSwapPortTextFlags` (page 478) Available in OS X v10.2 through OS X v10.6

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSwapTextFlags (page 478) Available in OS X v10.2 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SwapPortPolySaveHandle (page 479) Available in OS X v10.2 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SwapPortRegionSaveHandle (page 479) Available in OS X v10.2 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortClipRegionEmpty (page 458) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortVisibleRegionEmpty (page 459) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDAddRectToDirtyRegion (page 459) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDAddRegionToDirtyRegion (page 460) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDDisposeRegionBits (page 460) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetPatternOrigin (page 461) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDRestoreRegionBits (page 467) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSaveRegionBits (page 467) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSetPatternOrigin (page 468) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSwapPort (page 468) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

RgnToHandle (page 469) Available in OS X v10.1 through OS X v10.6

SectRegionWithPortClipRegion (page 469) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SectRegionWithPortVisibleRegion (page 470) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextFace (page 470) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextFont (page 471) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextMode (page 471) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextSize (page 472) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SwapPortPicSaveHandle (page 472) Available in OS X v10.1 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CloseCursorComponent (page 160) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CreateNewPort (page 176) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CursorComponentChanged (page 179) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CursorComponentSetData (page 179) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

deltapoint (page 182) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use DeltaPoint (page 181) instead.)

DisposeDeviceLoopDrawingUPP (page 186) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposePort (page 190) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetMaskTable (page 241) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPixBounds (page 248) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPixDepth (page 249) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPixRowBytes (page 252) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBackColor (page 254) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBackPixPat (page 254) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBitMapForCopyBits (page 255) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBounds (page 256) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortChExtra (page 256) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortClipRegion (page 257) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortCustomXFerProc (page 257) Available in OS X v10.0 through OS X v10.6

GetPortFillPixPat (page 258) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortForeColor (page 258) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortFracHPenLocation (page 259) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortGrafProcs (page 259) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortHiliteColor (page 260) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortOpColor (page 261) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenLocation (page 261) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenMode (page 262) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenPixPat (page 262) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenSize (page 263) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenVisibility (page 264) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPixMap (page 264) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortSpExtra (page 265) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextFace (page 265) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextFont (page 266) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextMode (page 266) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextSize (page 267) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortVisibleRegion (page 267) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsArrow (page 268) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsBlack (page 269) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsDarkGray (page 269) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsGray (page 270) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsLightGray (page 270) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsRandomSeed (page 271) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsScreenBits (page 271) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsThePort (page 272) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsWhite (page 272) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetRegionBounds (page 273) Available in OS X v10.0 through OS X v10.6

GrafDevice (page 275) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

HandleToRgn (page 276) Available in OS X v10.0 through OS X v10.6

InvokeDeviceLoopDrawingUPP (page 292) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortColor (page 292) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortOffscreen (page 293) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortPictureBeingDefined (page 293) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortPolyBeingDefined (page 294) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortRegionBeingDefined (page 294) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsRegionRectangular (page 295) Available in OS X v10.0 through OS X v10.6

IsValidPort (page 295) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetCursorNew (page 299) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetDeviceList (page 300) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetFractEnable (page 300) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetHiliteMode (page 301) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetHiliteRGB (page 301) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetLastFOND (page 302) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetLastSPExtra (page 302) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetMainDevice (page 303) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetQDColors (page 303) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetScrHRes (page 304) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetScrVRes (page 304) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetTheGDevice (page 304) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetWidthListHand (page 305) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetWidthPtr (page 305) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetWidthTabHandle (page 306) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetCursorNew (page 306) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetDeviceList (page 307) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetFractEnable (page 307) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetHiliteMode (page 308) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetHiliteRGB (page 308) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetLastFOND (page 309) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetLastSPExtra (page 309) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetMainDevice (page 310) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetQDColors (page 310) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetScrHRes (page 310) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetScrVRes (page 311) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetTheGDevice (page 311) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetWidthListHand (page 312) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetWidthPtr (page 312) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetWidthTabHandle (page 313) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewDeviceLoopDrawingUPP (page 326) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewGWorldFromPtr (page 333) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

OffscreenVersion (page 341) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

OpenCursorComponent (page 347) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDDisplayWaitCursor (page 375) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetDirtyRegion (page 379) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDIsPortBufferDirty (page 379) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDIsPortBuffered (page 380) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDRegionToRects (page 380) Available in OS X v10.0 through OS X v10.6

QDSetDirtyRegion (page 381) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetCursorComponent (page 405) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortBackPixPat (page 416) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortBounds (page 417) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortClipRegion (page 418) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortCustomXFerProc (page 418) Available in OS X v10.0 through OS X v10.6

SetPortFillPixPat (page 419) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortFracHPenLocation (page 419) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortGrafProcs (page 420) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortOpColor (page 421) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPenMode (page 421) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPenPixPat (page 422) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPenSize (page 422) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortVisibleRegion (page 424) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetQDError (page 425) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetQDGlobalsArrow (page 426) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetQDGlobalsRandomSeed (page 426) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

StdOpcode (page 439) Available in OS X v10.0 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

StuffHex (page 445) Available in OS X v10.0 through OS X v10.6
Sets byte values into memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeColorComplementUPP (page 482) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeColorSearchUPP (page 483) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeDragGrayRgnUPP (page 483) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDArcUPP (page 484) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDBitsUPP (page 484) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDCommentUPP (page 485) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDGetPicUPP (page 485) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDJShieldCursorUPP (page 485) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDLineUPP (page 486) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDOpcodeUPP (page 486) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDOvalUPP (page 486) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDPolyUPP (page 487) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDPutPicUPP (page 487) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDRectUPP (page 488) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDRgnUPP (page 488) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDRRectUPP (page 488) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDStdGlyphsUPP (page 489) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDTextUPP (page 489) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDTxMeasUPP (page 490) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeRegionToRectsUPP (page 490) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeColorComplementUPP (page 490) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeColorSearchUPP (page 491) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeDragGrayRgnUPP (page 491) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDArcUPP (page 492) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDBitsUPP (page 492) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDCommentUPP (page 493) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDGetPicUPP (page 493) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDJShieldCursorUPP (page 493) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDLineUPP (page 494) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDOpcodeUPP (page 494) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDOvalUPP (page 495) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDPolyUPP (page 495) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDPutPicUPP (page 496) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDRectUPP (page 496) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDRgnUPP (page 496) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDRRectUPP (page 497) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDStdGlyphsUPP (page 497) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDTextUPP (page 498) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDTxMeasUPP (page 498) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeRegionToRectsUPP (page 499) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewColorComplementUPP (page 499) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewColorSearchUPP (page 500) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewDragGrayRgnUPP (page 500) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDArcUPP (page 501) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDBitsUPP (page 501) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDCommentUPP (page 502) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDGetPicUPP (page 502) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDJShieldCursorUPP (page 502) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDLineUPP (page 503) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDOpcodeUPP (page 503) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDOvalUPP (page 504) Deprecated in OS X v10.4

(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDPolyUPP (page 504) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDPutPicUPP (page 505) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDRectUPP (page 505) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDRgnUPP (page 505) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDRRectUPP (page 506) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDStdGlyphsUPP (page 506) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDTextUPP (page 507) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDTxMeasUPP (page 507) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewRegionToRectsUPP (page 507) Deprecated in OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetCursorData (page 480) Available in OS X v10.3 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetPictureBounds (page 481) Available in OS X v10.3 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDUnregisterNamedPixMapCursor (page 481) Available in OS X v10.3 through OS X v10.6
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Functions

QDBeginCGContext

Returns a Quartz 2D drawing environment associated with a graphics port.

```
OSStatus QDBeginCGContext (
   CGrafPtr inPort,
   CGContextRef *outContext
);
```


Parameters

`port`

A color graphics port in which to draw. Offscreen graphics worlds with pixel depths of 1, 2, 4, and 8 are not supported. When using Quartz 2D to draw in a offscreen graphics world, alpha information is always ignored.

`contextPtr`

A pointer to your storage for a Quartz context. Upon completion, `contextPtr` points to a context associated with the port. The context matches the port's pixel depth, width, and height. Otherwise the context is in a default state and does not necessarily match other port attributes such as foreground color, background color, or clip region.

You should not retain or release the context. When you are finished using the context, you should call `QDEndCGContext` (page 61).

Return Value

A result code. If `noErr`, the context was successfully initiated.

Discussion

Applications running in Mac OS X can use Quartz 2D to draw in a QuickDraw graphics port. When you call this function, you obtain a Quartz context that's associated with the specified port. To improve performance, contexts returned by this function are cached and reused during subsequent calls whenever possible.

Each block of Quartz 2D drawing code in your application should be surrounded by calls to this function and `QDEndCGContext` (page 61). Nested calls to this function for the same graphics port are not permitted—that is, for a given port you should not call this function more than once without an intervening call to `QDEndCGContext` (page 61).

While the Quartz context is in use, all Quickdraw imaging operations in the associated graphics port are disabled. This is done because the operations would fail during printing.

For information about how to use a Quartz context, see *Quartz 2D Programming Guide*.

Availability

Available in OS X v10.1 and later.

Not available to 64-bit applications.

Related Sample Code

ATSUICurveAccessDemo
ATSUIDirectAccessDemo
Carbon Porting Tutorial
CarbonQuartzDrawingWPrinting
CarbonSketch

Declared in
Quickdraw.h

QDEndCGContext

Terminates a Quartz 2D drawing environment associated with a graphics port.

```
OSStatus QDEndCGContext (
   CGrafPtr inPort,
   CGContextRef *inoutContext
);
```

Parameters
port

A graphics port specified in a preceding call to QDBeginCGContext (page 59).

contextPtr

A pointer to the context obtained in the preceding call to QDBeginCGContext (page 59) for the port. Upon completion, the storage pointed to by contextPtr is set to NULL.

Return Value
A result code. If noErr, the context is terminated.

Discussion
After you finish using Quartz 2D to draw in a graphics port, you should call this function to terminate the context. For more information, see QDBeginCGContext (page 59).

Before calling this function, you should do one of the following:

- Call CGContextSynchronize to mark the affected areas of the port for update.
- Call CGContextFlush to immediately update the destination device.

Availability
Available in OS X v10.1 and later.

Not available to 64-bit applications.

Related Sample Code
ATSUICurveAccessDemo
ATSUIDirectAccessDemo
Carbon Porting Tutorial
CarbonQuartzDrawingWPrinting
CarbonSketch

Declared in
`Quickdraw.h`

Callbacks

ColorComplementProcPtr

Defines a pointer to a color inversion callback function that overrides the Color Manager's color inversion method.

```
typedef Boolean (*ColorComplementProcPtr) (
   RGBColor * rgb
);
```

If you name your function `MyColorComplementProc`, you would declare it like this:

```
Boolean ColorComplementProcPtr (
   RGBColor * rgb
);
```

Parameters

`rgb`

A pointer to the `RGBColor` data structure. Change it to reflect the inverted value.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

ColorSearchProcPtr

Defines a pointer to a color search callback function that overrides the Color Manager's code for inverse table mapping.

```
typedef Boolean (*ColorSearchProcPtr) (
   RGBColor * rgb,
   long * position
);
```


If you name your function `MyColorSearchProc`, you would declare it like this:

```
Boolean ColorSearchProcPtr (
   RGBColor * rgb,
   long * position
);
```

Parameters

`rgb`

A pointer to the `RGBColor` data structure passed to your search function. Your function should set the `ColorSpec.value` field to the index corresponding to the color indicated here.

`position`

A pointer to the index of the best-mapping color your function finds.

Return Value

`True` if your function succeeds, `false` if your function cannot find a match.

Discussion

Your `MyColorSearchCallback` function should examine the `RGBColor` data structure passed to it by the Color Manager and return the index to the best-mapping color in the current `GDevice` data structure.

The Color Manager specifies the desired color in the `RGBColor` field of a `ColorSpec` data structure and passes it by a pointer on the stack. Your function should return the corresponding index in the `ColorSpec.value` field. If your function cannot handle the search, return `false` as the function value, and pass the `RGBColor` data structure back to the Color Manager in the `rgb` parameter.

The Color Manager calls each search function in the list until one returns the Boolean value `true`. If no search function installed in the linked list returns `true`, the Color Manager calls the default search function.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

CustomXFerProcPtr

```
typedef void (*CustomXFerProcPtr) (
   CustomXFerRecPtr info
);
```


If you name your function `MyCustomXFerProc`, you would declare it like this:

```
void CustomXFerProcPtr (
   CustomXFerRecPtr info
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawAPI.h`

DeviceLoopDrawingProcPtr

```
typedef void (*DeviceLoopDrawingProcPtr) (
   short depth,
   short deviceFlags,
   GDHandle targetDevice,
   long userData
);
```

If you name your function `MyDeviceLoopDrawingProc`, you would declare it like this:

```
void DeviceLoopDrawingProcPtr (
   short depth,
   short deviceFlags,
   GDHandle targetDevice,
   long userData
);
```

Parameters

`depth`

The pixel depth of the graphics device.

`deviceFlags`

Constants which represent bits that are set to 1 in the `gdFlags` field of the `GDevice` structure for the current device. See "Device Attribute Constants" (page 125) for a description of the values which you can receive in this parameter.

`targetDevice`

A handle to the `GDevice` (page 91) structure for the current device.

`userData`

A value that your application supplies to the `DeviceLoop` function, which in turn passes the value to your drawing function for whatever purpose you deem useful.

Discussion

For each video device that intersects a drawing region that you define (generally, the update region of a window), `DeviceLoop` calls your drawing function. Your drawing function should analyze the pixel depth passed in the `depth` parameter and the values passed in the `deviceFlags` parameter, and then draw in a manner that is optimized for the current device.

When highlighting, for example, your application might invert black and white when drawing onto a 1-bit video device but use magenta as the highlight color when drawing onto a color video device. In this case, even were your window to span both a black-and-white and a color screen, the user sees the selection inverted on the black-and-white screen, while magenta would be used to highlight the selection on the color screen.

You must provide a pointer to your `MyDeviceLoopDrawingCallback` function in the `drawingProc` parameter for `DeviceLoop`.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

DragGrayRgnProcPtr

```
typedef void (*DragGrayRgnProcPtr) (
);
```

If you name your function `MyDragGrayRgnProc`, you would declare it like this:

```
void DragGrayRgnProcPtr ();
```

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

QDArcProcPtr

```
typedef void (*QDArcProcPtr) (
   GrafVerb verb,
   const Rect * r,
   short startAngle,
   short arcAngle
);
```

If you name your function `MyQDArcProc`, you would declare it like this:

```
void QDArcProcPtr (
   GrafVerb verb,
   const Rect * r,
   short startAngle,
   short arcAngle
);
```

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

QDBitsProcPtr

```
typedef void (*QDBitsProcPtr) (
   const BitMap * srcBits,
   const Rect * srcRect,
   const Rect * dstRect,
   short mode,
   RgnHandle maskRgn
);
```

If you name your function `MyQDBitsProc`, you would declare it like this:

```
void QDBitsProcPtr (
   const BitMap * srcBits,
   const Rect * srcRect,
   const Rect * dstRect,
   short mode,
   RgnHandle maskRgn
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDCommentProcPtr

```
typedef void (*QDCommentProcPtr) (
   short kind,
   short dataSize,
   Handle dataHandle
);
```

If you name your function `MyQDCommentProc`, you would declare it like this:

```
void QDCommentProcPtr (
   short kind,
   short dataSize,
   Handle dataHandle
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDGetPicProcPtr

```
typedef void (*QDGetPicProcPtr) (
   void * dataPtr,
   short byteCount
);
```

If you name your function `MyQDGetPicProc`, you would declare it like this:

```
void QDGetPicProcPtr (
   void * dataPtr,
   short byteCount
);
```


Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDJShieldCursorProcPtr

```
typedef void (*QDJShieldCursorProcPtr) (
   short left,
   short top,
   short right,
   short bottom
);
```

If you name your function `MyQDJShieldCursorProc`, you would declare it like this:

```
void QDJShieldCursorProcPtr (
   short left,
   short top,
   short right,
   short bottom
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDLineProcPtr

```
typedef void (*QDLineProcPtr) (
   Point newPt
);
```

If you name your function `MyQDLineProc`, you would declare it like this:

```
void QDLineProcPtr (
   Point newPt
);
```


Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDOpcodeProcPtr

```
typedef void (*QDOpcodeProcPtr) (
   const Rect * fromRect,
   const Rect * toRect,
   UInt16 opcode,
   SInt16 version
);
```

If you name your function `MyQDOpcodeProc`, you would declare it like this:

```
void QDOpcodeProcPtr (
   const Rect * fromRect,
   const Rect * toRect,
   UInt16 opcode,
   SInt16 version
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDOvalProcPtr

```
typedef void (*QDOvalProcPtr) (
   GrafVerb verb,
   const Rect * r
);
```

If you name your function `MyQDOvalProc`, you would declare it like this:

```
void QDOvalProcPtr (
   GrafVerb verb,
```



```
   const Rect * r
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDPolyProcPtr

```
typedef void (*QDPolyProcPtr) (
   GrafVerb verb,
   PolyHandle poly
);
```

If you name your function `MyQDPolyProc`, you would declare it like this:

```
void QDPolyProcPtr (
   GrafVerb verb,
   PolyHandle poly
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDPrinterStatusProcPtr

```
typedef OSStatus (*QDPrinterStatusProcPtr) (
   PrinterStatusOpcode opcode,
   CGrafPtr currentPort,
   void * printerStatus
);
```

If you name your function `MyQDPrinterStatusProc`, you would declare it like this:

```
OSStatus QDPrinterStatusProcPtr (
   PrinterStatusOpcode opcode,
```



```
   CGrafPtr currentPort,
   void * printerStatus
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDPutPicProcPtr

```
typedef void (*QDPutPicProcPtr) (
   const void * dataPtr,
   short byteCount
);
```

If you name your function `MyQDPutPicProc`, you would declare it like this:

```
void QDPutPicProcPtr (
   const void * dataPtr,
   short byteCount
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDRectProcPtr

```
typedef void (*QDRectProcPtr) (
   GrafVerb verb,
   const Rect * r
);
```

If you name your function `MyQDRectProc`, you would declare it like this:

```
void QDRectProcPtr (
```



```
   GrafVerb verb,
   const Rect * r
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDRgnProcPtr

```
typedef void (*QDRgnProcPtr) (
   GrafVerb verb,
   RgnHandle rgn
);
```

If you name your function `MyQDRgnProc`, you would declare it like this:

```
void QDRgnProcPtr (
   GrafVerb verb,
   RgnHandle rgn
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDRRectProcPtr

```
typedef void (*QDRRectProcPtr) (
   GrafVerb verb,
   const Rect * r,
   short ovalWidth,
   short ovalHeight
);
```


If you name your function `MyQDRRectProc`, you would declare it like this:

```
void QDRRectProcPtr (
   GrafVerb verb,
   const Rect * r,
   short ovalWidth,
   short ovalHeight
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDStdGlyphsProcPtr

```
typedef OSStatus (*QDStdGlyphsProcPtr) (
   void * dataStream,
   ByteCount size
);
```

If you name your function `MyQDStdGlyphsProc`, you would declare it like this:

```
OSStatus QDStdGlyphsProcPtr (
   void * dataStream,
   ByteCount size
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDTextProcPtr

```
typedef void (*QDTextProcPtr) (
   short byteCount,
   const void * textBuf,
```



```
   Point numer,
   Point denom
);
```

If you name your function `MyQDTextProc`, you would declare it like this:

```
void QDTextProcPtr (
   short byteCount,
   const void * textBuf,
   Point numer,
   Point denom
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDTxMeasProcPtr

```
typedef short (*QDTxMeasProcPtr) (
   short byteCount,
   const void * textAddr,
   Point * numer,
   Point * denom,
   FontInfo * info
);
```

If you name your function `MyQDTxMeasProc`, you would declare it like this:

```
short QDTxMeasProcPtr (
   short byteCount,
   const void * textAddr,
   Point * numer,
   Point * denom,
   FontInfo * info
);
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

RegionToRectsProcPtr

```
typedef OSStatus (*RegionToRectsProcPtr) (
   UInt16 message,
   RgnHandle rgn,
   const Rect * rect,
   void * refCon
);
```

If you name your function `MyRegionToRectsProc`, you would declare it like this:

```
OSStatus RegionToRectsProcPtr (
   UInt16 message,
   RgnHandle rgn,
   const Rect * rect,
   void * refCon
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

Data Types

BitMap

```
struct BitMap {
   Ptr baseAddr;
   short rowBytes;
   Rect bounds;
};
typedef struct BitMap BitMap;
   typedef BitMap * BitMapPtr;
```

Fields

`baseAddr`

A pointer to the beginning of the bit image.

`rowBytes`
: The offset in bytes from one row of the image to the next. The value of the `rowBytes` field must be less than $4000.

`bounds`
: The bitmap's boundary rectangle by default, the entire main screen.

Discussion
A bitmap, which is a data structure of type `BitMap`, defines a bit image in terms of the QuickDraw coordinate plane. (A bit image is a collection of bits in memory that form a grid.)

A bitmap has three parts: a pointer to a bit image, the row width of that image, and a boundary rectangle that links the local coordinate system of a graphics port to QuickDraw's global coordinate system and defines the area of the bit image into which QuickDraw can draw.

The width of the boundary rectangle determines how many bits of one row are logically owned by the bitmap. This width must not exceed the number of bits in each row of the bit image. The height of the boundary rectangle determines how many rows of the image are logically owned by the bitmap. The number of rows enclosed by the boundary rectangle must not exceed the number of rows in the bit image.

The boundary rectangle defines the local coordinate system used by the port rectangle for a graphics port (described next). The upper-left corner (which for a window is called the window origin) of the port rectangle usually has a vertical coordinate of 0 and a horizontal coordinate of 0, although you can use the function `SetOrigin` (page 411) to change the coordinates of the window origin.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

Bits16

```
typedef short Bits16[16];
```

Discussion
The `Bits16` array is used by the `Cursor` (page 86) structure to hold a black-and-white, 16-by-16 pixel square image.

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

CCrsr

```
struct CCrsr {
   short crsrType;
   PixMapHandle crsrMap;
   Handle crsrData;
   Handle crsrXData;
   short crsrXValid;
   Handle crsrXHandle;
   Bits16 crsr1Data;
   Bits16 crsrMask;
   Point crsrHotSpot;
   long crsrXTable;
   long crsrID;
};
typedef struct CCrsr CCrsr;
   typedef CCrsr * CCrsrPtr;
```

Fields

`crsrType`
: The type of cursor. Possible values are $8000 for a black-and-white cursor and $8001 for a color cursor.

`crsrMap`
: A handle to the `PixMap` structure defining the cursor's characteristics. When the screen depth is greater than 2 bits per pixel, the `crsrMap` field and the `crsrData` field define the image. The pixels within the mask replace the destination pixels. Color QuickDraw transfers the pixels outside the mask into the destination pixels using the XOR Boolean transfer mode. Therefore, if pixels outside the mask are white, the destination pixels aren't changed. If pixels outside the mask are all black, the destination pixels are inverted. All other values outside of the mask cause unpredictable results.

`crsrData`
: A handle to the cursor's pixel data. To work properly, a color cursor's image should contain white pixels `(R=G= B=$FFFF)` for the transparent part of the image, and black pixels `(R=G=B=$0000)` for the part of the image to be inverted, in addition to the other colors in the cursor's image. Thus, to define a cursor that contains two colors, it's necessary to use a 2-bit cursor image (that is, a four-color image.

`crsrXData`
: A handle to the expanded pixel image used internally by Color QuickDraw.

`crsrXValid`
: The depth of the expanded cursor image. If you change the cursor's data or color table, set this field to 0 to cause the cursor to be re-expanded. Never set it to any other values.

`crsrXHandle`
Reserved for future use.

`crsr1Data`
A 16-by-16 pixel image with a pixel depth of 1 to be displayed when the cursor is on screens with pixel depths of 1 or 2 bits.

`crsrMask`
The cursor's mask data. QuickDraw uses the mask to crop the cursor's outline into a background color or pattern. QuickDraw then draws the cursor into this shape. The same 1-bit mask is used with images specified by the `crsrData` and `crsr1Data` fields.

`crsrHotSpot`
The cursor's hot spot.

`crsrXTable`
Reserved for future use.

`crsrID`
The color table seed for the cursor.

Discussion
Your application typically does not create `CCrsr` structures. Although you can create a `CCrsr` structure, it is usually easier to create a color cursor in a color cursor resource, 'crsr'.

A color cursor is a 256-pixel color image in a 16-by-16 pixel square defined in a color cursor (`'crsr'`) resource. When your application uses the `GetCCursor` function to get a color cursor from a `'crsr'` resource, `GetCCursor` loads the resource into memory as a `CCrsr` structure. Your application can then display the color cursor by using the `SetCCursor` (page 401) function.

`CCrsr` is substantially different from the `Cursor` structure. The fields `crsr1Data`, `crsrMask`, and `crsrHotSpot` in the `CCrsr` structure are the only ones that have counterparts in the `Cursor` structure.

The first four fields of the `CCrsr` structure are similar to the first four fields of the `PixPat` record, and are used in the same manner by QuickDraw.

The display of a cursor involves a relationship between a mask, stored in the `crsrMask` field with the same format used for 1-bit cursor masks, and an image. There are two possible sources for a color cursor's image. When the cursor is on a screen whose depth is 1 or 2 bits per pixel, the image for the cursor is taken from the `crsr1Data` field, which contains bitmap cursor data, similar to the bitmap in a `'CURS'` resource.

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

CGrafPort

```
struct CGrafPort {
   SInt16 device;
   PixMapHandle portPixMap;
   SInt16 portVersion;
   Handle grafVars;
   SInt16 chExtra;
   SInt16 pnLocHFrac;
   Rect portRect;
   RgnHandle visRgn;
   RgnHandle clipRgn;
   PixPatHandle bkPixPat;
   RGBColor rgbFgColor;
   RGBColor rgbBkColor;
   Point pnLoc;
   Point pnSize;
   SInt16 pnMode;
   PixPatHandle pnPixPat;
   PixPatHandle fillPixPat;
   SInt16 pnVis;
   SInt16 txFont;
   StyleField txFace;
   SInt16 txMode;
   SInt16 txSize;
   Fixed spExtra;
   SInt32 fgColor;
   SInt32 bkColor;
   SInt16 colrBit;
   SInt16 patStretch;
   Handle picSave;
   Handle rgnSave;
   Handle polySave;
   CQDProcsPtr grafProcs;
};
```

CGrafPtr

```
typedef GrafPtr CGrafPtr;
```


Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

ColorComplementUPP

```
typedef ColorComplementProcPtr ColorComplementUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

ColorSearchUPP

```
typedef ColorSearchProcPtr ColorSearchUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

ColorSpec

```
struct ColorSpec {
   short value;
   RGBColor rgb;
};
typedef struct ColorSpec ColorSpec;
   typedef ColorSpec * ColorSpecPtr;
```


Fields

`value`

The pixel value assigned by QuickDraw for the color specified in the `rgb` field of this structure. QuickDraw assigns a pixel value based on the capabilities of the user's screen. For indexed devices, the pixel value is an index number assigned by QuickDraw to the closest color available on the indexed device for direct devices, this value expresses the best available red, green, and blue values for the color on the direct device.

`rgb`

An `RGBColor` (page 119) structure that fully specifies the color whose approximation QuickDraw specifies in the `value` field.

Discussion

When creating a `PixMap` (page 103) structure for an indexed device, QuickDraw creates a `ColorTable` structure that defines the best colors available for the pixel image on that graphics device. QuickDraw also stores a `ColorTable` structure for the currently available colors in the graphics device's CLUT.

One of the fields in a `ColorTable` structure requires a value of type `cSpecArray`, which is defined as an array of `ColorSpec` structures. Typically, your application never needs to create `ColorTable` structures or `ColorSpec` structures. For completeness, the data structure of type `ColorSpec` is shown here.

Availability

Available in OS X v10.0 and later.

Declared in

`IOMacOSTypes.h`

ColorTable

```
struct ColorTable {
   long ctSeed;
   short ctFlags;
   short ctSize;
   CSpecArray ctTable;
};
typedef struct ColorTable ColorTable;
   typedef ColorTable * CTabPtr;
   typedef CTabPtr * CTabHandle;
```


Fields

`ctSeed`

Identifies a particular instance of a color table. QuickDraw uses the `ctSeed` value to compare an indexed device's color table with its associated inverse table (a table it uses for fast color lookup). When the color table for a graphics device has been changed, QuickDraw needs to rebuild the inverse table.

`ctFlags`

Flags that distinguish pixel map color tables from color tables in `GDevice` structures.

`ctSize`

One less than the number of entries in the table.

`ctTable`

An array of `ColorSpec` (page 80) entries, each containing a pixel value and a color specified by an `RGBColor` structure.

Discussion

When creating a `PixMap` (page 103) structure for a particular graphics device, QuickDraw creates a `ColorTable` structure that defines the best colors available for the pixel image on that particular graphics device. QuickDraw also creates a `ColorTable` structure of all available colors for use by the CLUT on indexed devices.

Typically, your application needs to create `ColorTable` structures only if it uses the Palette Manager.

Your application should never need to directly change the fields of a `ColorTable` structure. If you find it absolutely necessary for your application to do so, immediately use the `CTabChanged` (page 178) function to notify QuickDraw that your application has changed the `ColorTable` structure.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

ConstPatternParam

```
typedef const Pattern* ConstPatternParam;
```

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

CProcRec

```
struct CProcRec {
   Handle nxtComp;
   ColorComplementUPP compProc;
};
typedef struct CProcRec CProcRec;
   typedef CProcRec * CProcPtr;
```

Fields

`nxtComp`

A handle to the next `CPRocRec` data structure in the list.

`compProc`

A pointer to a complement function, as described in `ColorComplementProcPtr` (page 62).

Discussion

The `CProcRec` data structure contains a pointer to a custom complement function and a pointer to the next complement function in the list.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

CQDProcs

```
struct CQDProcs {
   QDTextUPP textProc;
   QDLineUPP lineProc;
   QDRectUPP rectProc;
   QDRRectUPP rRectProc;
   QDOvalUPP ovalProc;
   QDArcUPP arcProc;
   QDPolyUPP polyProc;
   QDRgnUPP rgnProc;
   QDBitsUPP bitsProc;
   QDCommentUPP commentProc;
   QDTxMeasUPP txMeasProc;
   QDGetPicUPP getPicProc;
   QDPutPicUPP putPicProc;
   QDOpcodeUPP opcodeProc;
   UniversalProcPtr newProc1;
```

```
   QDStdGlyphsUPP glyphsProc;
   QDPrinterStatusUPP printerStatusProc;
   UniversalProcPtr newProc4;
   UniversalProcPtr newProc5;
   UniversalProcPtr newProc6;
};
typedef struct CQDProcs CQDProcs;
   typedef CQDProcs * CQDProcsPtr;
```

Fields

`textProc`

A pointer to the low-level function that draws text. The standard QuickDraw function is the `StdText` function.

`lineProc`

A pointer to the low-level function that draws lines. The standard QuickDraw function is the `StdLine` function.

`rectProc`

A pointer to the low-level function that draws rectangles. The standard QuickDraw function is the `StdRect` function.

`rRectProc`

A pointer to the low-level function that draws rounded rectangles. The standard QuickDraw function is the `StdRRect` function.

`ovalProc`

A pointer to the low-level function that draws ovals. The standard QuickDraw function is the `StdOval` function.

`arcProc`

A pointer to the low-level function that draws arcs. The standard QuickDraw function is the `StdArc` function.

`polyProc`

A pointer to the low-level function that draws polygons. The standard QuickDraw function is the `StdPoly` function.

`rgnProc`

A pointer to the low-level function that draws regions. The standard QuickDraw function is the `StdRgn` function.

`bitsProc`

A pointer to the low-level function that copies bitmaps. The standard QuickDraw function is the `StdBits` function.

`commentProc`
: A pointer to the low-level function for processing a picture comment. The standard QuickDraw function is the `StdComment` function.

`txMeasProc`
: A pointer to the low-level function for measuring text width. The standard QuickDraw function is the `StdTxMeas` function.

`getPicProc`
: A pointer to the low-level function for retrieving information from the definition of a picture. The standard QuickDraw function is the `StdGetPic` function.

`putPicProc`
: A pointer to the low-level function for saving information as the definition of a picture. The standard QuickDraw function is the `StdPutPic` function.

`opcodeProc`
: Reserved for future use.

`newProc1`
: Reserved for future use.

`glyphsProc`
: Reserved for future use.

`printerStatusProc`
: Reserved for future use.

`newProc4`
: Reserved for future use.

`newProc5`
: Reserved for future use.

`newProc6`
: Reserved for future use.

Discussion

Use the `CQDProcs` structure only if you customize one or more of QuickDraw's standard low-level drawing functions. Use the `SetStdCProcs` (page 429) function to create a `CQDProcs` structure.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

CSpecArray

```
typedef ColorSpec CSpecArray[1];
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

Cursor

```
struct Cursor {
   Bits16 data;
   Bits16 mask;
   Point hotSpot;
};
typedef struct Cursor Cursor;
   typedef Cursor * CursPtr;
```

Fields
`data`

Cursor image data, which must begin on a word boundary.

`mask`

The cursor's mask. QuickDraw uses the mask to crop the cursor's outline into a background color or pattern. QuickDraw then draws the cursor into this shape.

`hotSpot`

A point in the image that aligns a point (not a bit) in the image with the mouse location on the screen. Whenever the user moves the mouse, the low-level interrupt-driven mouse functions move the cursor. When the user clicks, the Event Manager function `WaitNextEvent` reports the location of the cursor's hot spot in global coordinates.

Discussion
Your application typically does not create `Cursor` structures. Although you can create a `Cursor` structure and its associated `Bits16` array in your program code, it is usually easier to create a black-and-white cursor in a cursor resource,'CURS'.

A cursor is a 256-pixel, black-and-white image in a 16-by-16 pixel square. When your application uses the `GetCursor` (page 232) function to get a cursor from a `'CURS'` resource, `GetCursor` loads the resource into memory as a `Cursor` structure. Your application then displays the color cursor by using the `SetCursor` (page 405) function.

The cursor appears on the screen as a 16-by-16 pixel square. The appearance of each bit if the square is determined by the corresponding bits in the data and the mask and, if the mask bit is 0, by the pixel under the cursor. The four possible combinations of values for the data bit and the mask bit are:

- Data bit 0, Mask bit 1. The resulting pixel on the screen is white.
- Data bit 1, Mask bit 1. The resulting pixel on the screen is black.
- Data bit 0, Mask bit 0. The resulting pixel on the screen is the same as the pixel under the cursor.
- Data bit 1, Mask bit 0. The resulting pixel on the screen is the inverse of the pixel under the cursor.

Notice that if all mask bits are 0, the cursor is completely transparent, in that the image under the cursor can still be viewed. Pixels under the white part of the cursor appear unchanged; under the black part of the cursor, black pixels show through as white.

Basic QuickDraw supplies a predefined cursor in the global variable named `arrow`; this is the standard arrow cursor.

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

CursorImageRec

```
struct CursorImageRec {
   UInt16 majorVersion;
   UInt16 minorVersion;
   PixMapHandle cursorPixMap;
   BitMapHandle cursorBitMask;
};
typedef struct CursorImageRec CursorImageRec;
   typedef CursorImageRec * CursorImagePtr;
```

Availability
Available in OS X v10.0 through OS X v10.4.

Declared in
Quickdraw.h

CursorInfo

```
struct CursorInfo {
   long version;
   long capabilities;
   long animateDuration;
   Rect bounds;
   Point hotspot;
   long reserved;
};
typedef struct CursorInfo CursorInfo;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
QuickdrawAPI.h

CustomXFerRec

```
struct CustomXFerRec {
   UInt32 version;
   void * srcPixels;
   void * destPixels;
   void * resultPixels;
   UInt32 refCon;
   UInt32 pixelSize;
   UInt32 pixelCount;
   Point firstPixelHV;
   Rect destBounds;
};
typedef struct CustomXFerRec CustomXFerRec;
   typedef CustomXFerRec * CustomXFerRecPtr;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
QuickdrawAPI.h

CWindowPtr

```
typedef WindowPtr CWindowPtr;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

DeviceLoopDrawingUPP

```
typedef DeviceLoopDrawingProcPtr DeviceLoopDrawingUPP;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

DeviceLoopFlags

```
typedef unsigned long DeviceLoopFlags;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

DialogPtr

An opaque type that represents a dialog.

```
typedef struct OpaqueDialogPtr * DialogPtr;
```

Discussion
This is a Dialog Manager data type, defined in QuickDraw for historical reasons. Its role in Mac OS X is to serve as the basis for the widely used `DialogRef` data type.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

DragConstraint

```
typedef UInt16 DragConstraint;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

DragGrayRgnUPP

```
typedef DragGrayRgnProcPtr DragGrayRgnUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

GammaTbl

```
struct GammaTbl {
   short gVersion;
   short gType;
   short gFormulaSize;
   short gChanCnt;
   short gDataCnt;
   short gDataWidth;
   short gFormulaData[1];
};
typedef struct GammaTbl GammaTbl;
   typedef GammaTbl * GammaTblPtr;
```


Availability
Available in OS X v10.0 and later.

Declared in
`IOMacOSTypes.h`

GDevice

```
struct GDevice {
   short gdRefNum;
   short gdID;
   short gdType;
   ITabHandle gdITable;
   short gdResPref;
   SProcHndl gdSearchProc;
   CProcHndl gdCompProc;
   short gdFlags;
   PixMapHandle gdPMap;
   long gdRefCon;
   GDHandle gdNextGD;
   Rect gdRect;
   long gdMode;
   short gdCCBytes;
   short gdCCDepth;
   Handle gdCCXData;
   Handle gdCCXMask;
   long gdReserved;
};
typedef struct GDevice GDevice;
   typedef GDevice * GDPtr;
   typedef GDPtr * GDHandle;
```

Fields
`gdRefNum`

The reference number of the driver for the screen associated with the video device. For most video devices, this information is set at system startup time.

`gdID`

Reserved. If you create your own `GDevice` structure, set this field to 0.

`gdType`

The general type of graphics device. See "Graphics Device Type Constants" (page 128) for a description of the values which you can use in this field.

`gdITable`

A handle to the inverse table for color mapping.

`gdResPref`

The preferred resolution for inverse tables.

`gdSearchProc`

A handle to the list of search functions. Its value is `NULL` for the default function.

`gdCompProc`

A handle to a list of complement functions. Its value is `NULL` for the default function.

`gdFlags`

The `GDevice` structure's attributes. To set the attribute bits in the `gdFlags` field, use the `SetDeviceAttribute` (page 406) function. Do not set `gdFlags` directly in the `GDevice` structure.

`gdPMap`

A handle to a `PixMap` structure giving the dimension of the image buffer, along with the characteristics of the graphics device (resolution, storage format, color depth, and color table). For `GDevice` structures, the high bit of the global variable

```
(((**TheGDevice).**gdPMap).**pmTable).ctFlags
```

is always set.

`gdRefCon`

A value used by system software to pass device-related parameters. Since a graphics device is shared, do not store data here.

`gdNextGD`

A handle to the next graphics device in the device list. If this is the last graphics device in the device list, the field contains 0.

`gdRect`

The boundary rectangle of the graphics device represented by the `GDevice` structure. The main screen has the upper-left corner of the rectangle set to (0,0). All other graphics devices are relative to this point.

`gdMode`

The current setting for the graphics device mode. This value is passed to the video driver to set its pixel depth and to specify color or black and white; applications do not need this information.

`gdCCBytes`

The `rowBytes` value of the expanded cursor. Your application should not change this field.

`gdCCDepth`

The depth of the expanded cursor. Your application should not change this field.

`gdCCXData`

A handle to the cursor's expanded data. Your application should not change this field.

`gdCCXMask`

A handle to the cursor's expanded mask. Your application should not change this field.

`gdReserved`
Reserved for future expansion; it must be set to 0 for future compatibility.

Discussion
Color QuickDraw stores state information for video devices and offscreen graphics worlds in `GDevice` structures. When the system starts up, it allocates and initializes one handle to a `GDevice` structure for each video device it finds. When you use the Offscreen Graphics Devices function, `NewGWorld`, Color QuickDraw automatically creates a `GDevice` structure for the new offscreen graphics world. The system links these `GDevice` structures in a list, called the device list. (You can find a handle to the first element in the device list in the global variable `DeviceList`.) By default, the `GDevice` structure corresponding to the first video device found is marked as the current device. All other graphics devices in the list are initially marked as inactive.

When the user moves a window or creates a window on another screen, and your application draws into that window, Color QuickDraw automatically makes the video device for that screen the current device. Color QuickDraw stores that information in the global variable `TheGDevice`.

`GDevice` structures that correspond to video devices have drivers associated with them. These drivers can be used to change the mode of the video device from black and white to color and to change the pixel depth. Application-created `GDevice` structures usually don't require drivers.

Your application should never need to directly change the fields of a `GDevice` structure. If you find it absolutely necessary for your application to so, immediately use the `GDeviceChanged` function to notify QuickDraw that your application has changed the `GDevice` structure.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

GrafPort

```
struct GrafPort {
   SInt16 device;
   BitMap portBits;
   Rect portRect;
   RgnHandle visRgn;
   RgnHandle clipRgn;
   Pattern bkPat;
   Pattern fillPat;
   Point pnLoc;
   Point pnSize;
   SInt16 pnMode;
```



```
   Pattern pnPat;
   SInt16 pnVis;
   SInt16 txFont;
   StyleField txFace;
   SInt16 txMode;
   SInt16 txSize;
   Fixed spExtra;
   SInt32 fgColor;
   SInt32 bkColor;
   SInt16 colrBit;
   SInt16 patStretch;
   Handle picSave;
   Handle rgnSave;
   Handle polySave;
   QDProcsPtr grafProcs;
};
```

GrafPtr

```
typedef struct OpaqueGrafPtr * GrafPtr;
```

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

GrafVars

```
struct GrafVars {
   RGBColor rgbOpColor;
   RGBColor rgbHiliteColor;
   Handle pmFgColor;
   short pmFgIndex;
   Handle pmBkColor;
   short pmBkIndex;
   short pmFlags;
};
typedef struct GrafVars GrafVars;
   typedef GrafVars * GVarPtr;
```


Fields

`rgbOpColor`

The color for the arithmetic transfer operations `addPin`, `subPin`, and `blend`.

`rgbHiliteColor`

The highlight color for this graphics port.

`pmFgColor`

A handle to the palette that contains the foreground color.

`pmFgIndex`

The index value into the palette for the foreground color.

`pmBkColor`

A handle to the palette that contains the background color.

`pmBkIndex`

The index value into the palette for the background color.

`pmFlags`

Flags private to the Palette Manager.

Discussion

The `GrafVars` structure contains color information in addition to that in the `CGrafPort` structure, of which it is logically a part; the information is used by QuickDraw and the Palette Manager.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

GrafVerb

```
typedef SInt8 GrafVerb;
```

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

GWorldFlags

```
typedef unsigned long GWorldFlags;
```

Discussion
Several functions expect or return values defined by the `GWorldFlags` data type. See "Graphics World Flags" (page 129) for a detailed description of these flags.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

GWorldPtr

Defines a pointer to a structure that your application can use to refer to an offscreen graphics world.

```
typedef CGrafPtr GWorldPtr;
```

Discussion
An offscreen graphics world in color QuickDraw contains a `CGrafPort` structure—and its handles to associated `PixMap` and `ColorTable` structures—that describes an offscreen graphics port and contains references to a `GDevice` structure and other state information. The actual data structure for an offscreen graphics world is kept private to allow for future extensions. However, when your application uses the `NewGWorld` function to create an offscreen world, `NewGWorld` returns a pointer of type `GWorldPtr` by which your application refers to the offscreen graphics world.

On computers lacking color QuickDraw, `GWorldPtr` points to an extension of the `GrafPort` structure.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

ITab

```
struct ITab {
   long iTabSeed;
```



```
   short iTabRes;
   Byte iTTable[1];
};
typedef struct ITab ITab;
   typedef ITab * ITabPtr;
   typedef ITabPtr * ITabHandle;
```

Fields

`iTabSeed`

The `iTabSeed` value, initially set from the corresponding CLUT's `ctSeed` field. If at any time these do not match, then the color table was changed, and the inverse table needs to be rebuilt.

`iTabRes`

The resolution of this inverse table.

`iTTable`

An array of index values. The size of the `iTabTable` field in bytes is `23*iTabRes`.

Discussion

The `ITab` data structure contains the inverse table information that the Color Manager uses for fast mapping of RGB color values.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

MacPolygon

```
struct MacPolygon {
   short polySize;
   Rect polyBBox;
   Point polyPoints[1];
};
typedef struct MacPolygon MacPolygon;
   typedef MacPolygon Polygon;
   typedef MacPolygon * PolyPtr;
   typedef PolyPtr * PolyHandle;
```

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

MacRegion

```
struct MacRegion {
   UInt16 rgnSize;
   Rect rgnBBox;
};
typedef struct MacRegion MacRegion;
```

MatchRec

```
struct MatchRec {
   unsigned short red;
   unsigned short green;
   unsigned short blue;
   long matchData;
};
typedef struct MatchRec MatchRec;
```

Fields

`red`

Red value of the seed.

`green`

Green value of the seed.

`blue`

Blue value of the seed.

`matchData`

The value passed in the `matchData` parameter of the `SeedCFill` or `CalcCMask` function.

Discussion

When `SeedCFill` (page 398) or `CalcCMask` (page 155) calls your color search function, the `GDRefCon` field of the current `GDevice` structure contains a pointer to a `MatchRec` structure. This structure contains the RGB value of the seed pixel or seed color for which your color search function searches.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

OpenCPicParams

```
struct OpenCPicParams {
   Rect srcRect;
   Fixed hRes;
   Fixed vRes;
   short version;
   short reserved1;
   long reserved2;
};
typedef struct OpenCPicParams OpenCPicParams;
```

Fields

`srcRect`

The optimal bounding rectangle for the resolution indicated by the `hRes` and `vRes` fields. When you later call the `DrawPicture` (page 192) function to play back the saved picture, specify a destination rectangle and `DrawPicture` scales the picture so that it is completely aligned with the destination rectangle.

`hRes`

The best horizontal resolution for the picture. A value of $00480000 specifies a horizontal resolution of 72 dpi.

`vRes`

The best vertical resolution for the picture. A value of $00480000 specifies a vertical resolution of 72 dpi.

`version`

Always set this field to –2.

`reserved1`

Reserved; set to 0.

`reserved2`

Reserved; set to 0.

Discussion

When you use the `OpenCPicture` function to begin creating a picture, you must pass it information in an `OpenCPicParams` structure. This structure provides a simple mechanism for specifying resolutions when creating images. For example, applications that create pictures from scanned images can specify resolutions higher than 72 dpi for these pictures in `OpenCPicParams` structures.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

Pattern

```
struct Pattern {
   UInt8 pat[8];
};
typedef struct Pattern Pattern;
   typedef Pattern * PatPtr;
   typedef PatPtr * PatHandle;
```

Discussion

Your application typically does not create `Pattern` structures. Although you can create `Pattern` structures in your program code, it is usually easier to create bit patterns using the pattern, 'PAT', or pattern list, 'PAT#', resource.

A bit pattern is a 64-bit image, organized as an 8-by-8 bit square, that defines a repeating design or tone. When a pattern is drawn, it is aligned so that adjacent areas of the same pattern in the same graphics port form a continuous, coordinated pattern. QuickDraw provides predefined patterns in global variables named `white`, `black`, `gray`, `ltGray`, and `dkGray`. The row width of a pattern is 1 byte.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

PenState

```
struct PenState {
   Point pnLoc;
   Point pnSize;
   short pnMode;
   Pattern pnPat;
};
typedef struct PenState PenState;
```

Fields

`pnLoc`

For the current graphics port at the time the `GetPenState` function was called, the value of that graphics port's `pnLoc` field. This value is the point where QuickDraw begins drawing next. The location of the graphics pen is a point in the graphics port's coordinate system, not a pixel in a bit image. The upper-left corner of the pen is at the pen location the graphics pen hangs below and to the right of this point.

`pnSize`

For the current graphics port at the time the `GetPenState` function was called, the value of that graphics port's `pnSize` field. The graphics pen is rectangular in shape, and its width and height are specified by the values in the `pnSize` field. The default size is a 1-by-1 bit square; the width and height can range from 0 by 0 to 32,767 by 32,767. If either the pen width or the pen height is 0, the pen does not draw. Heights or widths of less than 0 are undefined.

`pnMode`

The pattern mode—that is, for the current graphics port at the time the `GetPenState` function was called, the value of that graphics port's `pnMode` field. This value determines how the pen pattern is to affect what's already in the bit image when lines or shapes are drawn. When the graphics pen draws, QuickDraw first determines what bits in the bit image are affected, finds their corresponding bits in the pattern, and then transfers the bits from the pattern into the image according to this mode, which specifies one of eight Boolean transfer operations. The resulting bit is stored into its proper place in the bit image.

`pnPat`

For the current graphics port at the time the `GetPenState` function was called, the pen pattern for that graphics port. This pattern determines how the bits under the graphics pen are affected when lines or shapes are drawn.

Discussion

The `GetPenState` (page 245) function saves the location, size, pattern, and pattern mode of the graphics pen for the current graphics port in a `PenState` structure, which is a data structure of type `PenState`. After changing the graphics pen as necessary, you can later restore these pen states with the `SetPenState` (page 413) function.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

Picture

```
struct Picture {
   short picSize;
   Rect picFrame;
};
typedef struct Picture Picture;
   typedef Picture * PicPtr;
   typedef PicPtr * PicHandle;
```


Fields

`picSize`

The size of the rest of this structure for a version 1 picture. To maintain compatibility with the version 1 picture format, the `picSize` field was not changed for the version 2 picture or extended version 2 formats. The information in this field is used only for version 1 pictures, which cannot exceed 32 KB in size. Because version 2 and extended version 2 pictures can be much larger than the 32 KB limit imposed by the 2-byte `picSize` field, you should use the Memory Manager function `GetHandleSize` to determine the size of a picture in memory; you should use the File Manager function `PBGetFInfo` to determine the size of a picture in a `'PICT'` file; and you should use the Resource Manager function `GetMaxResourceSize` to determine the size of a `'PICT'` resource.

`picFrame`

The bounding rectangle for the picture defined in the rest of this structure. The `DrawPicture` function uses this rectangle to scale the picture if you draw it into a destination rectangle of a different size.

Discussion

When you use the `OpenCPicture` (page 345) or `OpenPicture` (page 348) function, QuickDraw begins collecting your subsequent drawing commands in a `Picture` structure. (You use the `ClosePicture` function to complete a picture definition.) When you use the `GetPicture` (page 246) function to retrieve a picture stored in a resource, `GetPicture` reads the resource into memory as a `Picture` structure. By using the `DrawPicture` (page 192) procedure, you can draw onscreen the picture defined by the commands stored in the `Picture` structure.

A picture opcode is a number that the `DrawPicture` function uses to determine what object to draw or what mode to change for subsequent drawing. Generally, do not read or write this picture data directly. Instead, use the `OpenCPicture` (or `OpenPicture`), `ClosePicture`, and `DrawPicture` functions to process these opcodes.

The `Picture` structure can also contain picture comments. Created by applications using the `PicComment` function, picture comments contain data or commands for special processing by output devices, such as PostScript printers.

You can use File Manager functions to save the picture in a file of type `'PICT'`, you can use Resource Manager functions to save the picture in a resource of type `'PICT'`, and you can use the Scrap Manager function `PutScrap` to store the picture in `'PICT'` scrap format.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

PixelType

```
typedef SInt8 PixelType;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

PixMap

```
struct PixMap {
   Ptr baseAddr;
   short rowBytes;
   Rect bounds;
   short pmVersion;
   short packType;
   long packSize;
   Fixed hRes;
   Fixed vRes;
   short pixelType;
   short pixelSize;
   short cmpCount;
   short cmpSize;
   long planeBytes;
   CTabHandle pmTable;
   long pmReserved;
};
typedef struct PixMap PixMap;
   typedef PixMap * PixMapPtr;
   typedef PixMapPtr * PixMapHandle;
```

Fields
`baseAddr`

For an onscreen pixel image, a pointer to the first byte of the image. For optimal performance, this should be a multiple of 4. The pixel image that appears on a screen is normally stored on a graphics card rather than in main memory.

Note that the `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer. You must use the `GetPixBaseAddr` function to obtain a pointer to the `PixMap` structure for an offscreen graphics world. Your application should never directly access the `baseAddr` field of the `PixMap` structure for an offscreen graphics world.

`rowBytes`

The offset in bytes from one row of the image to the next. The value must be even, less than $4000, and for best performance it should be a multiple of 4. The high 2 bits of `rowBytes` are used as flags. If bit 15 = 1, the data structure pointed to is a `PixMap` structure; otherwise it is a `BitMap` structure.

`bounds`

The boundary rectangle, which links the local coordinate system of a graphics port to QuickDraw's global coordinate system and defines the area of the bit image into which QuickDraw can draw. By default, the boundary rectangle is the entire main screen. Do not use the value of this field to determine the size of the screen instead use the value of the `gdRect` field of the `GDevice` structure for the screen.

`pmVersion`

The version number of QuickDraw that created this `PixMap` structure. The value of `pmVersion` is normally 0. If `pmVersion` is 4, QuickDraw treats the `PixMap` structure's `baseAddr` field as 32-bit clean. All other flags are private. Most applications never need to set this field.

`packType`

The packing algorithm used to compress image data. QuickDraw currently supports a `packType` of 0, which means no packing, and values of 1 to 4 for packing direct pixels.

`packSize`

The size of the packed image in bytes. When the `packType` field contains the value 0, this field is always set to 0.

`hRes`

The horizontal resolution of the pixel image in pixels per inch. This value is of type `Fixed`; by default, the value here is $00480000 (for 72 pixels per inch).

`vRes`

The vertical resolution of the pixel image in pixels per inch. This value is of type `Fixed`; by default, the value here is $00480000 (for 72 pixels per inch).

`pixelType`

The storage format for a pixel image. Indexed pixels are indicated by a value of 0. Direct pixels are specified by a value of `RGBDirect`, or 16. In the `PixMap` structure of the `GDevice` structure for a direct device, this field is set to the constant `RGBDirect` when the screen depth is set.

`pixelSize`

Pixel depth; that is, the number of bits used to represent a pixel. Indexed pixels can have sizes of 1, 2, 4, and 8 bits; direct pixel sizes are 16 and 32 bits.

`cmpCount`

The number of components used to represent a color for a pixel. With indexed pixels, each pixel is a single value representing an index in a color table, and therefore this field contains the value 1—the index is the single component. With direct pixels, each pixel contains three components—one integer each for the intensities of red, green, and blue—so this field contains the value 3.

`cmpSize`

The size in bits of each component for a pixel. QuickDraw expects that the sizes of all components are the same, and that the value of the `cmpCount` field multiplied by the value of the `cmpSize` field is less than or equal to the value in the `pixelSize` field.

For an indexed pixel value, which has only one component, the value of the `cmpSize` field is the same as the value of the `pixelSize` field—that is, 1, 2, 4, or 8.

For direct pixels there are two additional possibilities:

- A 16-bit pixel, which has three components, has a `cmpSize` value of 5. This leaves an unused high-order bit, which QuickDraw sets to 0.
- A 32-bit pixel, which has three components (red, green, and blue), has a `cmpSize` value of 8. This leaves an unused high-order byte, which QuickDraw sets to 0.

Generally, therefore, your application should clear the memory for the image to 0 before creating a 16-bit or 32-bit image. The Memory Manager functions `NewHandleClear` and `NewPtrClear` assist you in allocating pre-zeroed memory.

`planeBytes`

The offset in bytes from one drawing plane to the next. This field is set to 0.

`pmTable`

A handle to a `ColorTable` structure for the colors in this pixel map.

`pmReserved`

Reserved for future expansion. This field must be set to 0 for future compatibility.

Discussion

The `PixMap` structure contains information about the dimensions, contents, storage format, depth, resolution, and color usage of a pixel image. The pixel map for a window's color graphics port always consists of the pixel depth, color table, and boundary rectangle of the main screen, even if the window is created on or moved to an entirely different screen.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

PixPat

```
struct PixPat {
   short patType;
   PixMapHandle patMap;
   Handle patData;
   Handle patXData;
   short patXValid;
   Handle patXMap;
   Pattern pat1Data;
};
typedef struct PixPat PixPat;
   typedef PixPat * PixPatPtr;
   typedef PixPatPtr * PixPatHandle;
```

Fields

`patType`

The pattern's type. The value 0 specifies a basic QuickDraw bit pattern, the value 1 specifies a full-color pixel pattern, and the value 2 specifies an RGB pattern.

`patMap`

A handle to a `PixMap` (page 103) structure that describes the pattern's pixel image. The `PixMap` structure can contain indexed or direct pixels.

`patData`

A handle to the pattern's pixel image.

`patXData`

A handle to an expanded pixel image used internally by QuickDraw.

`patXValid`

A flag that, when set to –1, invalidates the expanded data.

`patXMap`

Reserved for use by QuickDraw.

`pat1Data`

A bit pattern to be used when this pattern is drawn into a `GrafPort` structure. The `NewPixPat` (page 334) function sets this field to 50 percent gray.

Discussion

Your application typically does not create `PixPat` structures. Although you can create such structures in your program code, it is usually easier to create pixel patterns using the pixel pattern resource, `'ppat'`.

When used for a color graphics port, the basic QuickDraw functions `PenPat` and `BackPat` store pixel patterns in, respectively, the `pnPixPat` and `bkPixPat` fields of the `CGrafPort` structure and set the `patType` field of the `PixPat` field to 0 to indicate that the `PixPat` structure contains a bit pattern. Such patterns are limited to 8-by-8 pixel dimensions and, instead of being drawn in black and white, are always drawn using the colors specified in the `CGrafPort` structure's `rgbFgColor` and `rgbBkColor` fields, respectively.

In a full-color pixel pattern, the `patType` field contains the value 1, and the pattern's dimensions, depth, resolution, set of colors, and other characteristics are defined by a `PixMap` structure, referenced by the handle in the `patMap` field of the `PixPat` structure. Full-color pixel patterns contain color tables that describe the colors they use. Generally such a color table contains one entry for each color used in the pattern. For instance, if your pattern has five colors, you would probably create a 4 bits per pixel pattern that uses pixel values 0–4, and a color table with five entries, numbered 0–4, that contain the RGB specifications for those pixel values.

However, if you don't specify a color table for a pixel value, QuickDraw assigns a color to that pixel value. The largest unassigned pixel value becomes the foreground color the smallest unassigned pixel value is assigned the background color. Remaining unassigned pixel values are given colors that are evenly distributed between the foreground and background.

For instance, in the color table mentioned above, pixel values 5–15 are unused. Assume that the foreground color is black and the background color is white. Pixel value 15 is assigned the foreground color, black pixel value 5 is assigned the background color, white the nine pixel values between them are assigned evenly distributed shades of gray. If the `PixMap` structure's color table is set to `NULL`, all pixel values are determined by blending the foreground and background colors.

Full-color pixel patterns are not limited to a fixed size: their height and width can be any power of 2, as specified by the height and width of the boundary rectangle for the `PixMap` structure specified in the `patMap` field. A pattern 8 bits wide, which is the size of a bit pattern, has a row width of just 1 byte, contrary to the usual rule that the `rowBytes` field must be even. Read this pattern type into memory using the `GetPixPat` (page 251) function, and set it using the `PenPixPat` (page 363) or `BackPixPat` (page 153) functions.

The pixel map specified in the `patMap` field of the `PixPat` structure defines the pattern's characteristics. The `baseAddr` field of the `PixMap` structure for that pixel map is ignored. For a full-color pixel pattern, the actual pixel image defining the pattern is stored in the handle in the `patData` field of the `PixPat` structure. The pattern's pixel depth need not match that of the pixel map into which it's transferred the depth is adjusted automatically when the pattern is drawn. QuickDraw maintains a private copy of the pattern's pixel image, expanded to the current screen depth and aligned to the current graphics port, in the `patXData` field of the `PixPat` structure.

In an RGB pixel pattern, the `patType` field contains the value 2. Using the `MakeRGBPat` (page 318) function, your application can specify the exact color it wants to use. QuickDraw selects a pattern to approximate that color. In this way, your application can effectively increase the color resolution of the screen. RGB pixel patterns are particularly useful for dithering: mixing existing colors together to create the illusion of a third color that's

unavailable on an indexed device. The `MakeRGBPat` function aids in this process by constructing a dithered pattern to approximate a given absolute color. An RGB pixel pattern can display 125 different patterns on a 4-bit screen, or 2197 different patterns on an 8-bit screen.

An RGB pixel pattern has an 8-by-8 pixel pattern that is 2 bits deep. For an RGB pixel pattern, the `RGBColor` structure that you specify to the `MakeRGBPat` function defines the image; there is no image data.

Your application should never need to directly change the fields of a `PixPat` structure. If you find it absolutely necessary for your application to so, immediately use the `PixPatChanged` (page 368) function to notify QuickDraw that your application has changed the `PixPat` structure.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

Polygon

```
typedef MacPolygon Polygon;
```

Discussion
After you use the `OpenPoly` function to create a polygon, QuickDraw begins collecting the line-drawing information you provide into a `MacPolygon` structure. The `OpenPoly` function returns a handle to the newly allocated `MacPolygon` structure. Thereafter, your application normally refers to your new polygon by this handle, because QuickDraw functions such as `FramePoly` and `PaintPoly` expect a handle to a `Polygon` as their first parameter.

A polygon is defined by a sequence of connected lines. A `MacPolygon` structure consists of two fixed-length fields followed by a variable-length array of points: the starting point followed by each successive point to which a line is drawn.

Your application typically does not need to create a `MacPolygon` structure.

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

PrinterFontStatus

```
struct PrinterFontStatus {
   SInt32 oResult;
   SInt16 iFondID;
   Style iStyle;
};
typedef struct PrinterFontStatus PrinterFontStatus;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

PrinterScalingStatus

```
struct PrinterScalingStatus {
   Point oScalingFactors;
};
typedef struct PrinterScalingStatus PrinterScalingStatus;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

PrinterStatusOpcode

```
typedef SInt32 PrinterStatusOpcode;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDArcUPP

```
typedef QDArcProcPtr QDArcUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDBitsUPP

```
typedef QDBitsProcPtr QDBitsUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDByte

```
typedef SignedByte QDByte;
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

QDCommentUPP

```
typedef QDCommentProcPtr QDCommentUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDErr

```
typedef short QDErr;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDGetPicUPP

```
typedef QDGetPicProcPtr QDGetPicUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDGlobals

```
struct QDGlobals {
   char privates[76];
   long randSeed;
   BitMap screenBits;
   Cursor arrow;
   Pattern dkGray;
   Pattern ltGray;
   Pattern gray;
   Pattern black;
   Pattern white;
   GrafPtr thePort;
};
typedef struct QDGlobals QDGlobals;
   typedef QDGlobals * QDGlobalsPtr;
```

QDJShieldCursorUPP

```
typedef QDJShieldCursorProcPtr QDJShieldCursorUPP;
```


Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDLineUPP

```
typedef QDLineProcPtr QDLineUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDOpcodeUPP

```
typedef QDOpcodeProcPtr QDOpcodeUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDOvalUPP

```
typedef QDOvalProcPtr QDOvalUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDPictRef

Defines an opaque data type that represents a QuickDraw picture in the Quartz 2D graphics environment.

```
typedef struct QDPict * QDPictRef;
```

Discussion

This opaque type is used to draw QuickDraw picture data in a Quartz context. (Quartz 2D defines an analogous opaque type called `CGPDFDocumentRef` which is used to draw PDF data in a Quartz context.) An instance of the `QDPictRef` type is called a **QDPict picture**. There are two ways to create a QDPict picture:

- You can call `QDPictCreateWithProvider` (page 461), passing in a Quartz data provider for the picture data. Typically the source of this data is a `'PICT'` resource.
- You can call `QDPictCreateWithURL` (page 462), passing in a Core Foundation URL that specifies a file with picture data in the data fork.

Both functions verify that picture header information is present, starting at either byte 1 or byte 513 of the picture data.

To draw a QDPict picture in a Quartz context, you call `QDPictDrawToCGContext` (page 463). To get the bounds or native resolution of a QDPict picture, you call `QDPictGetBounds` (page 464) or `QDPictGetResolution` (page 465).

When you draw a QDPict picture in a PDF context, you can save the drawing in a PDF file. This is the recommended way to convert QuickDraw pictures into single-page PDF documents.

These additional sources of information may be helpful:

- The sample Carbon program *CGDrawPicture* shows how to use this opaque type to draw QuickDraw pictures in a Quartz context.
- For general information about QuickDraw pictures and the PICT graphics format, see Chapter 7 in *Inside Macintosh: Imaging With QuickDraw* .

Availability

Available in OS X v10.1 through OS X v10.6.

Declared in

`QDPictToCGContext.h`

QDPolyUPP

```
typedef QDPolyProcPtr QDPolyUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDPrinterStatusUPP

```
typedef QDPrinterStatusProcPtr QDPrinterStatusUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDProcs

```
struct QDProcs {
   QDTextUPP textProc;
   QDLineUPP lineProc;
   QDRectUPP rectProc;
   QDRRectUPP rRectProc;
   QDOvalUPP ovalProc;
   QDArcUPP arcProc;
   QDPolyUPP polyProc;
   QDRgnUPP rgnProc;
   QDBitsUPP bitsProc;
   QDCommentUPP commentProc;
   QDTxMeasUPP txMeasProc;
   QDGetPicUPP getPicProc;
   QDPutPicUPP putPicProc;
};
typedef struct QDProcs QDProcs;
   typedef QDProcs * QDProcsPtr;
```


Fields

`textProc`

A pointer to the low-level function that draws text. The standard QuickDraw function is the `StdText` function.

`lineProc`

A pointer to the low-level function that draws lines. The standard QuickDraw function is the `StdLine` (page 439) function.

`rectProc`

A pointer to the low-level function that draws rectangles. The standard QuickDraw function is the `StdRect` (page 442) function.

`rRectProc`

A pointer to the low-level function that draws rounded rectangles. The standard QuickDraw function is the `StdRRect` (page 444) function.

`ovalProc`

A pointer to the low-level function that draws ovals. The standard QuickDraw function is the `StdOval` (page 440) function.

`arcProc`

A pointer to the low-level function that draws arcs. The standard QuickDraw function is the `StdArc` (page 435) function.

`polyProc`

A pointer to the low-level function that draws polygons. The standard QuickDraw function is the `StdPoly` (page 441) function.

`rgnProc`

A pointer to the low-level function that draws regions. The standard QuickDraw function is the `StdRgn` (page 443) function.

`bitsProc`

A pointer to the low-level function that copies bitmaps. The standard QuickDraw function is the `StdBits` (page 436) function.

`commentProc`

A pointer to the low-level function for processing a picture comment. The standard QuickDraw function is the `StdComment` (page 437) function.

`txMeasProc`

A pointer to the low-level function for measuring text width. The standard QuickDraw function is the `StdTxMeas` function.

`getPicProc`
: A pointer to the low-level function for retrieving information from the definition of a picture. The standard QuickDraw function is the `StdGetPic` (page 438) function.

`putPicProc`
: A pointer to the low-level function for saving information as the definition of a picture. The standard QuickDraw function is the `StdPutPic` (page 442) function.

Discussion
You need to use the `QDProcs` structure only if you customize one or more of QuickDraw's low-level drawing functions. Use `SetStdProcs` (page 430) to create a `QDProcs` structure.

The `QDProcs` structure contains pointers to low-level drawing functions. You can change the fields of this structure to point to functions of your own devising.

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

QDPutPicUPP

```
typedef QDPutPicProcPtr QDPutPicUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDRectUPP

```
typedef QDRectProcPtr QDRectUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDRegionBitsRef

```
typedef struct OpaqueQDRegionBitsRef * QDRegionBitsRef;
```

Availability
Available in OS X v10.1 through OS X v10.6.

Declared in
QuickdrawAPI.h

QDRegionParseDirection

```
typedef SInt32 QDRegionParseDirection;
```

Availability
Available in OS X v10.0 and later.

Declared in
Quickdraw.h

QDRgnUPP

```
typedef QDRgnProcPtr QDRgnUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
Quickdraw.h

QDRRectUPP

```
typedef QDRRectProcPtr QDRRectUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
Quickdraw.h

QDStdGlyphsUPP

```
typedef QDStdGlyphsProcPtr QDStdGlyphsUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDTextUPP

```
typedef QDTextProcPtr QDTextUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

QDTxMeasUPP

```
typedef QDTxMeasProcPtr QDTxMeasUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

RegionToRectsUPP

```
typedef RegionToRectsProcPtr RegionToRectsUPP;
```

Availability
Available in OS X v10.0 and later.

Declared in
`Quickdraw.h`

ReqListRec

```
struct ReqListRec {
   short reqLSize;
   short reqLData[1];
};
typedef struct ReqListRec ReqListRec;
```

Fields

`reqLSize`

The size of this `ReqListRec` data structure minus one.

`reqLData`

An array of integers representing offsets into a color table.

Discussion

The `ReqListRec` data structure is a parameter to the `SaveEntries` function by which you can describe color table entries to be saved.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

RGBColor

```
struct RGBColor {
   unsigned short red;
   unsigned short green;
   unsigned short blue;
};
typedef struct RGBColor RGBColor;
   typedef RGBColor * RGBColorPtr;
```

Fields

`red`

An unsigned integer specifying the red value of the color.

`green`

An unsigned integer specifying the green value of the color.

`blue`

An unsigned integer specifying the blue value of the color.

Discussion

You usually specify a color to QuickDraw by creating an `RGBColor` structure in which you assign the red, green, and blue values of the foreground color. For example, when you want to set the foreground color for drawing, you create an `RGBColor` structure that defines the foreground color you desire; then you pass that structure as a parameter to the `RGBForeColor` function.

In an `RGBColor` structure, three 16-bit unsigned integers give the intensity values for the three additive primary colors.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

RgnHandle

An opaque type that represents a QuickDraw region.

```
typedef struct OpaqueRgnHandle * RgnHandle;
```

Discussion

A region is an arbitrary area or set of areas on the QuickDraw coordinate plane. The outline of a region should be one or more closed loops.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

SProcRec

```
struct SProcRec {
   Handle nxtSrch;
   ColorSearchUPP srchProc;
};
typedef struct SProcRec SProcRec;
   typedef SProcRec * SProcPtr;
```


Fields

`nxtSrch`

A handle to the next `SProcRec` data structure in the chain of search functions.

`srchProc`

A pointer to a custom search function (described in `ColorSearchProcPtr` (page 62)).

Discussion

The `SProcRec` data structure contains a pointer to a custom search function and a handle to the next `SProcRec` data structure in the function list.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawTypes.h`

WindowPtr

An opaque type that represents a window.

```
typedef struct OpaqueWindowPtr * WindowPtr;
```

Discussion

This is a Window Manager data type, defined in QuickDraw for historical reasons. Its role in Mac OS X is to serve as the basis for the widely used `WindowRef` data type.

Availability

Available in OS X v10.0 and later.

Declared in

`Quickdraw.h`

xColorSpec

```
struct xColorSpec {
   short value;
   RGBColor rgb;
   short xalpha;
};
typedef struct xColorSpec xColorSpec;
   typedef xColorSpec * xColorSpecPtr;
```


Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

xCSpecArray

```
typedef xColorSpec xCSpecArray[1];
```

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawTypes.h`

Constants

chunky

```
enum {
   chunky = 0,
   chunkyPlanar = 1,
   planar = 2
};
```

Color Constants

```
enum {
   blackColor = 33,
   whiteColor = 30,
   redColor = 205,
   greenColor = 341,
   blueColor = 409,
   cyanColor = 273,
   magentaColor = 137,
   yellowColor = 69
};
```

Constants

`blackColor`

Represents black.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`whiteColor`

Represents white.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`redColor`

Represents red.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`greenColor`

Represents green.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`blueColor`

Represents blue.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`cyanColor`

Represents cyan.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`magentaColor`

Represents magenta.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`yellowColor`

Represents yellow.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

Discussion

These constants are used in the `color` parameter of the `ForeColor` (page 217) and `BackColor` (page 151) functions to specify one of the eight basic QuickDraw colors.

colorXorXFer

```
enum {
   colorXorXFer = 52,
   noiseXFer = 53,
   customXFer = 54
};
```

Cursor ID Constants

```
enum {
   sysPatListID = 0,
   iBeamCursor = 1,
   crossCursor = 2,
   plusCursor = 3,
   watchCursor = 4
};
```

Constants

`iBeamCursor`

The I-beam cursor; to select text

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`crossCursor`

The crosshairs cursor; to draw graphics

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`plusCursor`

The plus sign cursor; to select cells

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`watchCursor`

The wristwatch cursor; to indicate a short operation in progress

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

Discussion

When passing a value to the `Show_Cursor` function, use the `Cursors` data type to represent the kind of cursor to show.

cursorDoesAnimate

```
enum {
   cursorDoesAnimate = 1L << 0,
   cursorDoesHardware = 1L << 1,
   cursorDoesUnreadableScreenBits = 1L << 2
};
```

Device Attribute Constants

```
enum {
   interlacedDevice = 2,
   hwMirroredDevice = 4,
   roundedDevice = 5,
   hasAuxMenuBar = 6,
   burstDevice = 7,
   ext32Device = 8,
   ramInit = 10,
   mainScreen = 11,
   allInit = 12,
   screenDevice = 13,
   noDriver = 14,
   screenActive = 15,
   hiliteBit = 7,
   pHiliteBit = 0,
   defQDColors = 127,
   RGBDirect = 16,
   baseAddr32 = 4
};
```

Constants

`burstDevice`

If this bit is set to 1, the graphics device supports block transfer.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`ext32Device`

If this bit is set to 1, the graphics device must be used in 32-bit mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`ramInit`

If this bit is set to 1, the graphics device has been initialized from RAM.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`mainScreen`

If this bit is set to 1,the graphics device is the main screen.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`allInit`

If this bit is set to 1, all graphics devices were initialized from the `'scrn'` resource.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`screenDevice`

If this bit is set to 1, the graphics device is a screen.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`noDriver`

If this bit is set to 1, the `GDevice` structure has no driver.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`screenActive`

If this bit is set to 1, the graphics device is active.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

Discussion

These constants are used in the `attribute` parameters of the `SetDeviceAttribute` (page 406) and `TestDeviceAttribute` (page 448) functions, and in the `deviceFlags` parameter of the `DeviceLoopDrawingProcPtr` (page 64) callback. These constants represent the `GDevice` structure's attributes, as bits in the `gdFlags` field.

Device Loop Flags

```
enum {
   singleDevices = 1 << singleDevicesBit,
   dontMatchSeeds = 1 << dontMatchSeedsBit,
```



```
   allDevices = 1 << allDevicesBit
};
```

Constants

`singleDevices`

If this flag is not set, `DeviceLoop` calls your drawing function only once for each set of similar graphics devices, and the first one found is passed as the target device. (It is assumed to be representative of all the similar graphics devices.) If you set the `singleDevices` flag, then `DeviceLoop` does not group similar graphics devices, (that is, those having identical pixel depths, black-and-white or color settings, and matching color table seeds), when it calls your drawing function.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`dontMatchSeeds`

If you set the `dontMatchSeeds` flag, then `DeviceLoop` does not consider the `ctSeed` field of `ColorTable` structures for graphics devices when comparing them; `DeviceLoop` ignores this flag if you set the `singleDevices` flag.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`allDevices`

If you set the `allDevices` flag, `DeviceLoop` ignores the `drawingRgn` parameter and calls your drawing function for every device. The value of the current graphics port's `visRgn` field is not affected when you set this flag.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

Discussion

When you use the `DeviceLoop` (page 182) function, you can change its default behavior by using the `flags` parameter to specify one or more members of the set of flags defined by the `DeviceLoopFlags` data type. If you want to use the default behavior of `DeviceLoop`, specify 0 in the `flags` parameter.

deviceIsIndirect

```
enum {
   deviceIsIndirect = (1L << 0),
   deviceNeedsLock = (1L << 1),
   deviceIsStatic = (1L << 2),
   deviceIsExternalBuffer = (1L << 3),
   deviceIsDDSurface = (1L << 4),
   deviceIsDCISurface = (1L << 5),
   deviceIsGDISurface = (1L << 6),
```

```
   deviceIsAScreen = (1L << 7),
   deviceIsOverlaySurface = (1L << 8)
};
```

Drag Constraint Constants

When passed to the `DragControl` function, specify how a user can move a control.

```
enum {
   kNoConstraint = 0,
   kVerticalConstraint = 1,
   kHorizontalConstraint = 2
};
```

Constants

`kNoConstraint`

No constraint.

Available in OS X v10.0 and later.

Declared in `Quickdraw.h`.

`kVerticalConstraint`

Constrain movement to horizontal axis only.

Available in OS X v10.0 and later.

Declared in `Quickdraw.h`.

`kHorizontalConstraint`

Constrain movement to vertical axis only.

Available in OS X v10.0 and later.

Declared in `Quickdraw.h`.

Graphics Device Type Constants

```
enum {
   picLParen = 0,
   picRParen = 1,
   clutType = 0,
   fixedType = 1,
   directType = 2,
   gdDevType = 0
};
```

Constants

`clutType`

Represents a CLUT device--that is, one with colors mapped with a color lookup table.

Available in OS X v10.0 and later.

Declared in `IONDRVLibraries.h`.

`fixedType`

Represents a fixed colors device --that is, the color lookup table can't be changed.

Available in OS X v10.0 and later.

Declared in `IONDRVLibraries.h`.

`directType`

Represents a device with direct RGB colors.

Available in OS X v10.0 and later.

Declared in `IONDRVLibraries.h`.

`gdDevType`

If this bit is set to 0, the graphics device is black and white; if it is set to 1, the graphics device supports color.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

Discussion

These constants represent the general type of graphics device for the `gdType` field of the `GDevice` (page 91) structure.

Graphics World Flags

Specify additional information passed to and from `NewGWorld` (page 328) and related functions in parameters of type `GWorldFlags` (page 96).

```
enum {
   pixPurge = 1L << pixPurgeBit,
   noNewDevice = 1L << noNewDeviceBit,
   useTempMem = 1L << useTempMemBit,
   keepLocal = 1L << keepLocalBit,
   useDistantHdwrMem = 1L << useDistantHdwrMemBit,
   useLocalHdwrMem = 1L << useLocalHdwrMemBit,
   pixelsPurgeable = 1L << pixelsPurgeableBit,
   pixelsLocked = 1L << pixelsLockedBit,
   kNativeEndianPixMap = 1L << nativeEndianPixMapBit,
   kAllocDirectDrawSurface = 1L << 14,
   mapPix = 1L << mapPixBit,
```



```
    newDepth = 1L << newDepthBit,
    alignPix = 1L << alignPixBit,
    newRowBytes = 1L << newRowBytesBit,
    reallocPix = 1L << reallocPixBit,
    clipPix = 1L << clipPixBit,
    stretchPix = 1L << stretchPixBit,
    ditherPix = 1L << ditherPixBit,
    gwFlagErr = 1L << gwFlagErrBit
};
```

Constants

`pixPurge`

If you specify this flag for the `flags` parameter of the `NewGWorld` function, `UpdateGWorld` (page 454) makes the base address for the offscreen pixel image purgeable.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`noNewDevice`

If you specify this flag for the `flags` parameter of the `UpdateGWorld` (page 454) function, `NewGWorld` does not create a new offscreen `GDevice` structure; instead, `NewGWorld` uses either the `GDevice` structure you specify or the `GDevice` structure for a video card on the user's system.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`useTempMem`

If you specify this in the `flags` parameter of the `UpdateGWorld` (page 454) function, `NewGWorld` creates the base address for an offscreen pixel image in temporary memory. You generally should not use this flag. You should use temporary memory only for fleeting purposes and only with the `GetPixelsState` (page 250) function so that other applications can launch.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`keepLocal`

If you specify this in the `flags` parameter of the `UpdateGWorld` (page 454) function, `NewGWorld` creates a pixel image in Macintosh main memory where it cannot be cached to a graphics accelerator card.

If you specify this in the flags parameter of `GetPixelsState` (page 250), `UpdateGWorld` keeps the offscreen pixel image in Macintosh main memory.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`pixelsPurgeable`

If you specify this in the `state` parameter of the `UpdateGWorld` (page 454) function, `SetPixelsState` makes the base address for an offscreen pixel map purgeable. If you use the `SetPixelsState` function without passing it this flag, then `SetPixelsState` makes the base address for an offscreen pixel map unpurgeable. If the `GetPixelsState` (page 250) function returns this flag, then the base address for an offscreen pixel is purgeable.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`pixelsLocked`

If you specify this flag for the `state` parameter of the `SetPixelsState` function, `SetPixelsState` locks the base address for an offscreen pixel image. If you use the `SetPixelsState` function without passing it this flag, then `SetPixelsState` unlocks the offscreen pixel image. If the `GetPixelsState` function returns this flag, then the base address for an offscreen pixel is locked.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`kNativeEndianPixMap`

By default, the function `NewGWorld` (page 328) allocates pixel buffers with big-endian byte ordering regardless of the system architecture. If this flag is passed in the `flags` parameter of `NewGWorld`, the pixel format will be set to `k32ARGBPixelFormat` or `k16BE555PixelFormat` on a PowerPC system, and to `k32BGRAPixelFormat` or `k16LE555PixelFormat` on an Intel system, for depths 32 or 16, respectively. Note that `NewGWorld` is the only function where this flag is observed; `NewGWorldFromPtr` (page 333) and `UpdateGWorld` (page 454) ignore it.

Available in OS X v10.3 through OS X v10.6.

Declared in `QDOffscreen.h`.

`mapPix`

If the `UpdateGWorld` (page 454) function returns this flag, then it remapped the colors in the offscreen pixel map to a new color table.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`newDepth`

If the `UpdateGWorld` function returns this flag, then it translated the offscreen pixel map to a different pixel depth.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`alignPix`

If the `UpdateGWorld` function returns this flag, then it realigned the offscreen pixel image to an onscreen window.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`newRowBytes`

If the `UpdateGWorld` function returns this flag, then it changed the `rowBytes` field of the `PixMap` structure for the offscreen graphics world.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`reallocPix`

If the `UpdateGWorld` function returns this flag, then it reallocated the base address for the offscreen pixel image. Your application should then reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`clipPix`

If you specify this flag in the `flags` parameter of the `UpdateGWorld` (page 454) function, then `UpdateGWorld` updates and clips the pixel image to the new boundary rectangle specified. If the `UpdateGWorld` function returns this flag, then it clipped the pixel image.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`stretchPix`

If you specify this flag in the `flags` parameter of the `UpdateGWorld` (page 454) function, then `UpdateGWorld` updates and stretches or shrinks the pixel image to the new boundary rectangle specified. If the `UpdateGWorld` function returns this flag, then it stretched or shrank the offscreen image.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`ditherPix`

If you specify this flag in the `flags` parameter of the `UpdateGWorld` (page 454) function, then `UpdateGWorld` dithers the pixel image to the new boundary rectangle specified. Include this flag with the `clipPix` or `stretchPix` flag. If the `UpdateGWorld` function returns this flag, then it dithered the offscreen image.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

`gwFlagErr`

If the `UpdateGWorld` function returns this flag, then it was unsuccessful and the offscreen graphics world was left unchanged.

Available in OS X v10.0 through OS X v10.6.

Declared in `QDOffscreen.h`.

invalColReq

```
enum {
   invalColReq = -1
};
```

italicBit

```
enum {
   italicBit = 1,
   ulineBit = 2,
   outlineBit = 3,
   shadowBit = 4,
   condenseBit = 5,
   extendBit = 6
};
```

Pixel Formats

```
enum {
   k16LE555PixelFormat = 'L555',
   k16LE5551PixelFormat = '5551',
   k16BE565PixelFormat = 'B565',
   k16LE565PixelFormat = 'L565',
   k24BGRPixelFormat = '24BG',
   k32BGRAPixelFormat = 'BGRA',
   k32ABGRPixelFormat = 'ABGR',
   k32RGBAPixelFormat = 'RGBA',
   kYUVSPixelFormat = 'yuvs',
   kYUVUPixelFormat = 'yuvu',
   kYVU9PixelFormat = 'YVU9',
   kYUV411PixelFormat = 'Y411',
   kYVYU422PixelFormat = 'YVYU',
   kUYVY422PixelFormat = 'UYVY',
   kYUV211PixelFormat = 'Y211',
```

```
   k2vuyPixelFormat = '2vuy'
};
```

k1MonochromePixelFormat

```
enum {
   k1MonochromePixelFormat = 0x00000001,
   k2IndexedPixelFormat = 0x00000002,
   k4IndexedPixelFormat = 0x00000004,
   k8IndexedPixelFormat = 0x00000008,
   k16BE555PixelFormat = 0x00000010,
   k24RGBPixelFormat = 0x00000018,
   k32ARGBPixelFormat = 0x00000020,
   k1IndexedGrayPixelFormat = 0x00000021,
   k2IndexedGrayPixelFormat = 0x00000022,
   k4IndexedGrayPixelFormat = 0x00000024,
   k8IndexedGrayPixelFormat = 0x00000028
};
```

kCursorComponentInit

```
enum {
   kCursorComponentInit = 0x0001,
   kCursorComponentGetInfo = 0x0002,
   kCursorComponentSetOutputMode = 0x0003,
   kCursorComponentSetData = 0x0004,
   kCursorComponentReconfigure = 0x0005,
   kCursorComponentDraw = 0x0006,
   kCursorComponentErase = 0x0007,
   kCursorComponentMove = 0x0008,
   kCursorComponentAnimate = 0x0009,
   kCursorComponentLastReserved = 0x0050
};
```

kCursorComponentsVersion

```
enum {
   kCursorComponentsVersion = 0x00010001
};
```


kCursorComponentType

```
enum {
   kCursorComponentType = 'curs'
};
```

kCursorImageMajorVersion

```
enum {
   kCursorImageMajorVersion = 0x0001,
   kCursorImageMinorVersion = 0x0000
};
```

kPrinterFontStatus

```
enum {
   kPrinterFontStatus = 0,
   kPrinterScalingStatus = 1
};
```

kQDGrafVerbFrame

```
enum {
   kQDGrafVerbFrame = 0,
   kQDGrafVerbPaint = 1,
   kQDGrafVerbErase = 2,
   kQDGrafVerbInvert = 3,
   kQDGrafVerbFill = 4
};
```

kQDParseRegionFromTop

```
enum {
   kQDParseRegionFromTop = (1 << 0),
   kQDParseRegionFromBottom = (1 << 1),
   kQDParseRegionFromLeft = (1 << 2),
   kQDParseRegionFromRight = (1 << 3),
   kQDParseRegionFromTopLeft = kQDParseRegionFromTop | kQDParseRegionFromLeft,
```

```
   kQDParseRegionFromBottomRight = kQDParseRegionFromBottom |  kQDParseRegionFromRight
};
```

kQDRegionToRectsMsgInit

```
enum {
   kQDRegionToRectsMsgInit = 1,
   kQDRegionToRectsMsgParse = 2,
   kQDRegionToRectsMsgTerminate = 3
};
```

kQDUseDefaultTextRendering

```
enum {
   kQDUseDefaultTextRendering = 0,
   kQDUseTrueTypeScalerGlyphs = (1 << 0),
   kQDUseCGTextRendering = (1 << 1),
   kQDUseCGTextMetrics = (1 << 2),
   kQDSupportedFlags = kQDUseTrueTypeScalerGlyphs | kQDUseCGTextRendering  |
kQDUseCGTextMetrics,
   kQDDontChangeFlags = 0xFFFFFFFF
};
```

kRenderCursorInHardware

```
enum {
   kRenderCursorInHardware = 1L << 0,
   kRenderCursorInSoftware = 1L << 1
};
```

kXFer1PixelAtATime

```
enum {
   kXFer1PixelAtATime = 0x00000001,
   kXFerConvertPixelToRGB32 = 0x00000002
};
```


normalBit

```
enum {
   normalBit = 0,
   inverseBit = 1,
   redBit = 4,
   greenBit = 3,
   blueBit = 2,
   cyanBit = 8,
   magentaBit = 7,
   yellowBit = 6,
   blackBit = 5
};
```

pixPurgeBit

```
enum {
   pixPurgeBit = 0,
   noNewDeviceBit = 1,
   useTempMemBit = 2,
   keepLocalBit = 3,
   useDistantHdwrMemBit = 4,
   useLocalHdwrMemBit = 5,
   pixelsPurgeableBit = 6,
   pixelsLockedBit = 7,
   nativeEndianPixMapBit = 8,
   mapPixBit = 16,
   newDepthBit = 17,
   alignPixBit = 18,
   newRowBytesBit = 19,
   reallocPixBit = 20,
   clipPixBit = 28,
   stretchPixBit = 29,
   ditherPixBit = 30,
   gwFlagErrBit = 31
};
```

singleDevicesBit

```
enum {
   singleDevicesBit = 0,
   dontMatchSeedsBit = 1,
   allDevicesBit = 2
};
```

Source, Pattern, and Arithmetic Transfer Mode Constants

```
enum {
   srcCopy = 0,
   srcOr = 1,
   srcXor = 2,
   srcBic = 3,
   notSrcCopy = 4,
   notSrcOr = 5,
   notSrcXor = 6,
   notSrcBic = 7,
   patCopy = 8,
   patOr = 9,
   patXor = 10,
   patBic = 11,
   notPatCopy = 12,
   notPatOr = 13,
   notPatXor = 14,
   notPatBic = 15,
   grayishTextOr = 49,
   hilitetransfermode = 50,
   hilite = 50,
   blend = 32,
   addPin = 33,
   addOver = 34,
   subPin = 35,
   addMax = 37,
   adMax = 37,
   subOver = 38,
   adMin = 39,
   ditherCopy = 64,
   transparent = 36
};
```

Constants

`srcCopy`

For basic graphics ports, force the destination pixel black where the source pixel is black; where the source pixel is white, force the destination pixel white.

For color graphics ports, determines how close the color of the source pixel is to black, and assigns this relative amount of foreground color to the destination pixel. Determines how close the color of the source pixel is to white, and assigns this relative amount of background color to the destination pixel.

Available in OS X v10.0 and later.

Declared in `Quickdraw.h`.

`srcOr`

For basic graphics ports, forces the destination pixel black if the source pixel is black; where the source pixel is white, leaves the destination pixel unaltered.

For color graphics ports, determines how close the color of the source pixel is to black, and assigns this relative amount of foreground color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`srcXor`

For basic and color graphics ports, inverts destination pixel where the source pixel is black. For a basic graphics port, where the source pixel is white, leaves the destination pixel unaltered.

For a color graphics port, for a colored destination pixel, uses the complement of its color if the pixel is direct, or inverts its index if the pixel is indexed.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`srcBic`

For a basic graphics port, forces destination pixel white where source pixel is black; where source pixel is white, leaves the destination pixel unaltered.

For a color graphics port, determines how close the color of the source pixel is to black, and assigns this relative amount of background color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notSrcCopy`

For a basic graphics port, forces the destination pixel white where the source pixel is black; where the source pixel is white, forces the destination pixel black.

For a color graphics port, determines how close the color of the source pixel is to black, and assigns this relative amount of background color to the destination pixel. Determines how close the color of the source pixel is to white, and assigns this relative amount of foreground color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notSrcOr`

For a basic graphics port, leaves the destination pixel unaltered where the source pixel is black; where the source pixel is white, forces the destination pixel black.

For a color graphics port, determines how close the color of the source pixel is to white, and assigns this relative amount of foreground color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notSrcXor`

For basic and color graphics ports, where the source pixel is white, inverts the destination pixel. For a basic graphics port, where the source pixel is black, leaves the destination pixel unaltered.

For a color graphics port, for a colored destination pixel, uses the complement of its color if the pixel is direct, or inverts its index if the pixel is indexed.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notSrcBic`

For a basic graphics port, where the source pixel is black, leaves the destination pixel unaltered; where the source pixel is white, forces the destination pixel white.

For a color graphics port, determines how close the color of the source pixel is to white, and assigns this relative amount of background color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`patCopy`

Where the pattern pixel is black, applies foreground color to the destination pixel; where the pattern pixel is white, applies background color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`patOr`

Where the pattern pixel is black, inverts the destination pixel; where the pattern pixel is white, leaves the destination pixel unaltered.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`patXor`

Where the pattern pixel is black, inverts the destination pixel; where the pattern pixel is white, leaves the destination pixel unaltered.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`patBic`

Where the pattern pixel is black, applies the background color to destination pixel; where the pattern pixel is white, leaves the destination pixel unaltered.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notPatCopy`

Where the pattern pixel is black, applies background color to destination pixel; where the pattern pixel is white, applies foreground color to the destination pixel

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notPatOr`

Where the pattern pixel is black, leaves the destination pixel unaltered; where the pattern pixel is white, applies foreground color to the destination pixel

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notPatXor`

Where the pattern pixel is black, leaves the destination pixel unaltered; where the pattern pixel is white, inverts the destination pixel

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`notPatBic`

Where the pattern pixel is black, leaves the destination pixel unaltered; where the pattern pixel is white, applies background color to the destination pixel.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`grayishTextOr`

Draws dimmed text on the screen. You can use it for black-and-white or color graphics ports. The `grayishTextOr` transfer mode is not considered a standard transfer mode because currently it is not stored in pictures, and printing with it is undefined. (It does not pass through the QuickDraw bottleneck functions.)

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`hilite`

Adds highlighting to the source or pattern mode. With highlighting, QuickDraw replaces the background color with the highlight color when your application copies images between graphics ports. This has the visual effect of using a highlighting pen to select the object. (The global variable `HiliteRGB` is read from parameter RAM when the machine starts. Basic graphics ports use the color stored in the `HiliteRGB` global variable as the highlight color. Color graphics ports default to the `HiliteRGB` global variable, but can be overridden by the `HiliteColor` function.)

For text, specifies that the caret position should be determined according to the primary line direction, based on the value of `SysDirection`.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`blend`

Replaces the destination pixel with a blend of the source and destination pixel colors. If the destination is a bitmap or 1-bit pixel map, reverts to `srcCopy` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`addPin`

Replaces the destination pixel with the sum of the source and destination pixel colors-- up to a maximum allowable values. If the destination is a bitmap or 1-bit pixel map, reverts to `srcBic` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`addOver`

Replaces the destination pixel with the sum of the source and destination pixel colors, except if the value of the red, green, or blue component exceeds 65,536, then `addOver` subtracts 65,536 from that value. If the destination is a bitmap or 1-bit pixel map, reverts to `srcXor` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`subPin`

Replaces the destination pixel with the difference of the source and destination pixel colors, but not less than a minimum allowable value. If the destination is a bitmap or 1-bit pixel map, reverts to `srcOr` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`addMax`

Compares the source and destination pixels, and replaces the destination pixel with the color containing the greater saturation of each of the RGB components. If the destination is a bitmap or 1-bit pixel map, reverts to `srcBic` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`subOver`

Replaces the destination pixel with the difference of the source and destination pixel colors, except if the value of the red, green, or blue component is less than 0, then it adds the negative result to 65,536. if the destination is a bitmap or 1-bit pixel map, revert to `srcXor` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`adMin`

Compares the source and destination pixels, and replaces the destination pixel with the color containing the lesser saturation of each of the RGB components. If the destination is a bitmap or 1-bit pixel map, reverts to `srcOr` mode.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`ditherCopy`

On computers running System 7, you can add dithering to any source mode by adding this constant or the value it represents to the source mode.

Dithering is a technique that mixes existing colors to create the effect of additional colors. It also improves images that you shrink or that you copy from a direct pixel device to an indexed device.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

`transparent`

Replaces the destination pixel with the source pixel if the source pixel is not equal to the background color. The `transparent` mode replaces the destination pixel with the source pixel if the source pixel isn't equal to the background color. This mode is most useful in 8-bit, 4-bit, or 2-bit color modes.

Available in OS X v10.0 through OS X v10.6.

Declared in `QuickdrawTypes.h`.

Discussion

`CopyBits` (page 165) uses the source and arithmetic transfer mode constants in the `mode` parameter to specify the manner in which pixels are transferred from a source pixel map to a destination pixel map.

`PenMode` (page 360) uses the pattern mode constants in the `mode` parameter to specify source modes for transferring the bits from a source bitmap to a destination bitmap.

The `TextMode` function uses these constants to set the transfer mode in the graphics port `txMode` field.

The transfer mode determines the interplay between what an application is drawing (the source) and what already exists on the display device (the destination), resulting in the text display.

There are two basic kinds of modes: pattern (`pat`) and source (`src`).

The pattern mode constants are `patCopy`, `patOr`, `patXor`, `patBic`, `notPatCopy`, `notPatOr`, `notPatXor`, and `notPatBic`.

Source is the kind that you use for drawing text. There are four basic Boolean operations: `Copy`, `Or`, `Xor`, and `Bic` (bit clear), each of which has an inverse variant in which the source is inverted before the transfer, yielding eight operations in all. Basic QuickDraw supports these eight transfer modes. Color QuickDraw interprets the source mode constants differently than basic QuickDraw does. Color QuickDraw enables your application to

achieve color effects within those basic transfer modes, and offers an additional set of transfer modes that perform arithmetic operations on the RGB values of the source and destination pixels. Other transfer modes are `grayishTextOr`, `transparent` mode, and text mask mode.

The arithmetic transfer modes are `addOver`, `addPin`, `subOver`, `subPin`, `addMax`, `adMax`, `adMin`, and `blend`. For color, the arithmetic modes change the destination pixels by performing arithmetic operations on the source and destination pixels. Arithmetic transfer modes calculate pixel values by adding, subtracting, or averaging the RGB components of the source and destination pixels. They are most useful for 8-bit color, but they work on 4-bit and 2-bit color also. When the destination bitmap is one bit deep, the mode reverts to the basic transfer mode that best approximates the arithmetic mode requested.

Verb Constants

```
enum {
   frame = kQDGrafVerbFrame,
   paint = kQDGrafVerbPaint,
   erase = kQDGrafVerbErase,
   invert = kQDGrafVerbInvert,
   fill = kQDGrafVerbFill
};
```

Constants

`frame`
Specifies the frame action.

`paint`
Specifies the paint action.

`erase`
Specifies the erase action.

`invert`
Specifies the invert action.

`fill`
Specifies the fill action.

Discussion

When you use the `StdRect` (page 442) , `StdRRect` (page 444) , `StdOval` (page 440) , `StdArc` (page 435) , `StdPoly` (page 441) , or `StdRgn` (page 443) functions, these constants are used in the `verb` parameter to specify the type of action taken by those low-level drawing functions.

Result Codes

The table below lists the result codes specific to QuickDraw.

Result Code	Value	Description
updPixMemErr	-125	Insufficient memory to update a pixmap. Available in OS X v10.0 and later.
noMemForPictPlaybackErr	-145	Insufficient memory for drawing the picture. Available in OS X v10.0 and later.
pixMapTooDeepErr	-148	Pixel map is deeper than 1 bit per pixel. Available in OS X v10.0 and later.
nsStackErr	-149	Insufficient stack Available in OS X v10.0 and later.
cMatchErr	-150	Color2Index failed to find an index. Available in OS X v10.0 and later.
cTempMemErr	-151	Failed to allocate memory for temporary structures. Available in OS X v10.0 and later.
cNoMemErr	-152	Failed to allocate memory for structures. Available in OS X v10.0 and later.
cRangeErr	-153	Range error on color table requests. Available in OS X v10.0 and later.
cProtectErr	-154	ColorTable structure entry protection violation. Available in OS X v10.0 and later.
cDevErr	-155	Invalid type of graphics device. Available in OS X v10.0 and later.
cResErr	-156	Invalid resolution for MakeITable. Available in OS X v10.0 and later.
cDepthErr	-157	Invalid pixel depth. Available in OS X v10.0 and later.
rgnTooBigErr	-500	Bitmap too large to convert to a region. Available in OS X v10.0 and later.

Result Code	Value	Description
kQDNoPalette	-3950	Available in OS X v10.2 and later.
kQDNoColorHWCursorSupport	-3951	Available in OS X v10.2 and later.
kQDCursorAlreadyRegistered	-3952	Available in OS X v10.2 and later.
kQDCursorNotRegistered	-3953	Available in OS X v10.2 and later.
kQDCorruptPICTDataErr	-3954	Available in OS X v10.2 and later.

Deprecated QuickDraw Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Available in OS X v10.0 through OS X v10.6

AddComp

Adds a function to the head of the current device data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void AddComp (
   ColorComplementUPP compProc
);
```

Parameters

`compProc`

A pointer to your complement function, `ColorComplementProcPtr` (page 62).

Discussion

`AddComp` creates and allocates a `CProcRec` (page 83) data structure.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

AddPt

Adds the coordinates of two points. (Available in OS X v10.0 through OS X v10.6.)


```
void AddPt (
   Point src,
   Point *dst
);
```

Parameters

`src`

A point, the coordinates of which are to be added to the point in the `dstPt` parameter.

`dst`

A pointer to a point, the coordinates of which are to be added to the point in the `srcPt` parameter. On return, this value contains the result of adding the coordinates of the points you supplied in the `srcPt` and `dstPt` parameters.

Discussion

The `AddPt` function adds the coordinates of the point specified in the `srcPt` parameter to the coordinates of the point specified in the `dstPt` parameter, and returns the result in the `dstPt` parameter.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

CarbonSketch

Declared in

`QuickdrawAPI.h`

AddSearch

Adds a function to the head of the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void AddSearch (
   ColorSearchUPP searchProc
);
```

Parameters

`searchProc`

A pointer to your custom search function, `ColorSearchProcPtr` (page 62).

Discussion

`AddSearch` creates and allocates an `SProcRec` (page 120) data structure.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

AllocCursor

Reallocates cursor memory. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void AllocCursor (
   void
);
```

Discussion

Under normal circumstances, you should never need to use this function, since Color QuickDraw handles reallocation of cursor memory.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

AllowPurgePixels

Makes the base address for an offscreen pixel image purgeable. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void AllowPurgePixels (
```



```
   PixMapHandle pm
);
```

Parameters

`pm`

A handle to an offscreen pixel map.

Discussion

The `AllowPurgePixels` function allows the Memory Manager to free the memory it occupies if available memory space becomes low. By default, `NewGWorld` creates an unpurgeable base address for an offscreen pixel image.

To get a handle to an offscreen pixel map, first use the `GetGWorldPixMap` (page 238) function. Then supply this handle for the `pm` parameter of `AllowPurgePixels`.

Your application should call the `LockPixels` (page 314) function before drawing into or copying from an offscreen pixel map. If the Memory Manager has purged the base address for its pixel image, `LockPixels` returns `FALSE`. In that case either your application should use the `UpdateGWorld` (page 454) function to begin reconstructing the offscreen pixel image, or it should draw directly to an onscreen graphics port.

Only unlocked memory blocks can be made purgeable. If you use `LockPixels`, you must use the `UnlockPixels` function before calling `AllowPurgePixels`.

Special Considerations

The `AllowPurgePixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QDOffscreen.h`

AngleFromSlope

Converts a slope value to an angle value. (Available in OS X v10.0 through OS X v10.6.)

```
short AngleFromSlope (
```



```
   Fixed slope
);
```

Parameters

`slope`

The slope, defined as Dx/Dy, which is the horizontal change divided by the vertical change between any two points on a line with the slope.

Return Value

The angle corresponding to the slope specified in the `slope` parameter treated `MOD` 180. Angles are defined in clockwise degrees from 12 o'clock. The negative y-axis is defined as being at 12 o'clock, and the positive y-axis at 6 o'clock. The x-axis is defined as usual, with the positive side defined as being at 3 o'clock.

Special Considerations

The `AngleFromSlope` function is most useful when you require speed more than accuracy in performing the calculation. The integer result is within 1 degree of the correct answer, but not necessarily within half a degree.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

BackColor

Changes a basic graphics port's background color. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void BackColor (
   long color
);
```

Parameters

`color`

One of eight color values. See "Color Constants" (page 122).

Discussion

The background color is the color of the pixels in the bitmap wherever no drawing has taken place. By default the background color of a `GrafPort` is white.

The `BackColor` function sets the background color for the current graphics port to the color that you specify in the `color` parameter. When you draw with the `patCopy` and `srcCopy` transfer modes, for example, white pixels are drawn in the color you specify with `BackColor`.

All nonwhite colors appear as black on black-and-white screens. Before you use `BackColor`, use the `DeviceLoop` function to determine the color characteristics of the current screen.

Special Considerations
The `BackColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Version Notes
In System 7, use the Color QuickDraw function `RGBBackColor`.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DrawSprocketTestOld
GlyphaIVOld
MovieBrowser
QTMusicToo
SGDataProcSample

Declared in
`QuickdrawAPI.h`

BackPat

Changes the bit pattern used as the background pattern by the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void BackPat (
   const Pattern *pat
);
```


Parameters

`pat`

A bit pattern, as defined by a `Pattern` (page 100) structure.

Discussion

The `BackPat` function sets the bit pattern defined in the `Pattern` structure, which you specify in the `pat` parameter, to be the background pattern. (The standard bit patterns `white`, `black`, `gray`, `ltGray`, and `dkGray` are predefined; the initial background pattern for the graphics port is `white`.) This pattern is stored in the `bkPat` field of a `GrafPort` structure.

The `BackPat` function also sets a bit pattern for the background color in a color graphics port. The `BackPat` function creates a handle, of type `PixPatHandle`, for the bit pattern and stores this handle in the `bkPixPat` field of the `CGrafPort` structure. As in basic graphics ports, Color QuickDraw draws patterns in color graphics ports at the time of drawing, not at the time you use `PenPat` to set the pattern.

To define your own patterns, you typically create pattern, 'PAT', or pattern list, 'PAT#', resources.

Special Considerations

The `BackPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial

ControlBackground

Declared in

`QuickdrawAPI.h`

BackPixPat

Assigns a pixel pattern as the background pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void BackPixPat (
   PixPatHandle pp
);
```


Parameters

`pp`

A handle to the pixel pattern to use as the background pattern.

Discussion

Setting the background pattern allows the `ScrollRect` function and the shape-erasing functions (for example, `EraseRect`) to fill the background with a colored patterned "ink."

The `BackPixPat` function is similar to the basic QuickDraw function `BackPat`, except that you pass `BackPixPat` a handle to a multicolored pixel pattern instead of a bit pattern.

The handle to the pixel pattern is stored in the `bkPixPat` field of the `CGrafPort` structure, therefore, you should not dispose of this handle since QuickDraw removes all references to your pattern from an existing graphics port when you dispose of it.

If you use `BackPixPat` to set a background pixel pattern in a basic graphics port, the data in the `pat1Data` field of the `PixPat` (page 106) structure is placed into the `bkPat` field of the `GrafPort` structure.

To define your own pixel pattern, create a pixel pattern resource, x is described on `'ppat'`, or use the `NewPixPat` (page 334) function. To set the background pattern to a bit pattern, you can also use the QuickDraw function, `BackPat`.

Special Considerations

The `BackPixPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

BitMapToRegion

Converts a bitmap or pixel map to a region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
OSErr BitMapToRegion (
```



```
   RgnHandle region,
   const BitMap *bMap
);
```

Parameters

`region`

A handle to a region to hold the converted `BitMap` or `PixMap` structure. This must be a valid region handle created with the `NewRgn` function. The old region contents are lost.

`bMap`

A pointer to a `BitMap` or `PixMap` structure to be converted. If you supply a `PixMap` structure, its pixel depth must be 1.

Return Value

A result code.

Discussion

The `BitMapToRegion` function converts a given `BitMap` or `PixMap` structure to a region. Pixels are added to the region where the corresponding entries in the bitmap have a value of 1. You would generally use this region later for drawing operations.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CopyBits vs. CopyMask

CustomWindow

qtskins

StarMenu

Declared in

`QuickdrawAPI.h`

CalcCMask

Determines where filling will not occur when filling from the outside of a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void CalcCMask (
   const BitMap *srcBits,
```



```
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *dstRect,
   const RGBColor *seedRGB,
   ColorSearchUPP matchProc,
   long matchData
);
```

Parameters

`srcBits`

The source image. If the image is in a pixel map, you must coerce its `PixMap` structure to a `BitMap` structure.

`dstBits`

The destination image. The `CalcCMask` function returns the generated bitmap mask in this parameter. You can then use this mask with the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions.

`srcRect`

The rectangle of the source image.

`dstRect`

The rectangle of the destination image.

`seedRGB`

An `RGBColor` structure specifying the color for pixels that should not be filled.

`matchProc`

An optional matching function.

`matchData`

Data for the optional matching function.

Discussion

Specify a source image in the `srcBits` parameter and in the `srcRect` parameter, specify a rectangle within that source image. Starting from the edges of this rectangle, `CalcCMask` calculates which pixels cannot be filled. By default, `CalcCMask` returns 1's in the mask to indicate which pixels have the exact color that you specify in the `seedRGB` parameter and which pixels are enclosed by shapes whose outlines consist entirely of pixels with this color.

For instance, if the source image in `srcBits` contains a dark blue rectangle on a red background, and your application sets `seedRGB` equal to dark blue, then `CalcCMask` returns a mask with 1's in the positions corresponding to the edges and interior of the rectangle, and the 0's outside of the rectangle.

If you set the `matchProc` and `matchData` parameters to 0, `CalcCMask` uses the exact color specified in the `RGBColor` structure that you supply in the `seedRGB` parameter. You can customize `CalcCMask` by writing your own color search function and pointing to it in the `matchProc` parameter. As with `SeedCFill`, you can then use the `matchData` parameter in any manner useful for your application.

The `CalcCMask` function does not scale so the source and destination rectangles must be the same size. Calls to `CalcCMask` are not clipped to the current port and are not stored into QuickDraw pictures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

CalcMask

Determines where filling will not occur when filling from the outside of a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void CalcMask (
   const void *srcPtr,
   void *dstPtr,
   short srcRow,
   short dstRow,
   short height,
   short words
);
```

Parameters

`srcPtr`
: A pointer to the source bit image.

`dstPtr`
: A pointer to the destination bit image.

`srcRow`
: Row width of the source bitmap.

`dstRow`
: Row width of the destination bitmap.

`height`
Height (in pixels) of the fill rectangle.

`words`
Width (in words) of the fill rectangle.

Discussion
The `CalcMask` function produces a bit image with 1's in all pixels to which paint could not flow from any of the outer edges of the rectangle. Use this bit image as a mask with the `CopyBits` or `CopyMask` function. A hollow object produces a solid mask, but an open object produces a mask of itself.

As with the `SeedFill` function, point to the bit image you want to fill with the `srcPtr` parameter, which can point to the image's base address or a word boundary within the image. Specify a pixel height and word width with the `height` and `words` parameters to define a fill rectangle that delimits the area you want to fill. The fill rectangle can be the entire bit image or a subset of it. Point to a destination image with the `dstPtr` parameter. Specify the row widths of the source and destination bitmaps (their `rowBytes` values) with the `srcRow` and `dstRow` parameters. (The bitmaps can be different sizes, but they must be large enough to contain the fill rectangle at the origins specified by `srcPtr` and `dstPtr`.)

Calls to `CalcMask` are not clipped to the current port and are not stored into QuickDraw pictures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

ClipCGContextToRegion

Sets the clipping path in a Quartz 2D graphics context, using a clipping region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
OSStatus ClipCGContextToRegion (
   CGContextRef gc,
   const Rect *portRect,
   RgnHandle region
);
```


Parameters

`context`

A Quartz context associated with a graphics port. You can obtain such a context by calling `QDBeginCGContext` (page 59).

`portRect`

The `portRect` for the graphics port associated with the context.

`region`

A region that represents the desired clipping path.

Return Value

A result code. If `noErr`, the clipping path is now the region-based path.

Discussion

This function sets the clipping path in the specified context to closely approximate the geometry of the specified region.

Unlike clipping in Quartz 2D, this function does not intersect the new region-based path with the current clipping path—the new path simply replaces the current clipping path.

You should use this function only when absolutely necessary—it's relatively inefficient when compared to Quartz 2D clipping functions such as `CGContextClipToRect`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CTMClip

Declared in

`QuickdrawAPI.h`

ClipRect

Changes the clipping region of the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void ClipRect (
```



```
   const Rect *r
);
```

Parameters

`r`

A pointer to a rectangle for the boundary of the new clipping region. The `ClipRect` function changes the clipping region of the current graphics port to a region that's equivalent to this rectangle. `ClipRect` doesn't change the region handle, but it affects the clipping region itself.

Discussion

Since `ClipRect` makes a copy of the given rectangle, any subsequent changes you make to that rectangle do not affect the clipping region of the port.

The `ClipRect` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

GlyphaIVOld
HTMLSample
MovieBrowser
QTMusicToo
qtspritesplus

Declared in

`QuickdrawAPI.h`

CloseCursorComponent

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSErr CloseCursorComponent (
   ComponentInstance ci
);
```


Return Value
A result code.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

ClosePicture

Completes the collection of drawing commands and picture comments that define your picture. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void ClosePicture (
   void
);
```

Discussion
The `ClosePicture` function stops collecting drawing commands and picture comments for the currently open picture. You should perform one and only one call to `ClosePicture` for every call to the `OpenCPicture` (or `OpenPicture`) function.

The `ClosePicture` function calls the `ShowPen` function, balancing the call made by `OpenCPicture` (or `OpenPicture`) to the `HidePen` function.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
MovieBrowser
qdmediamaker.win
qteffects.win
QTMusicToo
qtspritesplus

Declared in
`QuickdrawAPI.h`

ClosePoly

Completes the collection of lines that defines a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void ClosePoly (
   void
);
```

Discussion
The `ClosePoly` function stops collecting line-drawing commands for the currently open polygon and computes the `polyBBox` field of the `Polygon` (page 108) structure. You should call `ClosePoly` only once for every call to the `OpenPoly` function.

The `ClosePoly` function uses the `ShowPen` function, balancing the call to the `HidePen` function made by the `OpenPoly` function.

Special Considerations
The `ClosePoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
MouseTracking

Declared in
`QuickdrawAPI.h`

CloseRgn

Organizes a collection of lines and shapes into a region definition. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void CloseRgn (
   RgnHandle dstRgn
);
```

Parameters

`dstRgn`

The handle to the region to close. This handle should be a region handle returned by the `NewRgn` (page 336) function.

Discussion

The `CloseRgn` function stops the collection of lines and framed shapes, organizes them into a region definition, and saves the result in the region whose handle you pass in the `dstRgn` parameter.

The `CloseRgn` function does not create the destination region; you must have already allocated space for it by using the `OpenRgn` function. The `CloseRgn` function calls the `ShowPen` function, balancing the call to the `HidePen` function made by `OpenRgn`.

When you no longer need the memory occupied by the region, use the `DisposeRgn` (page 190) function.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

Regions are limited to 32 KB in size in basic QuickDraw and 64 KB in Color QuickDraw. When you structure drawing operations in an open region, the resulting region description may overflow this limit. Should this happen in Color QuickDraw, the `QDError` function returns the result code `regionTooBigError`. Since the resulting region is potentially corrupt, the `CloseRgn` function returns an empty region if it detects `QDError` has returned `regionTooBigError`.

The `CloseRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CTMClip

kcapApp

QTMusicToo

Declared in
`QuickdrawAPI.h`

Color2Index

Obtains the index of the best available approximation for a given color in the color table of the current `GDevice` data structure. This function is used only by system software. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
long Color2Index (
   const RGBColor *myColor
);
```

Parameters
`myColor`

A pointer to the RGB color value to be approximated.

Return Value
The index of the best approximation for the given color that is available in the color table of the current `GDevice` data structure. Note that `Color2Index` returns a long integer, in which the low-order word is the index value; the high-order word contains zeros.

Discussion
You should not call `Color2Index` from within a custom search function (described in `ColorSearchProcPtr` (page 62)).

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

ColorBit

Sets the foreground color for all printing in the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void ColorBit (
   short whichBit
);
```

Parameters

`whichBit`

An integer specifying the plane to draw into.

Discussion

The `ColorBit` function is called by printing software for a color printer (or other color-imaging software) to set the `GrafPort` structure's `colorBit` field to the value in the `whichBit` parameter. This value tells QuickDraw which plane of the color picture to draw into. QuickDraw draws into the plane corresponding to the bit number specified by the `whichBit` parameter. Since QuickDraw can support output devices that have up to 32 bits of color information per pixel, the possible range of values for `whichBit` is 0 through 31. The initial value of the `colorBit` field is 0.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

CopyBits

Copies a portion of a bitmap or a pixel map from one graphics port or offscreen graphics world into another graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void CopyBits (
   const BitMap *srcBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
```



```
   RgnHandle maskRgn
);
```

Parameters

`srcBits`

The source `BitMap` structure.

`dstBits`

The destination `BitMap` structure.

`srcRect`

The source rectangle.

`dstRect`

The destination rectangle.

`mode`

One of the eight source modes in which the copy is to be performed. See "Source, Pattern, and Arithmetic Transfer Mode Constants" (page 138). The `CopyBits` function always dithers images when shrinking them between pixel maps on direct devices.

When transferring pixels from a source pixel map to a destination pixel map, color QuickDraw interprets the source mode constants differently than basic QuickDraw does.

When you use `CopyBits` on a computer running color QuickDraw, you can also specify one of the transfer modes in the `mode` parameter.

`maskRgn`

A region to use as a clipping mask. You can pass a region handle to specify a mask region the resulting image is always clipped to this mask region and to the boundary rectangle of the destination bitmap. If the destination bitmap is the current graphics port's bitmap, it is also clipped to the intersection of the graphics port's clipping region and visible region. If you do not want to clip to a masking region, just pass `NULL` for this parameter.

Discussion

The `CopyBits` function transfers any portion of a bitmap between two basic graphics ports, or any portion of a pixel map between two color graphics ports. Use `CopyBits` to move offscreen graphic images into an onscreen window, to blend colors for the image in a pixel map, and to shrink and expand images.

Specify a source bitmap in the `srcBits` parameter and a destination bitmap in the `dstBits` parameter. When copying images between color graphics ports, you must coerce each `CGrafPort` structure to a `GrafPort` structure, dereference the `portBits` fields of each, and then pass these "bitmaps" in the `srcBits` and `dstBits` parameters. If your application copies a pixel image from a color graphics port called `MyColorPort`, for example, you could specify (* `GrafPtr(MyColorPort)).portBits` in the `srcBits` parameter. In a

CGrafPort structure, the high 2 bits of the portVersion field are set. This field, which shares the same position in a CGrafPort structure as the portBits.rowBytes field in a GrafPort structure, indicates to CopyBits that you have passed it a handle to a pixel map rather than a bitmap.

Using the srcRect and dstRect parameters, you can specify identically or differently sized source and destination rectangles; for differently sized rectangles, CopyBits scales the source image to fit the destination. If the bit image is a circle in a square source rectangle, and the destination rectangle is not square, the bit image appears as an oval in the destination. When you specify rectangles in the srcRect and dstRect parameters, use the local coordinate systems of, respectively, the source and destination graphics ports.

The CopyDeepMask (page 168) function combines the functions of the CopyBits and CopyMask functions.

Special Considerations

When you use the CopyBits function to transfer an image between pixel maps, the source and destination images may be of different pixel depths, of different sizes, and they may have different color tables. However, CopyBits assumes that the destination pixel map uses the same color table as the color table for the current GDevice structure. (This is because the Color Manager requires an inverse table for translating the color table from the source pixel map to the destination pixel map.)

The CopyBits function applies the foreground and background colors of the current graphics port to the image in the destination pixel map (or bitmap), even if the source image is a bitmap. This causes the foreground color to replace all black pixels in the destination and the background color to replace all white pixels. To avoid unwanted coloring of the image, use the RGBForeColor function to set the foreground to black and use the RGBBackColor function to set the background to white before calling CopyBits.

The source bitmap or pixel map must not occupy more memory than half the available stack space. The stack space required by CopyBits is roughly five times the value of the rowBytes field of the source pixel map: one rowBytes value for the pixel map (or bitmap), an additional rowBytes value for dithering, another rowBytes value when stretching or shrinking the source pixel map into the destination, another rowBytes value for any color map changing, and a fifth additional rowBytes value for any color aliasing. If there is insufficient memory to complete a CopyBits operation in Color QuickDraw, the QDError function returns the result code –143.

If you use CopyBits to copy between two graphics ports that overlap, you must first use the LocalToGlobal function to convert to global coordinates, and then specify the global variable screenBits for both the srcBits and dstBits parameters.

The CopyBits function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

If you are reading directly from a NuBus video card with a base address of Fs00000 and there is not a card in the slot (s–1) below it, `CopyBits` reads addresses less than the base address of the pixel map. This causes a bus error. To work around the problem, remap the `baseAddr` field of the pixel map in your video card to at least 20 bytes above the NuBus boundary; an address link of Fs000020 precludes the problem.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ChromaKeyMovie
GlyphaIVOld
MovieBrowser
Palette and GWorld
qteffects.win

Declared in
`QuickdrawAPI.h`

CopyDeepMask

Uses a mask when copying bitmaps or pixel maps between graphics ports (or from an offscreen graphics world into a graphics port). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void CopyDeepMask (
   const BitMap *srcBits,
   const BitMap *maskBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *maskRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn
);
```

Parameters
`srcBits`
The source `BitMap` structure.

`maskBits`
The masking `BitMap` structure.

`dstBits`
The destination `BitMap` structure. The result is clipped to the mask region that you specify in the `maskRgn` parameter, and to the boundary rectangle that you specify in the `dstRect` parameter.

`srcRect`
The source rectangle.

`maskRect`
The mask rectangle. This must be the same size as the rectangle passed in the `srcRect` parameter. The rectangle you pass here selects the portion of the bitmap or pixel map that you specify in the `maskBits` parameter to use as the mask.

`dstRect`
The destination rectangle.

`mode`
The source mode.

`maskRgn`
The mask clipping region. If you do not want to clip to the mask region, specify `NULL`.

Discussion
`CopyDeepMask` combines the effects of the `CopyBits` and `CopyMask` functions. You specify a mask to `CopyDeepMask` so that it transfers the source image to the destination image only where the bits of the mask are set to 1. Use `CopyDeepMask` to move offscreen graphic images into an onscreen window, to blend colors for the image in a pixel map, and to shrink and expand images.

When copying images between color graphics ports, you must coerce each port's `CGrafPort` structure to a `GrafPort` structure, dereference the `portBits` fields of each, and then pass these "bitmaps" in the `srcBits` and `dstBits` parameters. If your application copies a pixel image from a color graphics port called `MyColorPort`, for example, you could specify (* `GrafPtr(MyColorPort)).portBits` in the `srcBits` parameter. The transfer can be performed in any of the transfer modes—with or without adding the `ditherCopy` constant—that are available to `CopyBits` (page 165).

Using the `srcRect` and `dstRect` parameters, you can specify identically or differently sized source and destination rectangles; for differently sized rectangles, `CopyDeepMask` scales the source image to fit the destination. When you specify rectangles in the `srcRect` and `dstRect` parameters, use the local coordinate systems of, respectively, the source and destination graphics ports.

If you specify pixel maps to `CopyDeepMask`, they may range from 1 to 32 pixels in depth. The pixel depth of the mask that you specify in the `maskBits` parameter is applied as a filter between the source and destination pixel maps that you specify in the `srcBits` and `dstBits` parameters. A black mask pixel value means that

the copy operation is to take the source pixel a white value means that the copy operation is to take the destination pixel. Intermediate values specify a weighted average, which is calculated on a color component basis. For each pixel's color component value, the calculation is

(1 – mask) x source + (mask) x destination

Thus high mask values for a pixel's color component reduce that component's contribution from the source `PixMap` structure.

Special Considerations

As with the `CopyMask` function, calls to `CopyDeepMask` are not recorded in pictures and do not print.

See the list of special considerations for `CopyBits` (page 165); these considerations also apply to `CopyDeepMask`.

The `CopyDeepMask` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

CopyMask

Copies a bit or pixel image from one graphics port or offscreen graphics world into another graphics port only where the bits in a mask are set to 1. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void CopyMask (
   const BitMap *srcBits,
   const BitMap *maskBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *maskRect,
   const Rect *dstRect
);
```


Parameters

`srcBits`

The source `BitMap` structure.

`maskBits`

The mask `BitMap` structure.

`dstBits`

The destination `BitMap` structure.

`srcRect`

The source rectangle.

`maskRect`

The mask rectangle. This must be the same size as the rectangle passed in the `srcRect` parameter. The rectangle you pass in this parameter selects the portion of the bitmap or pixel map that you specify in the `maskBits` parameter to use as the mask.

`dstRect`

The destination rectangle.

Discussion

The `CopyMask` function copies the source bitmap or pixel map that you specify in the `srcBits` parameter to a destination bitmap or pixel map that you specify in the `dstBits` parameter—but only where the bits of the mask bitmap or pixel map that you specify in the `maskBits` parameter are set to 1. When copying images between color graphics ports, you must coerce each `CGrafPort` structure to a `GrafPort` structure, dereference the `portBits` fields of each, and then pass these "bitmaps" in the `srcBits` and `dstBits` parameters. If your application copies a pixel image from a color graphics port called `MyColorPort`, for example, you could specify (* `GrafPtr(MyColorPort)).portBits` in the `srcBits` parameter.

Using the `srcRect` and `dstRect` parameters, you can specify identically or differently sized source and destination rectangles; for differently sized rectangles, `CopyMask` scales the source image to fit the destination. When you specify rectangles in the `srcRect` and `dstRect` parameters, use the local coordinate systems of, respectively, the source and destination graphics ports.

If you specify pixel maps to `CopyMask`, they may range from 1 to 32 pixels in depth. The pixel depth of the mask that you specify in the `maskBits` parameter is applied as a filter between the source and destination pixel maps that you specify in the `srcBits` and `dstBits` parameters. A black mask pixel value means that the copy operation is to take the source pixel a white value means that the copy operation is to take the destination pixel. Intermediate values specify a weighted average, which is calculated on a color component basis. For each pixel's color component value, the calculation is

(1 – mask) x source + (mask) x destination

Thus high mask values for a pixel's color component reduce that component's contribution from the source `PixMap` structure.

Use the bitmap returned by `CalcMask` (page 157) as the mask in order to implement a mask copy similar to that performed by the MacPaint lasso tool. In the same way, you can use the pixel map returned by the `CalcCMask` function.

The `CopyDeepMask` (page 168) function combines the functions of the `CopyMask` and `CopyBits` functions.

Special Considerations

Calls to `CopyMask` are not recorded in pictures and do not print.

See the list of special considerations for `CopyBits` (page 165); these considerations also apply to `CopyMask`.

The `CopyMask` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CopyBits vs. CopyMask

GlyphaIVOld

Declared in

`QuickdrawAPI.h`

CopyPixMap

Duplicates a `PixMap` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void CopyPixMap (
   PixMapHandle srcPM,
   PixMapHandle dstPM
);
```


Parameters

`srcPM`

A handle to the `PixMap` structure to be copied.

`dstPM`

On return, a handle to the duplicated `PixMap` structure.

Discussion

Typically, you do not need to call this function in your application code, because the `CopyPixMap` function copies the contents of the source `PixMap` structure to the destination `PixMap` structure. The contents of the color table are copied, so the destination `PixMap` has its own copy of the color table. Because the `baseAddr` field of the `PixMap` structure is a pointer, the pointer, but not the image itself, is copied.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

CopyPixPat

Copies the contents of one pixel pattern to another. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void CopyPixPat (
   PixPatHandle srcPP,
   PixPatHandle dstPP
);
```

Parameters

`srcPP`

A handle to a source pixel pattern, the contents of which you want to copy.

`dstPP`

A handle to a destination pixel pattern, into which you want to copy the contents of the pixel pattern in the `srcPP` parameter.

Discussion

The `CopyPixPat` function copies all of the fields in the source `PixPat` (page 106) structure, including the contents of the data handle, expanded data handle, expanded map, pixel map handle, and color table.

Generally, your application should create a pixel pattern in a `'ppat'` resource, instead of using this function.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

CopyRgn

Makes a copy of a region. (Available in OS X v10.0 through OS X v10.6.)

```
void CopyRgn (
   RgnHandle srcRgn,
   RgnHandle dstRgn
);
```

Parameters

`srcRgn`

A handle to the region to copy.

`dstRgn`

A handle to the region to receive the copy.

Discussion

The `CopyRgn` function copies the mathematical structure of the region whose handle you pass in the `srcRgn` parameter into the region whose handle you pass in the `dstRgn` parameter; that is, `CopyRgn` makes a duplicate copy of `srcRgn`. When calling `CopyRgn`, pass handles that have been returned by the `NewRgn` function in the `srcRgn` and `dstRgn` parameters.

Once this is done, the region indicated by `srcRgn` may be altered (or even disposed of) without affecting the region indicated by `dstRgn`. The `CopyRgn` function does not create the destination region; space must already have been allocated for it by using the `NewRgn` function.

Special Considerations

The `CopyRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Fragment Tool
GlyphaIVOld
Password
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

CreateCGContextForPort

Creates a Quartz 2D drawing environment associated with a graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus CreateCGContextForPort (
   CGrafPtr inPort,
   CGContextRef *outContext
);
```

Parameters

`port`

A color graphics port in which to draw. Offscreen graphics worlds with pixel depths of 1, 2, 4, and 8 are not supported. When using Quartz 2D to draw in a offscreen graphics world, alpha information is always ignored. Printing ports are not supported—if you specify a printing port, this function does nothing and returns a non-zero result code.

`contextPtr`

A pointer to your storage for a Quartz context. Upon completion, `contextPtr` points to a context associated with the port. The context matches the port's pixel depth, width, and height. Otherwise the context is in a default state and does not necessarily match other port attributes such as foreground color, background color, or clip region.

You should release this context when you no longer need it.

Return Value
A result code. If `noErr`, the context was successfully created.

Discussion
This function is not recommended in Mac OS X version 10.1 and later. For information about its replacement, see `QDBeginCGContext` (page 59).

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ATSUICurveAccessDemo

CTMClip

CTMDemo

Declared in
`QuickdrawAPI.h`

CreateNewPort

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
CGrafPtr CreateNewPort (
   void
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites

DesktopSprites.win

MoreOSL

QISA

Declared in
`QuickdrawAPI.h`

CreateNewPortForCGDisplayID

Creates a graphics port associated with a display. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
CGrafPtr CreateNewPortForCGDisplayID (
   UInt32 inCGDisplayID
);
```

Parameters
`displayID`

A display identifier. If the identifier is not valid, the main display is used instead. For information about finding displays, see *Quartz Display Services Reference*.

Return Value
A new display port. The `portBounds` rectangle is the same size as the display. When you are finished using the port, you should call `DisposePort` (page 190) to release it.

Discussion
This function returns a graphics port used to draw directly to a display. The pixel map for the new port is taken from the `GDevice` record corresponding to the display. There is no back buffer associated with the port.

Before calling this function, you should capture the display. For information about capturing displays, see *Quartz Display Services Reference*.

You should not call this function and then attempt to create a Quartz drawing environment inside the port. Instead, applications using Quartz 2D can call `CGDisplayGetDrawingContext` to obtain a context suitable for drawing directly to a captured display.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

CTabChanged

Signals QuickDraw that the content of a `ColorTable` structure has been modified. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void CTabChanged (
   CTabHandle ctab
);
```

Parameters

`ctab`

A handle to the `ColorTable` structure changed by your application.

Discussion

The `CTabChanged` function calls the function `GetCTSeed` and gets a new, unique identifier in the `ctSeed` field of the `ColorTable` structure, and notifies QuickDraw of the change.

Your application should never need to directly modify a `ColorTable` structure and use the `CTabChanged` function; instead, your application should use the QuickDraw functions provided for manipulating the values in a `ColorTable` structure.

Special Considerations

The `CTabChanged` function may move or purge memory in the application heap; do not call the `CTabChanged` function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

ControlBackground

NewCCursor

SetWindBackColor

Declared in

`QDOffscreen.h`

CursorComponentChanged

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSErr CursorComponentChanged (
   ComponentInstance ci
);
```

Return Value
A result code.

Carbon Porting Notes
This function is not implemented on Mac OS X.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

CursorComponentSetData

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSErr CursorComponentSetData (
   ComponentInstance ci,
   long data
);
```

Return Value
A result code.

Carbon Porting Notes
This function is not implemented on Mac OS X.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

DelComp

Removes a custom complement function from the current `GDevice` data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DelComp (
   ColorComplementUPP compProc
);
```

Parameters
`compProc`

A pointer to the complement function, `ColorComplementProcPtr` (page 62), to be deleted. `DelComp` disposes of the chain element, but does nothing to the `ProcPtr` data structure.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

DelSearch

Removes a custom search function from the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DelSearch (
   ColorSearchUPP searchProc
);
```


Parameters

`searchProc`

A pointer to the custom search function, `ColorSearchProcPtr` (page 62) to be deleted. `DelSearch` disposes of the chain element, but does nothing to the `ProcPtr` data structure.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

DeltaPoint

Subtracts the coordinates of one point from another. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
long DeltaPoint (
   Point ptA,
   Point ptB
);
```

Parameters

`p1`

The first point.

`p2`

The second point, the coordinates of which are to be subtracted from the coordinates of the first point.

Return Value

A 32-bit value that contains the differences between the coordinates of the points `p1` and `p2`. The vertical difference is returned in the high 16 bits and the horizontal difference is returned in the low 16 bits.

Discussion

You should not cast the result to a `Point` data structure. Instead, use `HiWord` and `LoWord` to obtain the horizontal and vertical differences.

For example:

```
    Point pointDiff;
```



```
    SInt32 difference = DeltaPoint(p1, p2);
    pointDiff.h = LoWord(difference);
    pointDiff.v = HiWord(difference);
```

While `DeltaPoint` is supported in Carbon, you can achieve the same result in a more direct manner using the function `SubPt` (page 446).

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

deltapoint

(Available in OS X v10.0 through OS X v10.6. Use `DeltaPoint` (page 181) instead.)

```
long deltapoint (
   Point *ptA,
   Point *ptB
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

DeviceLoop

Draws images that are optimized for every screen they cross. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void DeviceLoop (
   RgnHandle drawingRgn,
   DeviceLoopDrawingUPP drawingProc,
   long userData,
   DeviceLoopFlags flags
);
```

Parameters

`drawingRgn`

A handle to the region in which you will draw; this drawing region uses coordinates that are local to its graphics port.

`drawingProc`

A pointer to your own drawing function.

`userData`

Any additional data that you wish to supply to your drawing function.

`flags`

One or more members of the set of flags defined by the " Device Loop Flags" (page 126) data type. if you want to use the default behavior of `DeviceLoop`, specify an empty set ([]) in this parameter.

Discussion

The `DeviceLoop` function searches for graphics devices that intersect your window's drawing region, and it calls your drawing function for each dissimilar video device it finds.

Because `DeviceLoop` provides your drawing function with the pixel depth and other attributes of each video device, your drawing function can optimize its drawing for each video device.

See `DeviceLoopDrawingProcPtr` (page 64) for a description of the drawing function you must provide for the `drawingProc` parameter.

Special Considerations

The `DeviceLoop` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

GetDragHiliteColor

Declared in
QuickdrawAPI.h

DiffRgn

Subtracts one region from another. (Available in OS X v10.0 through OS X v10.6.)

```
void DiffRgn (
   RgnHandle srcRgnA,
   RgnHandle srcRgnB,
   RgnHandle dstRgn
);
```

Parameters

`srcRgnA`

A handle to the region to subtract from.

`srcRgnB`

A handle to the region to subtract.

`dstRgn`

On return, a handle to the region holding the resulting area. If the first source region is empty, `DiffRgn` sets the destination to the empty region defined by the rectangle (0,0,0,0).

The `DiffRgn` function does not create the destination region; you must have already allocated memory for it by using the `NewRgn` (page 336) function.

The destination region may be one of the source regions, if desired.

Discussion

The DiffRgn procedure subtracts the region whose handle you pass in the srcRgnB parameter from the region whose handle you pass in the srcRgnA parameter and places the difference in the region whose handle you pass in the dstRgn parameter. If the first source region is empty, DiffRgn sets the destination to the empty region defined by the rectangle (0,0,0,0).

The DiffRgn procedure does not create the destination region; you must have already allocated memory for it by using the NewRgn function. The destination region may be one of the source regions, if desired.

Special Considerations

The `DiffRgn` function may temporarily use heap space that's twice the size of the two input regions.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
FinderDragPro
Inside Mac Movie TB Code
MovieBrowser
Password
TE Over Background

Declared in
`QuickdrawAPI.h`

DisposeCCursor

Disposes of all structures allocated by the `GetCCursor` function. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeCCursor (
   CCrsrHandle cCrsr
);
```

Parameters
`cCrsr`
A handle to the color cursor to be disposed of.

Discussion
Use `DisposeCCursor` for each call to the `GetCCursor` (page 227) function.

The `DisposeCCursor` function is also available as the `DisposCCursor` function.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
qtshoweffect
qtshoweffect.win

Declared in
`QuickdrawAPI.h`

DisposeCTable

Disposes a `ColorTable` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void DisposeCTable (
   CTabHandle cTable
);
```

Parameters

`cTable`

A handle to a `ColorTable` structure to dispose of.

Discussion

The DisposeCTable procedure disposes of the ColorTable record whose handle you pass in the cTable parameter.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CollectPictColors

ElectricImageComponent.win

Inside Mac Movie TB Code

QTMusicToo

Declared in

`QuickdrawAPI.h`

DisposeDeviceLoopDrawingUPP

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void DisposeDeviceLoopDrawingUPP (
   DeviceLoopDrawingUPP userUPP
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Declared in
`QuickdrawTypes.h`

DisposeGDevice

Disposes of a `GDevice` structure, releases the space allocated for it, and disposes of all the data structures allocated for it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeGDevice (
   GDHandle gdh
);
```

Parameters
`gdh`
A handle to the `GDevice` structure.

Discussion
Generally, you should never need to use this function. Color QuickDraw calls this function when appropriate. The `DisposeGDevice` function is also available as the `DisposGDevice` function.

When your application uses the `DisposeGWorld` function to dispose of an offscreen graphics world, `DisposeGDevice` disposes of its `GDevice` structure.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

DisposeGWorld

Disposes of all the memory allocated for an offscreen graphics world. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeGWorld (
```



```
   GWorldPtr offscreenGWorld
);
```

Parameters

`offscreenGWorld`

A pointer to an offscreen graphics world. In this parameter, pass the pointer returned to your application by the `NewGWorld` function when you created the offscreen graphics world.

Discussion

The `DisposeGWorld` function disposes of all the memory allocated for the specified offscreen graphics world, including the pixel map, color table, pixel image, and `GDevice` structure (if one was created).

Call `DisposeGWorld` only when your application no longer needs the pixel image associated with this offscreen graphics world. If this offscreen graphics world was the current device, the current device is reset to the device stored in the global variable `MainDevice`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

DesktopSprites

DesktopSprites.win

qteffects

qteffects.win

qtspritesplus

Declared in

`QDOffscreen.h`

DisposePixMap

Disposes a `PixMap` structure and its color table. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void DisposePixMap (
   PixMapHandle pm
);
```


Parameters

`pm`

A handle to the `PixMap` structure to be disposed of.

Discussion

The `CloseCPort` function calls `DisposePixMap`.

Your application typically does not need to call this function. This function is also available as `DisposPixMap`.

If your application uses `DisposePixMap`, take care that it does not dispose of a `PixMap` structure whose color table is the same as the current device's CLUT.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

DisposePixPat

Releases the storage allocated to a pixel pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void DisposePixPat (
   PixPatHandle pp
);
```

Parameters

`pp`

A handle to the pixel pattern to be disposed of.

Discussion

The `DisposePixPat` function disposes of the data handle, expanded data handle, and pixel map handle allocated to the pixel pattern that you specify in the `ppat` parameter.

The `DisposePixPat` function is also available as the `DisposPixPat` function.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CopyBits vs. CopyMask

Declared in
`QuickdrawAPI.h`

DisposePort

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void DisposePort (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites
DesktopSprites.win
MoreOSL
QISA

Declared in
`QuickdrawAPI.h`

DisposeRgn

Releases the memory occupied by a region. (Available in OS X v10.0 through OS X v10.6.)

```
void DisposeRgn (
   RgnHandle rgn
);
```


Parameters

`rgn`

A handle to the region to dispose. This handle should be a region handle returned by the `NewRgn` (page 336) function.

Discussion

Use `DisposeRgn` only after you are completely through with a region.

Special Considerations

The `DisposeRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial
FinderDragPro
HTMLSample
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

DisposeScreenBuffer

Disposes an offscreen graphics world. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeScreenBuffer (
   PixMapHandle offscreenPixMap
);
```

Parameters

`offscreenPixMap`

A handle to an existing offscreen `PixMap` structure.

Discussion

Generally, applications do not need to use `DisposeScreenBuffer`. The `DisposeGWorld` (page 187) function uses the `DisposeScreenBuffer` function when disposing of an offscreen graphics world.

The `DisposeScreenBuffer` function disposes of the memory allocated for the base address of an offscreen pixel image.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QDOffscreen.h`

DrawPicture

Draws a picture on any type of output device. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void DrawPicture (
   PicHandle myPicture,
   const Rect *dstRect
);
```


Parameters

`myPicture`

A handle to the picture to be drawn. You must access a picture through its handle.

When creating pictures, the `OpenCPicture` (page 345) and `OpenPicture` (page 348) functions return their handles. You can use the `GetPicture` (page 246) function to get a handle to a QuickDraw picture stored in a `'PICT'` resource. To get a handle to a QuickDraw picture stored in a `'PICT'` file, you must use File Manager functions. To get a picture stored in the scrap, use the Scrap Manager function `GetScrap` to get a handle to its data and then coerce this handle to one of type `PicHandle`.

`dstRect`

A destination rectangle, specified in coordinates local to the current graphics port, in which to draw the picture. The `DrawPicture` function shrinks or expands the picture as necessary to align the borders of its bounding rectangle with the rectangle you specify in this parameter. To display a picture at a resolution other than that at which it was created, your application should compute an appropriate destination rectangle by scaling its width and height by the following factor:

```
scale factor = destination resolution / source resolution
```

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle. Use the `GetPictInfo` function to gather information about a picture. The `PictInfo` structure returned by `GetPictInfo` returns the picture's resolution in its `hRes` and `vRes` fields. The `sourceRect` field contains the bounding rectangle for displaying the image at its optimal resolution.

Discussion

Within the rectangle that you specify in the `dstRect` parameter, the `DrawPicture` function draws the picture that you specify in the `myPicture` parameter.

The `DrawPicture` function passes any picture comments to the `StdComment` function pointed to by the `commentProc` field of the `CQDProcs` or `QDProcs` structure, which in turn is pointed to by the `grafProcs` field of a `CGrafPort` or `GrafPort` structure. The default `StdComment` function provided by QuickDraw does no comment processing whatsoever. If you want to process picture comments when drawing a picture, use the `SetStdCProcs` function to assist you in changing the `CQDProcs` structure and use the `SetStdProcs` function to assist you in changing the `QDProcs` structure.

Special Considerations

Always use the `ClipRect` function to specify a clipping region appropriate for your picture before defining it with the `OpenCPicture` (or `OpenPicture`) function. If you do not use `ClipRect` to specify a clipping region, `OpenCPicture` uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you want to scale the picture, the clipping region can become invalid when `DrawPicture` scales the clipping region—in which case, your picture will not be drawn.

On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when `DrawPicture` draws it. Setting a clipping region equal to the port rectangle of the current graphics port always sets a valid clipping region.

When it scales, `DrawPicture` changes the size of the font instead of scaling the bits. However, the widths used by bitmap fonts are not always linear. For example, the 12-point width isn't exactly 1/2 of the 24-point width. This can cause lines of text to become slightly longer or shorter as the picture is scaled. The difference is often insignificant, but if you are trying to draw a line of text that fits exactly into a box (a spreadsheet cell, for example), the difference can become noticeable to the user—most typically, at print time. The easiest way to avoid such problems is to specify a destination rectangle that is the same size as the bounding rectangle for the picture. Otherwise, your application may need to directly process the opcodes in the picture instead of using `DrawPicture`.

You may also have disappointing results if the fonts contained in an image are not available on the user's system. Before displaying a picture, your application may want to use the Picture Utilities to determine what fonts are contained in the picture, and then use Font Manager functions to determine whether the fonts are available on the user's system. If they are not, you can use Dialog Manager functions to display an alert box warning the user of display problems.

If there is insufficient memory to draw a picture in Color QuickDraw, the `QDError` function returns the result code `noMemForPictPlaybackErr`.

The `DrawPicture` function may move or purge memory.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
HTMLSample

MovieBrowser

qdmediamaker.win

qteffects.win

QTMusicToo

Declared in
`QuickdrawAPI.h`

EmptyRect

Determines whether a rectangle is an empty rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean EmptyRect (
   const Rect *r
);
```

Parameters

`r`

The rectangle to examine.

Return Value

`TRUE` if the rectangle that you specify in the `r` parameter is an empty rectangle, `FALSE` if it is not. A rectangle is considered empty if the bottom coordinate is less than or equal to the top coordinate or if the right coordinate is less than or equal to the left.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

MovieBrowser

QTCarbonShell

vrmovies.win

vrscript

vrscript.win

Declared in

`QuickdrawAPI.h`

EmptyRgn

Determines whether a region is empty. (Available in OS X v10.0 through OS X v10.6.)


```
Boolean EmptyRgn (
   RgnHandle rgn
);
```

Parameters

`rgn`

A handle to the region to test for emptiness.

Return Value

`TRUE` if the region whose handle you pass in the `rgn` parameter is an empty region or `FALSE` if it is not.

Discussion

The `EmptyRgn` function does not create an empty region. To create an empty region, you can perform any of the following operations:

- Use `NewRgn` (page 336).
- Pass the handle to an empty region to `CopyRgn` (page 174).
- Pass an empty rectangle to either `SetRectRgn` (page 428) or `RectRgn` (page 384).
- Call `CloseRgn` (page 162) without a previous call to `OpenRgn` (page 350).
- Call `CloseRgn` (page 162) without performing any drawing after calling `OpenRgn` (page 350).
- Pass an empty region to `OffsetRgn` (page 343).
- Pass an empty region or too large an inset to `InsetRgn` (page 283)
- Pass two nonintersecting regions to `SectRgn` (page 396).
- Pass two empty regions to `UnionRgn` (page 450).
- Pass two identical or nonintersecting regions to `DiffRgn` (page 184) or `XorRgn` (page 457).

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

CopyBits vs. CopyMask
HITextViewDemo
MoreOSL
Password
TE Over Background

Declared in
QuickdrawAPI.h

EqualPt

Determines whether the coordinates of two given points are equal. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean EqualPt (
   Point pt1,
   Point pt2
);
```

Parameters

`pt1`

The first of two points to be compared.

`pt2`

The second of two points to be compared.

Return Value
`TRUE` if the coordinates of the two points are equal, or `FALSE` if they are not.

Discussion
The EqualPt function compares the points specified in the pt1 and pt2 parameters and returns TRUE if their coordinates are equal or FALSE if they are not.

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
QuickdrawAPI.h

EqualRect

Determines whether two rectangles are equal. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean EqualRect (
   const Rect * rect1,
   const Rect * rect2
);
```


Parameters

`rect1`

The first of two rectangles to compare.

`rect2`

The second of two rectangles to compare.

Return Value

`TRUE` if the rectangles are equal, `FALSE` if they are not.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

HTMLUserPane

Live Scroll

Declared in

`QuickdrawAPI.h`

EqualRgn

Determines whether two regions have identical sizes, shapes, and locations. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean EqualRgn (
   RgnHandle rgnA,
   RgnHandle rgnB
);
```

Parameters

`rgnA`

A handle to the first of two regions to compare.

`rgnB`

A handle to the second of two regions to compare.

Return Value

`TRUE` if the two regions are equal; `FALSE` if they are not. The two regions must have identical sizes, shapes, and locations to be considered equal. Any two empty regions are always equal.

Discussion

The EqualRgn function compares the two regions whose handles you pass in the rgnA and rgnB parameters and returns TRUE if they're equal or FALSE if they're not.

The two regions must have identical sizes, shapes, and locations to be considered equal. Any two empty regions are always equal.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

EraseArc

Erases a wedge. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void EraseArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

`r`

The rectangle that defines an oval's boundaries.

`startAngle`

The angle indicating the start of the arc.

`arcAngle`

The angle indicating the arc's extent.

Discussion

Using the `patCopy` pattern mode, the `EraseArc` function draws a wedge of the oval bounded by the rectangle that you specify in the `r` parameter with the background pattern for the current graphics port. As in `FrameArc` (page 218) , use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `EraseArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

EraseOval

Erases an oval. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void EraseOval (
   const Rect *r
);
```

Parameters

`r`

The rectangle that defines the oval's boundary.

Discussion

Using the background pattern for the current graphics port and the `patCopy` pattern mode, the `EraseOval` function draws the interior of an oval just inside the bounding rectangle that you specify in the `r` parameter. This effectively erases the oval bounded by the specified rectangle.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `EraseOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qdmediamaker.win

Declared in

`QuickdrawAPI.h`

ErasePoly

Erases a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void ErasePoly (
   PolyHandle poly
);
```

Parameters

`poly`

A handle to the polygon to erase. The `OpenPoly` (page 349) function returns this handle when you first create the polygon.

Discussion

Using the `patCopy` pattern mode, the `ErasePoly` function draws the interior of the polygon whose handle you pass in the `poly` parameter with the background pattern for the current graphics port.

This function leaves the location of the graphics pen unchanged.

This function temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 319) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Special Considerations
The `ErasePoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

EraseRect

Erases a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void EraseRect (
   const Rect *r
);
```

Parameters
`r`
The rectangle to erase.

Discussion
Using the `patCopy` pattern mode, the `EraseRect` function draws the interior of the rectangle that you specify in the `r` parameter with the background pattern for the current graphics port. This effectively erases the rectangle, making the shape blend into the background pattern of the graphics port. For example, use `EraseRect` to erase the port rectangle for a window before redrawing into the window.

This function leaves the location of the graphics pen unchanged.

Special Considerations
The `EraseRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
qteffects

qteffects.win

QTMusicToo

qtspritesplus

qtwiredspritesjr.win

Declared in
`QuickdrawAPI.h`

EraseRgn

Erases a region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void EraseRgn (
   RgnHandle rgn
);
```

Parameters
`rgn`

The region to erase.

Discussion
Using the `patCopy` pattern mode, the `EraseRgn` function draws the interior of the region whose handle you pass in the `rgn` parameter with the background pattern for the current graphics port.

This function leaves the location of the graphics pen unchanged.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Special Considerations

The `EraseRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial

CustomWindow

hacktv

HackTV Carbon

hacktv.win

Declared in

`QuickdrawAPI.h`

EraseRoundRect

Erases a rounded rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void EraseRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

`r`

The rectangle that defines the rounded rectangle's boundaries.

`ovalWidth`

The width of the oval defining the rounded corner.

`ovalHeight`

The height of the oval defining the rounded corner.

Discussion

Using the `patCopy` pattern mode, the `EraseRoundRect` function draws the interior of the rounded rectangle bounded by the rectangle that you specify in the `r` parameter with the background pattern of the current graphics port. This effectively erases the rounded rectangle. Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners of the rounded rectangle.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `EraseRoundRect` function may move or purge memory blocks in the application; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillArc

Fills a wedge with any available bit pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FillArc (
   const Rect *r,
   short startAngle,
   short arcAngle,
   const Pattern *pat
);
```

Parameters

`r`

The rectangle that defines an oval's boundaries.

`startAngle`

The angle indicating the start of the arc.

`arcAngle`

The bit pattern to use for the fill.

`pat`

The angle indicating the arc's extent.

Discussion

Using the `patCopy` pattern mode and the pattern defined in the `Pattern` (page 100) structure that you specify in the `pat` parameter, the `FillArc` function draws a wedge of the oval bounded by the rectangle that you specify in the `r` parameter. As in `FrameArc` (page 218) use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

This function leaves the location of the graphics pen unchanged.

Use `GetPattern` (page 243) and `GetIndPattern` (page 239) to get a pattern stored in a resource.

Use `PaintArc` (page 354) to draw a wedge with the pen pattern for the current graphics port.

To fill a wedge with a pixel pattern, use the `FillCArc` function.

Special Considerations

The `FillArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillCArc

Fills a wedge with the given pixel pattern, using the `patCopy` pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FillCArc (
   const Rect *r,
   short startAngle,
   short arcAngle,
   PixPatHandle pp
);
```


Parameters

`r`

The rectangle that defines the oval's boundaries.

`startAngle`

The angle indicating the start of the arc.

`arcAngle`

The angle indicating the arc's extent.

`pp`

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

Use the `startAngle` and `arcAngle` parameters to define the arc of the wedge. This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillCOval

Fills an oval with the given pixel pattern, using the `patCopy` pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FillCOval (
   const Rect *r,
   PixPatHandle pp
);
```

Parameters

`r`

The rectangle containing the oval to be filled.

`pp`

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillCPoly

Fills a polygon with the given pixel pattern, using the `patCopy` pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FillCPoly (
   PolyHandle poly,
   PixPatHandle pp
);
```

Parameters

`poly`

A handle to the polygon to be filled.

`pp`

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillCRect

Fills a rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FillCRect (
   const Rect *r,
   PixPatHandle pp
);
```

Parameters

`r`

The rectangle to be filled.

`pp`

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CopyBits vs. CopyMask

Declared in
`QuickdrawAPI.h`

FillCRgn

Fills a region with the given pixel pattern, using the `patCopy` pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FillCRgn (
   RgnHandle rgn,
   PixPatHandle pp
);
```

Parameters

`rgn`

A handle to the region to be filled.

`pp`

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion
This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations
The `FillCRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

FillCRoundRect

Fills a rounded rectangle with the given pixel pattern, using the `patCopy` *pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)*

```
void FillCRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight,
   PixPatHandle pp
);
```

Parameters

`r`

The rectangle that defines the rounded rectangle's boundaries.

`ovalWidth`

The width of the oval defining the rounded corner.

`ovalHeight`

The height of the oval defining the rounded corner.

`pp`

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners. This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillOval

Fills an oval with any available bit pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FillOval (
   const Rect *r,
   const Pattern *pat
);
```

Parameters

`r`

The rectangle that defines the oval's boundaries.

`pat`

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode and the bit pattern defined in the `Pattern` (page 100) structure that you specify in the `pat` parameter, the `FillOval` function draws the interior of an oval just inside the bounding rectangle that you specify in the `r` parameter. The pen location does not change.

Use `GetPattern` (page 243) and `GetIndPattern` (page 239) , to get a pattern stored in a resource. Use the `PaintOval` (page 355) function to draw the interior of an oval with the pen pattern for the current graphics port.

To fill an oval with a pixel pattern, use the `FillCOval` function.

Special Considerations

The `FillOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillPoly

Fills a polygon with any available bit pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FillPoly (
   PolyHandle poly,
   const Pattern *pat
);
```

Parameters

`poly`

A handle to the polygon to fill. The `OpenPoly` (page 349) function returns this handle when you first create the polygon.

`pat`

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode, the `FillPoly` function draws the interior of the polygon whose handle you pass in the `poly` parameter with the pattern defined in the `Pattern` (page 100) structure that you specify in the `pat` parameter.

This function leaves the location of the graphics pen unchanged.

This function temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 319) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Use `GetPattern` (page 243) and `GetIndPattern` (page 239) to get a pattern stored in a resource.

Use `PaintPoly` (page 356) to draw the interior of a polygon with the pen pattern for the current graphics port. To fill a polygon with a pixel pattern, use the `FillCPoly` function.

Special Considerations

The `FillPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

FillRect

Fills a rectangle with any available bit pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FillRect (
   const Rect * r,
   const Pattern * pat
);
```

Parameters
`r`

The rectangle to fill.

`pat`

The bit pattern to use for the fill.

Discussion
Using the `patCopy` pattern mode, the `FillRect` function draws the interior of the rectangle that you specify in the `r` parameter with the pattern defined in the `Pattern` (page 100) structure that you specify in the `pat` parameter. This function leaves the pen location unchanged.

Use `GetPattern` (page 243) and `GetIndPattern` (page 239) , to get a pattern stored in a resource.

Use the `PaintRect` (page 357) to draw the interior of a rectangle with the pen pattern for the current graphics port. To fill a rectangle with a pixel pattern, use the `FillCRect` function.

Special Considerations
The `FillRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
offscreen.win

Declared in
`QuickdrawAPI.h`

FillRgn

Fills a region with any available bit pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FillRgn (
   RgnHandle rgn,
   const Pattern * pat
);
```

Parameters

`rgn`

A handle to the region to fill.

`pat`

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode, the `FillRgn` function draws the interior of the region with the pattern defined in the `Pattern` (page 100) structure that you specify in the `pat` parameter.

This function leaves the location of the graphics pen unchanged.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Use `GetPattern` (page 243) and `GetIndPattern` (page 239) to get a pattern stored in a resource.

Use `PaintRgn` (page 358) to draw the interior of a region with the pen pattern for the current graphics port. To fill a region with a pixel pattern, use the `FillCRegion` function.

Special Considerations

The `FillRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

FillRoundRect

Fills a rounded rectangle with any available bit pattern. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FillRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight,
   const Pattern *pat
);
```

Parameters

`r`

The rectangle that defines the rounded rectangle's boundaries.

`ovalWidth`

The width of the oval defining the rounded corner.

`ovalHeight`

The height of the oval defining the rounded corner.

`pat`

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode, the `FillRoundRect` function draws the interior of the rounded rectangle bounded by the rectangle that you specify in the `r` parameter with the bit pattern defined in the `Pattern` structure that you specify in the `pat` parameter. Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners. The pen location does not change.

To fill a rounded rectangle with a pixel pattern, use the `FillCRoundRect` function.

Use `GetPattern` (page 243) and `GetIndPattern` (page 239) to get a pattern stored in a resource. Use `PaintRoundRect` (page 359) to draw the interior of a rounded rectangle with the pen pattern for the current graphics port.

Special Considerations
The `FillRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

ForeColor

Changes the color of the "ink" used for framing, painting, and filling on computers that support only basic QuickDraw. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void ForeColor (
   long color
);
```

Parameters
`color`
One of eight color values. See "Color Constants" (page 122).

Discussion
By default, the foreground color of a `GrafPort` is black.

The `ForeColor` function sets the foreground color for the current graphics port to the color that you specify in the `color` parameter. When you draw with the `patCopy` and `srcCopy` transfer modes, for example, black pixels are drawn in the color you specify with `ForeColor`.

When printing, use the `ColorBit` (page 165) function to set the foreground color.

All nonwhite colors appear as black on black-and-white screens. Before you use `ForeColor`, use the `DeviceLoop` function to determine the color characteristics of the current screen.

Special Considerations

The `ForeColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Version Notes

In System 7, you may instead use the color QuickDraw function `RGBForeColor`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

ATSUICurveAccessDemo
CollectPictColors
CreateMovie
Inside Mac Movie TB Code
QTMusicToo

Declared in

`QuickdrawAPI.h`

FrameArc

Draws an arc of the oval that fits inside a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FrameArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

`r`

The rectangle that defines an oval's boundaries.

`startAngle`
The angle indicating the start of the arc.

`arcAngle`
The angle indicating the arc's extent.

Discussion

Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameArc` function draws an arc of the oval bounded by the rectangle that you specify in the `r` parameter. Use the `startAngle` parameter to specify where the arc begins as modulo 360. Use the `arcAngle` parameter to specify how many degrees the arc covers. Specify whether the angles are in positive or negative degrees a positive angle goes clockwise, while a negative angle goes counterclockwise. Zero degrees is at 12 o'clock high, 90 (or –270) is at 3 o'clock, 180 (or –180) is at 6 o'clock, and 270 (or –90) is at 9 o'clock. Measure other angles relative to the bounding rectangle.

A line from the center of the rectangle through its upper-right corner is at 45, even if the rectangle is not square a line through the lower-right corner is at 135, and so on.

The arc is as wide as the pen width and as tall as the pen height. The pen location does not change.

Special Considerations

The `FrameArc` function differs from other QuickDraw functions that frame shapes in that the arc is not mathematically added to the boundary of a region that's open and being formed.

The `FrameArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Out of This GWorld
Tiler

Declared in

`QuickdrawAPI.h`

FrameOval

Draws an outline inside an oval. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FrameOval (
   const Rect *r
);
```

Parameters

`r`

The rectangle that defines the oval's boundary.

Discussion

Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameOval` function draws an outline just inside the oval with the bounding rectangle that you specify in the `r` parameter. The outline is as wide as the pen width and as tall as the pen height. The pen location does not change.

If a region is open and being formed, the outside outline of the new oval is mathematically added to the region's boundary.

Special Considerations

The `FrameOval` function may move or purge memory blocks in the application; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CTMClip

GrabBag

TubeTest

Declared in

`QuickdrawAPI.h`

FramePoly

Draws the outline of a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FramePoly (
   PolyHandle poly
);
```

Parameters

`poly`

A handle to the polygon to draw. The `OpenPoly` (page 349) function returns this handle when you first create the polygon.

Discussion

Using the current graphics port's pen pattern, pattern mode, and size, the `FramePoly` function plays back the line-drawing commands that define the polygon whose handle you pass in the `poly` parameter.

The graphics pen hangs below and to the right of each point on the boundary of the polygon. Thus, the drawn polygon extends beyond the right and bottom edges of the polygon's bounding rectangle (which is stored in the `polyBBox` field of the `Polygon` structure) by the pen width and pen height, respectively. All other graphics operations, such as painting a polygon with the `PaintPoly` function, occur strictly within the boundary of the polygon.

If a polygon is open and being formed, `FramePoly` affects the outline of the polygon just as if the line-drawing functions themselves had been called. If a region is open and being formed, the outside outline of the polygon being framed is mathematically added to the region's boundary.

The result of this function is undefined whenever any horizontal or vertical line through the polygon would intersect the polygon's outline more than 50 times.

Special Considerations

The `FramePoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

MouseTracking

Declared in
`QuickdrawAPI.h`

FrameRect

Draws an outline inside a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FrameRect (
   const Rect * r
);
```

Parameters
`r`
The rectangle to frame.

Discussion
Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameRect` function draws an outline just inside the rectangle that you specify in the `r` parameter. The outline is as wide as the pen width and as tall as the pen height. The pen location does not change.

If a region is open and being formed, the outside outline of the new rectangle is mathematically added to the region's boundary.

Special Considerations
The `FrameRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CollectPictColors
kcapApp
MovieBrowser
QTMusicToo
TimeCode Media Handlers

Declared in
`QuickdrawAPI.h`

FrameRgn

Draws an outline inside a region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void FrameRgn (
   RgnHandle rgn
);
```

Parameters
`rgn`
A handle to the region to frame.

Discussion
Using the current graphics port's pen pattern, pattern mode, and pen size, the `FrameRgn` function draws an outline just inside the region whose handle you pass in the rgn parameter. The outline never goes outside the region boundary. The pen location does not change.

If a region is open and being formed, the outside outline of the region being framed is mathematically added to that region's boundary.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined. The `FrameRgn` function in particular requires that there would be no more than 25 such intersections.

Special Considerations
The `FrameRgn` function calls the functions `CopyRgn`, `InsetRgn`, and `DiffRgn`, so `FrameRgn` may temporarily use heap space that's three times the size of the original region.

The `FrameRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
kcapApp

Declared in
`QuickdrawAPI.h`

FrameRoundRect

Draws an outline inside a rounded rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void FrameRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

`r`

The rectangle that defines the rounded rectangle's boundaries.

`ovalWidth`

The width of the oval defining the rounded corner.

`ovalHeight`

The height of the oval defining the rounded corner.

Discussion

Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameRoundRect` function draws an outline just inside the rounded rectangle bounded by the rectangle that you specify in the `r` parameter. The outline is as wide as the pen width and as tall as the pen height. The pen location does not change.

Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners of the rounded rectangle.

If a region is open and being formed, the outside outline of the new rounded rectangle is mathematically added to the region's boundary.

Special Considerations

The `FrameRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

MovieBrowser
QTMusicToo
qtspritesplus
qtwiredactions
vrbackbuffer

Declared in

`QuickdrawAPI.h`

GDeviceChanged

Notifies QuickDraw that the content of a `GDevice` structure has been modified. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void GDeviceChanged (
   GDHandle gdh
);
```

Discussion

If your application changes the `pmTable` field of the `PixMap` structure specified in a `GDevice` structure, call `GDeviceChanged`. If your application changes the content of the `ColorTable` structure referenced by the `PixMap` structure, call both `GDeviceChanged` and `CTabChanged`.

Your application should never need to directly modify a `GDevice` structure and use the `GDeviceChanged` function; instead, your application should use the QuickDraw functions described in this book for manipulating the values in a `GDevice` structure.

Special Considerations

The `GDeviceChanged` function may move or purge memory in the application heap; do not call the `GDeviceChanged` function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QDOffscreen.h`

GetBackColor

Obtains the background color of the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void GetBackColor (
   RGBColor *color
);
```

Parameters
`color`

On return, the `RGBColor` structure for the current background color.

Discussion
This function operates for graphics ports defined by both the `GrafPort` and `CGrafPort` structures. If the current graphics port is defined by a `CGrafPort` structure, the returned value is taken directly from the `rgbBkColor` field.

If the current graphics port is defined by a `GrafPort` structure, then only eight possible colors can be returned. These eight colors are determined by the values in a global variable named `QDColors`, which is a handle to a color table containing the current QuickDraw colors.

Use the `RGBBackColor` (page 388) function to change the background color.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial

qteffects.win
qtspritesplus
qtwiredactions
qtwiredspritesjr.win

Declared in
`QuickdrawAPI.h`

GetCCursor

Loads a color cursor resource into memory. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
CCrsrHandle GetCCursor (
   short crsrID
);
```

Parameters
`crsrID`
The resource ID of the cursor that you want to display.

Return Value
A handle to the new `CCrsr` structure. To display this cursor on the screen, call `SetCCursor`. If a resource with the specified ID isn't found, then this function returns a `NULL` handle.

Discussion
The `GetCCursor` function creates a new `CCrsr` (page 77) structure and initializes it using the information in the 'crsr' resource with the specified ID.

Since the `GetCCursor` function creates a new `CCrsr` structure each time it is called, do not call the `GetCCursor` function before each call to the `SetCCursor` function. Unlike the way `GetCursor` and `SetCursor` are normally used, `GetCCursor` does not dispose of or detach the resource, so resources of type `'crsr'` should typically be purgeable. Call the `DisposeCCursor` (page 185) function when you are finished using the color cursor created with `GetCCursor`.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
qtshoweffect
qtshoweffect.win

Declared in
`QuickdrawAPI.h`

GetClip

Saves the clipping region of the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void GetClip (
   RgnHandle rgn
);
```

Parameters
`rgn`
A handle to the region to be clipped. The `GetClip` function changes this region to one that's equivalent to the clipping region of the current graphics port. The `GetClip` function doesn't change the region handle.

Discussion
You can use the `GetClip` and `SetClip` functions to preserve the current clipping region: use `GetClip` to save the current port's clipping region, and use `SetClip` to restore it. If, for example, you want to draw a half-circle on the screen, you can set the clipping region to half of the square that would enclose the whole circle, and then draw the whole circle. Only the half within the clipping region is actually drawn in the graphics port.

The `GetClip` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
HTMLSample
HTMLUserPane

Live Scroll
TE Over Background

Declared in
`QuickdrawAPI.h`

GetCPixel

Determines the color of an individual pixel specified in the `h` and `v` parameters. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void GetCPixel (
   short h,
   short v,
   RGBColor *cPix
);
```

Parameters

`h`

The horizontal coordinate of the point at the upper-left corner of the pixel.

`v`

The vertical coordinate of the point at the upper-left corner of the pixel.

`cPix`

On return, the `RGBColor` structure for the pixel color.

Discussion

Use the `SetCPixel` (page 404) function to change the color of this pixel.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

GetCTable

Obtains a color table stored in a `'clut'` resource. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
CTabHandle GetCTable (
   short ctID
);
```

Parameters

`ctID`

The resource ID of a `'clut'` resource.

Return Value

A handle to the color table. If the `'clut'` resource with that ID is not found, `GetCTable` returns `NULL`. Before you place this handle in the `pmTable` field of a `PixMap` structure, first use the `DisposeCTable` function to dispose of the handle already there.

Discussion

Before you modify a `ColorTable` structure, change its `ctSeed` field to invalidate it. To do this, use the `CTabChanged` (page 178) function.

The `GetCTable` function recognizes a number of standard `'clut'` resource IDs. You can obtain the default grayscale color table for a given pixel depth by calling `GetCTable`, adding 32 (decimal) to the pixel depth, and passing these values in the `ctID` parameter:

- A pixel depth of 1. Pass a resource ID of 33. Color table composition: black, white.
- A pixel depth of 2. Pass a resource ID of 34. Color table composition: black, 33% gray, 66% gray, white.
- A pixel depth of 4. Pass a resource ID of 36. Color table composition: black, 14 shades of gray, white.
- A pixel depth of 8. Pass a resource ID of 40. Color table composition: black, 254 shades of gray, white.

For full color, obtain the default color tables by adding 64 to the pixel depth and passing these values in the `ctID` parameter:

- A pixel depth of 2. Pass a resource ID of 66. Color table composition: black, 50% gray, highlight color, white.
- A pixel depth of 4. Pass a resource ID of 68. Color table composition: black, 14 colors including the highlight color, white.
- A pixel depth of 8. Pass a resource ID of 72. Color table composition: black, 254 colors including the highlight color, white.

Special Considerations

The `GetCTable` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Inside Mac Movie TB Code
Palette and GWorld
QTMusicToo
qtreadwritejpeg
qtreadwritejpeg.win

Declared in

`QuickdrawAPI.h`

GetCTSeed

Obtains a unique seed value for a color table created by your application. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
long GetCTSeed (
   void
);
```

Return Value

A unique seed value that you can use in the `ctSeed` field of a color table created by your application. It is greater than the value stored in the constant `minSeed`.

Discussion

The seed value guarantees that the color table is recognized as distinct from the destination, and that color table translation is performed properly.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CollectPictColors
ElectricImageComponent.win
Out of This GWorld

Declared in
`QuickdrawAPI.h`

GetCursor

Loads a cursor resource into memory. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
CursHandle GetCursor (
   short cursorID
);
```

Parameters
`cursorID`
The resource ID for the cursor you want to display. You can supply one of the "Cursor ID Constants" (page 124) to get a handle to one of the standard cursors.

Return Value
A handle to a `Cursor` structure for the cursor with the resource ID that you specify in the `cursorID` parameter. If the resource cannot be read into memory, `GetCursor` returns `NULL`.

Discussion
To get a handle to a color cursor, use the `GetCCursor` (page 227) function.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CopyBits vs. CopyMask
MovieBrowser
Password
QTMusicToo

TE Over Background

Declared in
`QuickdrawAPI.h`

GetDeviceList

Obtains a handle to the first `GDevice` structure in the device list. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
GDHandle GetDeviceList (
   void
);
```

Return Value
A handle to the first `GDevice` structure in the global variable `DeviceList`.

Discussion
All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations
The `GetDeviceList` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
GLCarbon1ContextPbuffer
GLCarbonSharedPbuffer

Declared in
`QuickdrawAPI.h`

GetForeColor

Obtains the color of the foreground color for the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void GetForeColor (
   RGBColor *color
);
```

Parameters

`color`

On return, the `RGBColor` structure for the current foreground color.

Discussion

This function operates for graphics ports defined by both the `GrafPort` and `CGrafPort` structures. If the current graphics port is defined by a `CGrafPort` structure, the returned value is taken directly from the `rgbFgColor` field.

If the current graphics port is defined by a `GrafPort` structure, then only eight possible RGB values can be returned. These eight values are determined by the values in a global variable named `QDColors`, which is a handle to a color table containing the current QuickDraw colors.

Use the `RGBForeColor` (page 389) function to change the foreground color.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial
FinderDragPro
HTMLSample
Live Scroll
vrmovies.win

Declared in

`QuickdrawAPI.h`

GetGDevice

Obtains a handle to the `GDevice` structure for the current device. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
GDHandle GetGDevice (
   void
);
```

Return Value
A handle to the current device.

Discussion
At any given time, exactly one video device is the current device—that is, the one on which drawing is actually taking place.

Color QuickDraw stores a handle to the current device in the global variable `TheGDevice`.

All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

You can also use the `GetGWorld` function to get a handle to the `GDevice` structure for the current device.

Special Considerations
The `GetGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DrawSprocketTestOld

GlyphaIVOld

Declared in
`QuickdrawAPI.h`

GetGWorld

Saves the current graphics port (basic, color, or offscreen) and the current `GDevice` *structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)*

```
void GetGWorld (
   CGrafPtr *port,
   GDHandle *gdh
);
```

Parameters

`port`

On return, a pointer to the current graphics port in the `port` parameter. This parameter can return values of type `GrafPtr`, `CGrafPtr`, or `GWorldPtr`, depending on whether the current graphics port is a basic graphics port, color graphics port, or offscreen graphics world.

`gdh`

On return, a pointer to a handle to the `GDevice` structure for the current device.

Discussion

After using `GetGWorld` to save a graphics port and a `GDevice` structure, use the `SetGWorld` (page 410) function to restore them.

Special Considerations

The `GetGWorld` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Palette and GWorld
qteffects
qteffects.win
SGDataProcSample
vrscript.win

Declared in

`QDOffscreen.h`

GetGWorldDevice

Obtains a handle to the `GDevice` structure associated with an offscreen graphics world. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
GDHandle GetGWorldDevice (
   GWorldPtr offscreenGWorld
);
```

Parameters

`offscreenGWorld`

A pointer to an offscreen graphics world. The pointer returned to your application by the `NewGWorld` function.

Return Value

A handle to the `GDevice` structure associated with the offscreen graphics world specified by the `offscreenGWorld` parameter.

If you created the offscreen world by specifying the `noNewDevice` flag, the `GDevice` structure is for one of the screen devices or is the `GDevice` structure that you specified to `NewGWorld` or `UpdateGWorld`.

If you point to a `GrafPort` or `CGrafPort` structure in the `offscreenGWorld` parameter, `GetGWorldDevice` returns the current device.

Special Considerations

The `GetGWorldDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qtfullscreen

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared in

`QDOffscreen.h`

GetGWorldPixMap

Obtains the pixel map created for an offscreen graphics world. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PixMapHandle GetGWorldPixMap (
   GWorldPtr offscreenGWorld
);
```

Parameters

`offscreenGWorld`

A pointer to an offscreen graphics world. Pass the pointer returned to your application by the `NewGWorld` (page 328) function when you created the offscreen graphics world.

Return Value

A handle to the pixel map created for an offscreen graphics world. Your application can, in turn, pass the handle returned by `GetGWorldPixMap` as a parameter to other QuickDraw functions that accept a handle to a pixel map.

On a system running only basic QuickDraw, the `GetGWorldPixMap` function returns the handle to a 1-bit pixel map that your application can supply as a parameter to the other functions related to offscreen graphics worlds. However, your application should not supply this handle to color QuickDraw functions.

Special Considerations

To ensure compatibility on systems running basic QuickDraw instead of Color QuickDraw, use `GetGWorldPixMap` whenever you need to gain access to the bitmap created for a graphics world—that is, do not dereference the `GWorldPtr` structure for that graphics world.

Version Notes

The `GetGWorldPixMap` function is not available in systems preceding System 7. You can make sure that the `GetGWorldPixMap` function is available by using the `Gestalt` function with the `gestaltSystemVersion` selector. Test the low-order word in the `response` parameter; if the value is $0700 or greater, then `GetGWorldPixMap` is available.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qteffects

qteffects.win
qtspritesplus
qtwiredactions
qtwiredspritesjr.win

Declared in
QDOffscreen.h

GetIndPattern

Obtains a pattern stored in a pattern list ('PAT#') resource. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void GetIndPattern (
   Pattern *thePat,
   short patternListID,
   short index
);
```

Parameters

`thePat`

On return, a pointer to a `Pattern` (page 100) structure for the pattern stored in the specified pattern list resource.

`patternListID`

The resource ID for a resource of type `'PAT#'`.

`index`

The index number for the desired pattern within the pattern list (`'PAT#'`) resource. The index number can range from 1 to the number of patterns in the pattern list resource.

Discussion

The `GetIndPattern` function calls the following Resource Manager function with these parameters:

```
GetResource('PAT#', patternListID);
```

There is a pattern list resource in the System file that contains the standard Macintosh patterns used by MacPaint. The resource ID is represented by the constant `sysPatListID`.

Special Considerations

The `GetIndPattern` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial

Declared in
`QuickdrawAPI.h`

GetMainDevice

Obtains a handle to the `GDevice` structure for the main screen. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
GDHandle GetMainDevice (
   void
);
```

Return Value
A handle to the device for the main screen, which is the device containing the menu bar.

Discussion
A handle to the main device is kept in the global variable `MainDevice`.

All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations
The `GetMainDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
AlwaysPreview
HTMLSample
Palette and GWorld
qtdataexchange.win
SGDataProcSample

Declared in
`QuickdrawAPI.h`

GetMaskTable

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Ptr GetMaskTable (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetMaxDevice

Obtains a handle to the `GDevice` structure for the video device with the greatest pixel depth. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
GDHandle GetMaxDevice (
   const Rect *globalRect
);
```

Parameters
`globalRect`
A rectangle, in global coordinates, that intersects the graphics devices that you are searching to find the one with the greatest pixel depth.

Return Value
A handle to the device with the greatest pixel depth.

Discussion
All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations
The `GetMaxDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
HTMLSample
qteffects.win
qtspritesplus
vrmovies.win

Declared in
`QuickdrawAPI.h`

GetNextDevice

Returns a handle to the next `GDevice` structure in the device list. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
GDHandle GetNextDevice (
   GDHandle curDevice
);
```

Parameters
`curDevice`

A handle to the `GDevice` structure at which you want the search to begin.

Return Value

A handle to the next device. If there are no more `GDevice` structures in the list, `NULL`.

Discussion

After using any of the functions `GetDeviceList` (page 233) , `GetGDevice` (page 235) , `GetMainDevice` (page 240) , or `GetMaxDevice` (page 241) to obtain a handle to a `GDevice` structure, use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations

The `GetNextDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

Declared in

`QuickdrawAPI.h`

GetPattern

Obtains a pattern (`'PAT'`) resource stored in a resource file. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PatHandle GetPattern (
   short patternID
);
```

Parameters

`patternID`

The resource ID for a resource of type ‘PAT’.

Return Value

a handle to the pattern having the resource ID that you specify in the `patID` parameter. If a pattern resource with the ID that you request does not exist, the `GetPattern` function returns `NULL`.

Discussion

The `GetPattern` function calls the following Resource Manager function with these parameters:

GetResource('PAT', patID);

When you are finished using the pattern, dispose of its handle with the Memory Manager function `DisposeHandle`.

Special Considerations

The `GetPattern` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

GetPen

Determines the location of the graphics pen. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void GetPen (
   Point *pt
);
```

Parameters

`pt`

On return, a pointer to the graphics pen's current position in the current graphics port. The point returned is in the local coordinates of the current graphics port.

Discussion

In the pt parameter, the GetPen procedure returns the current pen position. The point returned is in the local coordinates of the current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
kcapApp
QTMusicToo
ReKeyTrans

Declared in
`QuickdrawAPI.h`

GetPenState

Determines the graphics pen's location, size, pattern, and pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void GetPenState (
   PenState *pnState
);
```

Parameters
`pnState`
On return, a pointer to a `PenState` structure holding information about the graphics pen. The `GetPenState` function saves the location, size, pattern, and pattern mode of the graphics pen for the current graphics port in this structure.

Discussion
After changing the graphics pen as necessary, restore these pen states with the `SetPenState` (page 413) function.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DialogsToHIViews

FinderDragPro
HTMLSample
qtmovietrack.win
QTMusicToo

Declared in
`QuickdrawAPI.h`

GetPicture

Obtains a handle to a picture stored in a `'PICT'` resource. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
PicHandle GetPicture (
   short pictureID
);
```

Parameters
`pictureID`
The resource ID for a `'PICT'` resource.

Return Value
A handle to the picture in the specified 'PICT' resource. To draw the picture stored in the resource, pass this handle to the `DrawPicture` (page 192) function. If the resource cannot be read, `GetPicture` returns `NULL`.

Discussion
The `GetPicture` function calls the Resource Manager function `GetResource` as follows:

```
GetResource('PICT', picID)
```

Special Considerations
To release the memory occupied by a picture stored in a `'PICT'` resource, use the Resource Manager function `ReleaseResource`.

The `GetPicture` function may move or purge memory.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
HTMLSample
MovieBrowser
NewCCursor
qteffects.win
Tiler

Declared in
`QuickdrawAPI.h`

GetPixBaseAddr

Obtains a pointer to an offscreen pixel map. (Available in OS X v10.0 through OS X v10.6.)

```
Ptr GetPixBaseAddr (
   PixMapHandle pm
);
```

Parameters
`pm`
A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 238) function.

Return Value
A 32-bit pointer to the beginning of a pixel image. If the offscreen buffer has been purged, `GetPixBaseAddr` returns `NULL`.

Discussion
The `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer, which is what the `baseAddr` field for an onscreen pixel map contains. You must use the `GetPixBaseAddr` function to obtain a pointer to the `PixMap` structure for an offscreen graphics world.

Your application should never directly access the `baseAddr` field of the `PixMap` structure for an offscreen graphics world; instead, always use `GetPixBaseAddr`. If your application is using 24-bit mode, use the `PixMap32Bit` (page 367) function to determine whether a pixel map requires 32-bit addressing mode for access to its pixel image.

Special Considerations
Any QuickDraw functions that your application uses after calling `GetPixBaseAddr` may change the base address for the offscreen pixel image.

The `GetPixBaseAddr` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CTMClip

QTCarbonShell

SGDataProcSample

vrscript

vrscript.win

Declared in
`QDOffscreen.h`

GetPixBounds

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Rect * GetPixBounds (
   PixMapHandle pixMap,
   Rect *bounds
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CocoaVideoFrameToGWorld

CocoaVideoFrameToNSImage

SGDataProcSample

Declared in
QuickdrawAPI.h

GetPixDepth

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
short GetPixDepth (
   PixMapHandle pixMap
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
MungSaver

Declared in
QuickdrawAPI.h

GetPixel

Determines whether the pixel associated with a point is black or white. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean GetPixel (
   short h,
   short v
);
```

Parameters
h

The horizontal coordinate of the point for the pixel to be tested.

v

The vertical coordinate of the point for the pixel to be tested.

Return Value
Returns TRUE if the pixel is black or FALSE if it is white.

Discussion
The selected pixel is immediately below and to the right of the point whose coordinates you supply in the h and v parameters, in the local coordinates of the current graphics port. There's no guarantee that the specified pixel actually belongs to the current graphics port, however it may have been drawn in a graphics port overlapping the current one. To see if the point indeed belongs to the current graphics port, you could use the PtInRgn function to test whether the point is in the current graphics port's visible region.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QuickdrawAPI.h

GetPixelsState

Saves the current information about the memory allocated for an offscreen pixel image. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
GWorldFlags GetPixelsState (
   PixMapHandle pm
);
```

Parameters
pm

A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the GetGWorldPixMap (page 238) function.

Return Value
Information about the memory allocated for the base address for an offscreen pixel image. This result can be either of the constants, pixelsPurgeable or pixelsLocked. If the pixelsPurgeable flag is not returned, then the base address for the offscreen pixel image is unpurgeable. If the pixelsLocked flag is not returned, then the base address for the offscreen pixel image is unlocked.

Discussion

After using `GetPixelsState` to save this state information, use the `SetPixelsState` (page 414) function to restore this state to the offscreen graphics world.

After using `GetPixelsState` and before using `SetPixelsState`, temporarily use the `AllowPurgePixels` (page 149) function to make the base address for an offscreen pixel image purgeable, the `NoPurgePixels` (page 339) function to make it unpurgeable, the `LockPixels` (page 314) function to prevent it from being moved, and the `UnlockPixels` (page 451) function to allow it to be moved.

Special Considerations

The `GetPixelsState` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qtcompress.win

qtstdcompr.win

Declared in

`QDOffscreen.h`

GetPixPat

Obtains a pixel pattern (`'ppat'`) resource stored in a resource file. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
PixPatHandle GetPixPat (
   short patID
);
```

Parameters

`patID`

The resource ID for a resource of type `'ppat'`.

Return Value

A handle to the pixel pattern having the resource ID you specify in the `patID` parameter. The `GetPixPat` function calls the following Resource Manager function with these parameters:


```
GetResource('ppat', patID);
```

If a `'ppat'` resource with the ID that you request does not exist, the `GetPixPat` function returns `NULL`.

Discussion
When you are finished with the pixel pattern, use the `DisposePixPat` (page 189) function. For more information on the pixel pattern resource, see `'ppat'`.

Pixel patterns can use colors at any pixel depth and can be of any width and height that's a power of 2. To create a pixel pattern, you typically define it in a `'ppat'` resource, which you store in a resource file. To retrieve the pixel pattern stored in a `'ppat'` resource, you can use the `GetPixPat` function.

Special Considerations
The `GetPixPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CopyBits vs. CopyMask

Declared in
`QuickdrawAPI.h`

GetPixRowBytes

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
SInt32 GetPixRowBytes (
   PixMapHandle pm
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CocoaCreateMovie
CocoaVideoFrameToNSImage
CTMClip
CTMDemo
SGDataProcSample

Declared in
`QDOffscreen.h`

GetPort

Saves the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void GetPort (
   GrafPtr *port
);
```

Parameters
`port`
On return, a pointer to a `GrafPort` structure for the current graphics port. If the current graphics port is a color graphics port, `GetPort` coerces its `CGrafPort` structure into a `GrafPort` structure.

Discussion
When your application runs in Color QuickDraw or uses offscreen graphics worlds, it should use the `GetGWorld` function instead of `GetPort`. The `GetGWorld` function saves the current graphics port for basic and color graphics ports as well as offscreen graphics worlds.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
MovieSprites
Play Movie with Controller
qtmovietrack.win

WiredSprites

Declared in
`QuickdrawAPI.h`

GetPortBackColor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
RGBColor * GetPortBackColor (
   CGrafPtr port,
   RGBColor *backColor
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortBackPixPat

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PixPatHandle GetPortBackPixPat (
   CGrafPtr port,
   PixPatHandle backPattern
);
```


Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortBitMapForCopyBits

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
const BitMap * GetPortBitMapForCopyBits (
   CGrafPtr port
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CustomWindow
GLCarbon1ContextPbuffer
GLCarbonSharedPbuffer
Out of This GWorld
Palette and GWorld

Declared in
`QuickdrawAPI.h`

GetPortBounds

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Rect * GetPortBounds (
   CGrafPtr port,
   Rect *rect
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial

HackTV Carbon

HTMLSample

qtshoweffect

qtshoweffect.win

Declared in

`QuickdrawAPI.h`

GetPortChExtra

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
short GetPortChExtra (
   CGrafPtr port
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortClipRegion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
RgnHandle GetPortClipRegion (
   CGrafPtr port,
   RgnHandle clipRgn
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CTMClip

Declared in
`QuickdrawAPI.h`

GetPortCustomXFerProc

(Available in OS X v10.0 through OS X v10.6.)

```
OSErr GetPortCustomXFerProc (
   CGrafPtr port,
   CustomXFerProcPtr *proc,
   UInt32 *flags,
   UInt32 *refCon
);
```


Return Value
A result code.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortFillPixPat

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
PixPatHandle GetPortFillPixPat (
   CGrafPtr port,
   PixPatHandle fillPattern
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortForeColor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
RGBColor * GetPortForeColor (
   CGrafPtr port,
   RGBColor *foreColor
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortFracHPenLocation

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
short GetPortFracHPenLocation (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortGrafProcs

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
CQDProcsPtr GetPortGrafProcs (
   CGrafPtr port
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites
qteffects.win
qtspritesplus
qtwiredactions
qtwiredspritesjr.win

Declared in
`QuickdrawAPI.h`

GetPortHiliteColor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
RGBColor * GetPortHiliteColor (
   CGrafPtr port,
   RGBColor *hiliteColor
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortOpColor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
RGBColor * GetPortOpColor (
   CGrafPtr port,
   RGBColor *opColor
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortPenLocation

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Point * GetPortPenLocation (
   CGrafPtr port,
   Point *penLocation
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortPenMode

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
SInt32 GetPortPenMode (
   CGrafPtr port
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CarbonCustomList

Declared in
`QuickdrawAPI.h`

GetPortPenPixPat

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PixPatHandle GetPortPenPixPat (
   CGrafPtr port,
```



```
   PixPatHandle penPattern
);
```

Return Value

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortPenSize

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Point * GetPortPenSize (
   CGrafPtr port,
   Point *penSize
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortPenVisibility

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
short GetPortPenVisibility (
   CGrafPtr port
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

GetPortPixMap

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PixMapHandle GetPortPixMap (
   CGrafPtr port
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CreateMovie

CTMClip

DesktopSprites

MungSaver
SGDataProcSample

Declared in
`QuickdrawAPI.h`

GetPortSpExtra

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Fixed GetPortSpExtra (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortTextFace

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Style GetPortTextFace (
   CGrafPtr port
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortTextFont

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
short GetPortTextFont (
   CGrafPtr port
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortTextMode

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
short GetPortTextMode (
   CGrafPtr port
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortTextSize

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
short GetPortTextSize (
   CGrafPtr port
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetPortVisibleRegion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
RgnHandle GetPortVisibleRegion (
   CGrafPtr port,
   RgnHandle visRgn
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
CreateMovie
HTMLSample
MovieSprites
WiredSprites

Declared in
`QuickdrawAPI.h`

GetQDGlobalsArrow

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Cursor * GetQDGlobalsArrow (
   Cursor *arrow
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CreateMovie
qtspritesplus
qtwiredactions
samplemakeeffectmovie
vrbackbuffer

Declared in
`QuickdrawAPI.h`

GetQDGlobalsBlack

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Pattern * GetQDGlobalsBlack (
   Pattern *black
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
HTMLSample

Declared in
`QuickdrawAPI.h`

GetQDGlobalsDarkGray

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Pattern * GetQDGlobalsDarkGray (
   Pattern *dkGray
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetQDGlobalsGray

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Pattern * GetQDGlobalsGray (
   Pattern *gray
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DialogsToHIViews

HTMLSample

Declared in
`QuickdrawAPI.h`

GetQDGlobalsLightGray

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Pattern * GetQDGlobalsLightGray (
   Pattern *ltGray
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CocoaVideoFrameToGWorld

Declared in
`QuickdrawAPI.h`

GetQDGlobalsRandomSeed

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
long GetQDGlobalsRandomSeed (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetQDGlobalsScreenBits

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
BitMap * GetQDGlobalsScreenBits (
   BitMap *screenBits
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
AGLSurfaceTexture
ConvertMovieSndTrack
HackTV Carbon
QISA
SGDataProcSample

Declared in
`QuickdrawAPI.h`

GetQDGlobalsThePort

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
CGrafPtr GetQDGlobalsThePort (
   void
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CarbonMDEF

Declared in
`QuickdrawAPI.h`

GetQDGlobalsWhite

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Pattern * GetQDGlobalsWhite (
   Pattern *white
);
```


Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

GetRegionBounds

(Available in OS X v10.0 through OS X v10.6.)

```
Rect * GetRegionBounds (
   RgnHandle region,
   Rect *bounds
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CreateMovie

MovieSprites

Play Movie with Controller

qtshoweffect

WiredSprites

Declared in
`QuickdrawAPI.h`

GetSubTable

Searches one color table for the best matches to colors in another color table. Your application should not need to call this function; it is used by system software only. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void GetSubTable (
   CTabHandle myColors,
   short iTabRes,
   CTabHandle targetTbl
);
```

Parameters

`myColors`

A handle to a color table containing the colors for which you want matches.

`iTabRes`

The resolution of the inverse table to be used.

`targetTbl`

A handle to a color table whose colors are to be matched. If you supply `NULL` for `targetTbl`, then the Color Manager searches the current `GDevice` data structure's CLUT, and uses its inverse table. Otherwise a temporary inverse table is built, with a resolution of the value in the `iTabRes` parameter.

Discussion

The Color Manager uses the `Color2Index` (page 164) function for each `RGBColor` data structure in the color table of the `myColors` parameter. It determines the best match in the target table and stores that index value in the `value` field of the color table of the `myColors` parameter.

Depending on the requested resolution, building the inverse table can require large amounts of temporary space in the application heap: twice the size of the table itself, plus a fixed overhead of 3–15 KB for each inverse table resolution.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

GlobalToLocal

Converts the coordinates of a point from global coordinates to the local coordinates of the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void GlobalToLocal (
   Point *pt
);
```

Parameters

`pt`

A pointer to a point expressed in global coordinates (where the upper-left corner of the main screen has coordinates [0,0]). On return, this point is converted to local coordinates.

Discussion

The GlobalToLocal procedure takes a point expressed in global coordinates (where the upper-left corner of the main screen has coordinates [0,0]) and converts it into the local coordinates of the current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial
Fragment Tool
Live Scroll
QTMusicToo
qtskins

Declared in

`QuickdrawAPI.h`

GrafDevice

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void GrafDevice (
   short device
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

HandleToRgn

(Available in OS X v10.0 through OS X v10.6.)

```
void HandleToRgn (
   Handle oldRegion,
   RgnHandle region
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

HideCursor

Hides the cursor if it is visible on the screen. (Available in OS X v10.0 through OS X v10.6.)

```
void HideCursor (
   void
);
```

Discussion
The `HideCursor` function removes the cursor from the screen, restores the bits under the cursor image, and decrements the cursor level (which `InitCursor` initialized to 0). You might want to use HideCursor when the user is using the keyboard to create content in one of your application's windows. Every call to `HideCursor` should be balanced by a subsequent call to the `ShowCursor` (page 432) function.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
GlyphaIVOld
Simple DrawSprocket

Declared in
`QuickdrawAPI.h`

HidePen

Makes the graphics pen invisible, so that pen drawing doesn't show on the screen. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void HidePen (
   void
);
```

Discussion
The `HidePen` function is called by the `OpenRgn` (page 350) , `OpenPicture`, and `OpenPoly` (page 349) functions so that you can create regions, pictures, and polygons without drawing on the screen.

The `HidePen` function decrements the `pnVis` field of the current graphics port. The `pnVis` field is initialized to 0 by the `OpenPort` function. Whenever `pnVis` is negative, the pen does not draw on the screen. The `pnVis` field keeps track of the number of times the pen has been hidden to compensate for nested calls to the `HidePen` and `ShowPen` functions.

Every call to `HidePen` should be balanced by a subsequent call to `ShowPen` (page 433).

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites
DesktopSprites.win
MouseTracking
MovieSprites

WiredSprites

Declared in
`QuickdrawAPI.h`

HiliteColor

Changes the highlight color for the current color graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void HiliteColor (
   const RGBColor *color
);
```

Parameters
`color`
An `RGBColor` structure that defines the highlight color.

Discussion
All drawing operations that use the `hilite` transfer mode use the highlight color. When a color graphics port is created, its highlight color is initialized from the global variable `HiliteRGB`.

If the current graphics port is a basic graphics port, `HiliteColor` has no effect.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
QTMusicToo

Declared in
`QuickdrawAPI.h`

Index2Color

Obtains the `RGBColor` data structure corresponding to an index value in the color table of the current `GDevice` data structure. Your application should not need to call this function; it is used by system software only. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void Index2Color (
   long index,
   RGBColor *aColor
);
```

Parameters

`index`

The index value whose color entry is sought; you should supply a long integer in which the high-order word is padded with zeros.

`aColor`

A pointer to the returned `RGBColor` data structure.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

GlyphaIVOld

Palette and GWorld

Declared in

`QuickdrawAPI.h`

InitCursor

Sets the cursor to the standard arrow and makes the cursor visible. (Available in OS X v10.0 through OS X v10.6.)

```
void InitCursor (
   void
);
```


Discussion

This function initializes the standard arrow cursor, sets the current cursor to the standard arrow, and makes the cursor visible. Classic Mac OS applications need to call this function when launching because the system sets the cursor to the watch cursor. Carbon applications running in Mac OS 9 or Mac OS X do not need to call this function.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial
LocalServer
MovieBrowser
QTMusicToo
SGDataProcSample

Declared in

`QuickdrawAPI.h`

InitGDevice

Initializes a `GDevice` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InitGDevice (
   short qdRefNum,
   long mode,
   GDHandle gdh
);
```

Parameters

`qdRefNum`

Reference number of the graphics device. System software sets this number at system startup time for most graphics devices.

`mode`

The device configuration mode. Used by the screen driver, this value sets the pixel depth and specifies color or black and white.

`gdh`

The handle, returned by the `NewGDevice` function, to the `GDevice` (page 91) structure to be initialized.

Discussion

The `InitGDevice` function sets the graphics device whose driver has the reference number specified in the `gdRefNum` parameter to the mode specified in the `mode` parameter. The `InitGDevice` function then fills out the `GDevice` structure, previously created with the `NewGDevice` function, to contain all information describing that mode.

The `mode` parameter determines the configuration of the device. Possible modes for a device are determined by interrogating the video device's ROM through Slot Manager functions. The information describing the device's mode is primarily contained in the video device's ROM. If the video device has a fixed color table, then that table is read directly from the ROM. If the video device has a variable color table, then `InitGDevice` uses the default color table defined in a `'clut'` resource, contained in the System file, that has a resource ID equal to the video device's pixel depth.

In general, your application should never need to call `InitGDevice`. All video devices are initialized at start time, and users change modes through the Monitors control panel.

If your program uses `NewGDevice` to create a graphics device without a driver, `InitGDevice` does nothing; instead, your application must initialize all fields of the `GDevice` structure. After your application initializes the color table for the `GDevice` structure, call the Color Manager function `MakeITable` to build the inverse table for the graphics device.

Special Considerations

The `InitGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

InsetRect

Shrinks or expands a rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void InsetRect (
   Rect * r,
   short dh,
```



```
   short dv
);
```

Parameters

`r`

A pointer to the rectangle to alter.

`dh`

The horizontal distance to move the left and right sides in toward or outward from the center of the rectangle.

`dv`

The vertical distance to move the top and bottom sides in toward or outward from the center of the rectangle.

Discussion

The `InsetRect` function shrinks or expands the rectangle that you specify in the `r` parameter: the left and right sides are moved in by the amount you specify in the `dh` parameter; the top and bottom are moved toward the center by the amount you specify in the `dv` parameter. If the value you pass in `dh` or `dv` is negative, the appropriate pair of sides is moved outward instead of inward. The effect is to alter the size by `2*dh` horizontally and `2*dv` vertically, with the rectangle remaining centered in the same place on the coordinate plane.

If the resulting width or height becomes less than 1, the rectangle is set to the empty rectangle (0,0,0,0).

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

Carbon Porting Tutorial
FinderDragPro
GlyphaIVOld
MovieBrowser
QTMusicToo

Declared in

`QuickdrawAPI.h`

InsetRgn

Shrinks or expands a region. (Available in OS X v10.0 through OS X v10.6.)

```
void InsetRgn (
   RgnHandle rgn,
   short dh,
   short dv
);
```

Parameters

`rgn`

A handle to the region to alter.

`dh`

The horizontal distance to move points on the left and right boundaries in toward or outward from the center.

`dv`

The vertical distance to move points on the top and bottom boundaries in toward or outward from the center.

Discussion

The `InsetRgn` function moves all points on the region boundary of the region whose handle you pass in the `rgn` parameter inward by the vertical distance that you specify in the `dv` parameter and by the horizontal distance that you specify in the `dh` parameter. If you specify negative values for `dh` or `dv`, the `InsetRgn` function moves the points outward in that direction.

The `InsetRgn` function leaves the region's center at the same position, but moves the outline in (for positive values of `dh` and `dv`) or out (for negative values of `dh` and `dv`). Using `InsetRgn` on a rectangular region has the same effect as using the `InsetRect` function.

Special Considerations

The `InsetRgn` function temporarily uses heap space that's twice the size of the original region.

The `InsetRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

CustomWindow

FinderDragPro
Fragment Tool

Declared in
`QuickdrawAPI.h`

InvertArc

Inverts the pixels of a wedge. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void InvertArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

`r`

The rectangle that defines an oval's boundaries.

`startAngle`

The angle indicating the start of the arc.

`arcAngle`

The angle indicating the arc's extent.

Discussion

The `InvertArc` function inverts the pixels enclosed by a wedge of the oval bounded by the rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. As in `FrameArc` (page 218) , use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `InvertArc` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct devices or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed

device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

InvertColor

Finds the complement of an `RGBColor` data structure. This function is used only by system software. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void InvertColor (
   RGBColor *myColor
);
```

Parameters
`myColor`

A pointer to the `RGBColor` data structure for which the complement is to be found. The `InvertColor` function returns the complement of an absolute color, using the list of complement functions in the current device data structure. The default complement function uses the one's complement of each component of the given color.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

InvertOval

Inverts the pixels enclosed by an oval. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvertOval (
   const Rect *r
);
```

Parameters

`r`

The rectangle that defines the oval's boundary.

Discussion

The `InvertOval` function inverts the pixels enclosed by an oval just inside the bounding rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. The pen location does not change.

Special Considerations

The `InvertOval` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct devices or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

InvertPoly

Inverts the pixels enclosed by a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void InvertPoly (
   PolyHandle poly
);
```

Parameters
`poly`

A handle to a polygon, the pixels of which you want to invert. The `OpenPoly` (page 349) function returns this handle when you first create the polygon.

Discussion
The `InvertPoly` function inverts the pixels enclosed by the polygon whose handle you pass in the `poly` parameter. Every white pixel becomes black and every black pixel becomes white.

This function leaves the location of the graphics pen unchanged.

`InvertPoly` temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 319) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Special Considerations

The `InvertPoly` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with 1-bit or direct pixels. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

InvertRect

Inverts the pixels enclosed by a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void InvertRect (
   const Rect * r
);
```

Parameters

`r`

The rectangle whose enclosed pixels are to be inverted.

Discussion

The `InvertRect` function inverts the pixels enclosed by the rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. The pen location does not change.

Special Considerations

The `InvertRect` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct pixels or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

PopUpMenuWithCurFont
QTMusicToo
SICN LDEF
StarMenu

Declared in

`QuickdrawAPI.h`

InvertRgn

Inverts the pixels enclosed by a region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvertRgn (
   RgnHandle rgn
);
```


Parameters

`rgn`

A handle to the region whose pixels are to invert.

Discussion

The `InvertRgn` function inverts the pixels enclosed by the region whose handle you pass in the `rgn` parameter. Every white pixel becomes black and every black pixel becomes white.

This function leaves the location of the graphics pen unchanged.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Special Considerations

The `InvertRgn` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with 1-bit or direct pixels. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

QTMusicToo

Declared in
`QuickdrawAPI.h`

InvertRoundRect

Inverts the pixels enclosed by a rounded rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void InvertRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

`r`

The rectangle that defines the rounded rectangle's boundaries.

`ovalWidth`

The width of the oval defining the rounded corner.

`ovalHeight`

The height of the oval defining the rounded corner.

Discussion

The `InvertRoundRect` function inverts the pixels enclosed by the rounded rectangle bounded by the rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. The `ovalWidth` and `ovalHeight` parameters specify the diameters of curvature for the corners. The pen location does not change.

Special Considerations

The `InvertRoundRect` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct devices or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertRoundRect` function may move or purge memory blocks in the application; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

InvokeDeviceLoopDrawingUPP

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void InvokeDeviceLoopDrawingUPP (
   short depth,
   short deviceFlags,
   GDHandle targetDevice,
   SRefCon userData,
   DeviceLoopDrawingUPP userUPP
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Declared in
`QuickdrawTypes.h`

IsPortColor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean IsPortColor (
   CGrafPtr port
);
```


Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsPortOffscreen

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean IsPortOffscreen (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsPortPictureBeingDefined

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean IsPortPictureBeingDefined (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsPortPolyBeingDefined

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean IsPortPolyBeingDefined (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsPortRegionBeingDefined

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean IsPortRegionBeingDefined (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsRegionRectangular

(Available in OS X v10.0 through OS X v10.6.)

```
Boolean IsRegionRectangular (
   RgnHandle region
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsValidPort

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean IsValidPort (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

KillPicture

Releases the memory occupied by a picture not stored in a `'PICT'` resource. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void KillPicture (
```



```
   PicHandle myPicture
);
```

Parameters

`myPicture`

A handle to the picture whose memory can be released.

Discussion

Use this function only when you are completely finished with a picture.

Special Considerations

If you use the Window Manager function `SetWindowPic` to store a picture handle in the window structure, use the Window Manager function `DisposeWindow` or `CloseWindow` to release the memory allocated to the picture. These functions automatically call `KillPicture` for the picture.

If the picture is stored in a `'PICT'` resource, use the Resource Manager function `ReleaseResource` instead of `KillPicture`. The Window Manager functions `DisposeWindow` and `CloseWindow` will not delete it. Instead, call `ReleaseResource` before calling `DisposeWindow` or `CloseWindow`.

The `KillPicture` function may move or purge memory.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

AlwaysPreview

qdmediamaker.win

qteffects

qteffects.win

qtspritesplus

Declared in

`QuickdrawAPI.h`

KillPoly

Releases the memory occupied by a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void KillPoly (
   PolyHandle poly
);
```

Parameters

`poly`

A handle to the polygon to dispose of.

Discussion

Use `KillPoly` only when you are completely through with a polygon.

Special Considerations

The `KillPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

MouseTracking

Declared in

`QuickdrawAPI.h`

Line

Draws a line a specified distance from the graphics pen's current location in the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void Line (
   short dh,
   short dv
);
```

Parameters

`dh`

The horizontal distance of the graphics pen's movement.

`dv`

The vertical distance of the graphics pen's movement.

Discussion

Starting at the current location of the graphics pen, the `Line` function draws a line the horizontal distance that you specify in the `dh` parameter and the vertical distance that you specify in the `dv` parameter. The `Line` function calls

LineTo(h+dh,v+dv)

where `(h,v)` is the current location in local coordinates. The pen location becomes the coordinates of the end of the line after the line is drawn. If you are using `Line` to draw a region or polygon, its outline is infinitely thin and is not affected by the values of the `pnSize`, `pnMode`, and `pnPat` fields of the graphics port.

Special Considerations

The `Line` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Palette and GWorld
qtmovietrack.win
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

LineTo

Draws a line from the graphics pen's current location to a new location. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void LineTo (
   short h,
   short v
);
```


Parameters

`h`

The horizontal coordinate of the graphics pen's new location.

`v`

The vertical coordinate of the graphics pen's new location.

Discussion

The `LineTo` function draws a line from the graphics pen's current location in the current graphics port to the new location `(h,v)`, which you specify in the local coordinates of the current graphics port. If you are using `LineTo` to draw a region or polygon, its outline is infinitely thin and is not affected by the values of the `pnSize`, `pnMode`, or `pnPat` field of the graphics port.

Special Considerations

The `LineTo` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

ATSUICurveAccessDemo
CarbonQuartzDrawingWPrinting
MovieBrowser
Out of This GWorld
QTMusicToo

Declared in

`QuickdrawAPI.h`

LMGetCursorNew

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean LMGetCursorNew (
   void
);
```


Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetDeviceList

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
GDHandle LMGetDeviceList (
   void
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetFractEnable

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
UInt8 LMGetFractEnable (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetHiliteMode

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
UInt8 LMGetHiliteMode (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetHiliteRGB

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMGetHiliteRGB (
   RGBColor *hiliteRGBValue
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial

Declared in
`QuickdrawAPI.h`

LMGetLastFOND

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Handle LMGetLastFOND (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetLastSPExtra

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
SInt32 LMGetLastSPExtra (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetMainDevice

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
GDHandle LMGetMainDevice (
   void
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetQDColors

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Handle LMGetQDColors (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetScrHRes

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
SInt16 LMGetScrHRes (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetScrVRes

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
SInt16 LMGetScrVRes (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetTheGDevice

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
GDHandle LMGetTheGDevice (
   void
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetWidthListHand

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Handle LMGetWidthListHand (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetWidthPtr

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Ptr LMGetWidthPtr (
```



```
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMGetWidthTabHandle

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Handle LMGetWidthTabHandle (
   void
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetCursorNew

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetCursorNew (
   Boolean value
);
```


Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetDeviceList

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void LMSetDeviceList (
   GDHandle value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetFractEnable

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void LMSetFractEnable (
   UInt8 value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetHiliteMode

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void LMSetHiliteMode (
   UInt8 value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetHiliteRGB

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void LMSetHiliteRGB (
   const RGBColor *hiliteRGBValue
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetLastFOND

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetLastFOND (
   Handle value
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

LMSetLastSPExtra

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetLastSPExtra (
   SInt32 value
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

PopUpMenuWithCurFont

Declared in

`QuickdrawAPI.h`

LMSetMainDevice

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetMainDevice (
   GDHandle value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetQDColors

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetQDColors (
   Handle value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetScrHRes

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void LMSetScrHRes (
   SInt16 value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetScrVRes

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetScrVRes (
   SInt16 value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetTheGDevice

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetTheGDevice (
   GDHandle value
);
```


Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetWidthListHand

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetWidthListHand (
   Handle value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetWidthPtr

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetWidthPtr (
   Ptr value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LMSetWidthTabHandle

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LMSetWidthTabHandle (
   Handle value
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

LocalToGlobal

Converts a point's coordinates from the local coordinates of the current graphics port (basic or color) to global coordinates. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void LocalToGlobal (
   Point *pt
);
```

Parameters
`pt`
A pointer to a point in local coordinates. On return, this point is converted to global coordinates.

Discussion

The `LocalToGlobal` function converts the given point from the current graphics port's local coordinate system into the global coordinate system (where the upper-left corner of the main screen has coordinates [0,0]). This global point can then be compared to other global points, or it can be changed into the local coordinates of another graphics port.

Because a rectangle is defined by two points, you can convert a rectangle into global coordinates with two calls to `LocalToGlobal`. In conjunction with `LocalToGlobal`, you can use the `OffsetRect`, `OffsetRgn`, or `OffsetPoly` functions to convert a rectangle, region, or polygon into global coordinates.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial
FinderDragPro
Fragment Tool
HTMLSample
QTMusicToo

Declared in

`QuickdrawAPI.h`

LockPixels

Prevents the base address for an offscreen pixel image from being moved while you draw into or copy from its pixel map. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean LockPixels (
   PixMapHandle pm
);
```

Parameters

`pm`

A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 238) function .

Return Value

If the base address for an offscreen pixel image hasn't been purged by the Memory Manager or is not purgeable, `LockPixels` returns `TRUE` as its function result, and your application can draw into or copy from the offscreen pixel map. However, if the base address for an offscreen pixel image has been purged, `LockPixels` returns `FALSE` to indicate that you can perform no drawing to or copying from the pixel map. At that point, your application should either call the `UpdateGWorld` (page 454) function to reallocate the offscreen pixel image and then reconstruct it, or draw directly in a window instead of preparing the image in an offscreen graphics world.

Discussion

You must call `LockPixels` before drawing to or copying from an offscreen graphics world.

The `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer (which is what the `baseAddr` field for an onscreen pixel map contains). The `LockPixels` function dereferences the `PixMap` handle into a pointer. When you use the `UnlockPixels` function the handle is recovered.

As soon as you are finished drawing into and copying from the offscreen pixel image, call the `UnlockPixels` (page 451) function.

Special Considerations

The `LockPixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qteffects

qteffects.win

vrmakepano

vrscript

vrscript.win

Declared in

`QDOffscreen.h`

LockPortBits

Acquires an exclusive lock on the back buffer for a Carbon window. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
OSErr LockPortBits (
   GrafPtr port
);
```

Parameters

`port`

A window port.

Return Value

A result code. If `noErr`, the window's back buffer is locked and available for direct access.

Discussion

In Mac OS X, a Carbon window's port bits are in a back buffer shared by the application and the Quartz compositor (sometimes called the window server). When an application needs to update this buffer, the Quartz compositor must be locked out temporarily. You can use this function together with `UnlockPortBits` (page 453) to acquire and release an exclusive lock.

If you're using QuickDraw or Quartz 2D to draw in a window, you do not need to call this function—buffer locks are handled for you automatically. If you're writing code that reads or modifies the port bits directly, you should bracket your code with calls to this function and `UnlockPortBits` (page 453).

Nested calls to this function for the same port are permitted. For a given port, if you call `LockPortBits` *n* times, the lock is actually released after the *n*th balancing call to `UnlockPortBits` (page 453).

You should not call any QuickTime functions while holding the lock. To avoid degrading the user experience, you should release the lock as quickly as possible.

In Mac OS 9, this function does nothing and returns `noErr`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared in
`QuickdrawAPI.h`

MakeITable

Generates an inverse table for a color table. Your application should not need to call this function; it is used by system software only. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void MakeITable (
   CTabHandle cTabH,
   ITabHandle iTabH,
   short res
);
```

Parameters

`cTabH`

The color table for which an inverse table is to be generated. Passing `NULL` substitutes an appropriate handle from the current `GDevice` data structure.

`iTabH`

The generated inverse table. Passing `NULL` substitutes an appropriate handle from the current `GDevice` data structure.

`res`

The resolution needed for the inverse table. Passing 0 substitutes the current `GDevice` data structure's inverse table resolution.

Discussion

The `MakeITable` function generates an inverse table based on the current contents of the color table pointed to by the `cTabH` parameter, with a resolution specified by the value in the `res` parameter. Reserved color table pixel values are not included in the resulting color table. `MakeITable` tests its input parameters and returns an error in `QDError` if the resolution is less than three or greater than five.

This function allows maximum precision in mapping colors, even if colors in the color table differ by less than the resolution of the inverse table. Five-bit inverse tables are not needed when drawing in normal Color QuickDraw modes. However, Color QuickDraw transfer modes such as add, subtract, and blend may require a 5-bit inverse table for best results with certain color tables. The `'mitq'` resource governs how much memory is allocated for temporary internal structures; this resource type is for internal use only.

Depending on the requested resolution, building the inverse table can require large amounts of temporary space in the application heap: twice the size of the table itself, plus a fixed overhead of 3–15 KB for each inverse table resolution.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

MakeRGBPat

Creates the appearance of otherwise unavailable colors on indexed devices. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void MakeRGBPat (
   PixPatHandle pp,
   const RGBColor *myColor
);
```

Parameters
`pp`

On return, a handle to the generated pixel pattern.

`myColor`

An `RGBColor` structure that defines the color you want to approximate.

Discussion
The `MakeRGBPat` function generates a `PixPat` (page 106) structure that approximates the color you specify in the `myColor` parameter. For example, if your application draws to an indexed device that supports 4 bits per pixel, you only have 16 colors available if you simply set the foreground color and draw. If you use `MakeRGBPat` to create a pattern, and then draw using that pattern, you effectively get 125 different colors. If the graphics device has 8 bits per pixel, you effectively get 2197 colors. (More color are theoretically possible; this implementation opted for a fast pattern selection rather than the best possible pattern selection.)

For a pixel pattern, the (** `patMap).bounds` field of the `PixPat` structure always contains the values (0,0,8,8), and the (** `patMap).rowbytes` field equals 2.

Because patterns produced with `MakeRGBPat` aren't usually solid—they provide a selection of colors by alternating between colors, with up to four colors in a pattern— lines that are only one pixel wide may not look good.

When `MakeRGBPat` creates a `ColorTable` structure, it fills in only the `rgb` fields of its `ColorSpec` structures; the `value` fields are computed at the time the drawing actually takes place, using the current pixel depth for the system.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

MapPoly

Maps and scales a polygon within one rectangle to another rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void MapPoly (
   PolyHandle poly,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

`poly`

A handle to a polygon. Upon input, this is the polygon to map. Upon completion, this polygon is the one mapped to a new location.

`srcRect`

The rectangle containing the polygon.

`dstRect`

The rectangle in which the new region will be mapped.

Discussion
The `MapPoly` function takes a polygon within one rectangle and maps and scales it to another rectangle. In the `poly` parameter, you specify a handle to a polygon that lies within the rectangle that you specify in the `srcRect` parameter. By calling the `MapPt` function to map all the points that define the polygon specified in the `poly` parameter, `MapPoly` maps and scales it to the rectangle that you specify in the `dstRect` parameter. The `MapPoly` function returns the result in the polygon whose handle you initially passed in the `poly` parameter.

Similar to the `MapRgn` function described in the previous section, the `MapPoly` function is useful for determining whether a polygon operation will exceed available memory.

Special Considerations
If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

MapPt

Maps a point in one rectangle to an equivalent position in another rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void MapPt (
   Point *pt,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

`pt`

Upon input, a pointer to the point in the source rectangle to map; upon completion, a pointer to its mapped position in the destination rectangle.

`srcRect`

The source rectangle containing the original point.

`dstRect`

The destination rectangle in which the point will be mapped.

Discussion

The `MapPt` function maps a point in one rectangle to an equivalent position in another rectangle.

In the `pt` parameter, you specify a point that lies within the rectangle that you specify in the `srcRect` parameter. The `MapPt` function maps this point to a similarly located point within the rectangle that you specify in the `dstRect` parameter—that is, to where it would fall if it were part of a drawing being expanded or shrunk to fit the destination rectangle. The `MapPt` function returns the location of the mapped point in the `pt` parameter. For example, a corner point of the source rectangle would be mapped to the corresponding corner point of the destination rectangle in `dstRect`, and the center of the source rectangle would be mapped to the center of destination rectangle.

The source and destination rectangles may overlap, and the point you specify need not actually lie within the source rectangle.

If you are going to draw inside the destination rectangle, you'll probably also want to scale the graphics pen size accordingly with `ScalePt` (page 391).

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawAPI.h`

MapRect

Maps and scales a rectangle within one rectangle to another rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void MapRect (
   Rect *r,
   const Rect *srcRect,
   const Rect *dstRect
);
```


Parameters

`r`

Upon input, a pointer to the rectangle to map; upon completion, the mapped rectangle.

`srcRect`

The rectangle containing the rectangle to map.

`dstRect`

The rectangle in which the new rectangle will be mapped.

Discussion

The `MapRect` function takes a rectangle within one rectangle and maps and scales it to another rectangle. In the `r` parameter, you specify a rectangle that lies within the rectangle that you specify in the `srcRect` parameter. By calling the `MapPt` function to map the upper-left and lower-right corners of the rectangle in the `r` parameter, `MapRect` maps and scales it to the rectangle that you specify in the `dstRect` parameter. The `MapRect` function returns the newly mapped rectangle in the `r` parameter.

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

hacktv

HackTV Carbon

hacktv.win

HTMLUserPane

Declared in

`QuickdrawAPI.h`

MapRgn

Maps and scales a region within one rectangle to another rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void MapRgn (
```



```
   RgnHandle rgn,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

`rgn`

A handle to a region. Upon input, this is the region to map. Upon completion, this region is the one mapped to a new location.

`srcRect`

The rectangle containing the region to map.

`dstRect`

The rectangle in which the new region will be mapped.

Discussion

The `MapRgn` function takes a region within one rectangle and maps and scales it to another rectangle. In the `rgn` parameter, you specify a handle to a region that lies within the rectangle that you specify in the `srcRect` parameter. By calling the `MapPt` function to map all the points of the region in the `rgn` parameter, `MapRgn` maps and scales it to the rectangle that you specify in the `dstRect` parameter. The `MapRgn` function returns the result in the region whose handle you initially passed in the `rgn` parameter.

The `MapRgn` function is useful for determining whether a region operation will exceed available memory. By mapping a large region into a smaller one and performing the operation (without actually drawing), you can estimate how much memory will be required by the anticipated operation.

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

The `MapRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

CustomWindow

Declared in
`QuickdrawAPI.h`

Move

Moves the graphics pen a particular distance. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void Move (
   short dh,
   short dv
);
```

Parameters
`dh`

The horizontal distance of the graphics pen's movement.

`dv`

The vertical distance of the graphics pen's movement.

Discussion
The `Move` function moves the graphics pen from its current location in the current graphics port a horizontal distance that you specify in the `dh` parameter and a vertical distance that you specify in the `dv` parameter. The `Move` function calls

MoveTo(h+dh,v+dv)

where `(h,v)` is the graphics pen's current location in local coordinates. The `Move` function performs no drawing.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
kcapApp

QTMusicToo

Declared in
`QuickdrawAPI.h`

MovePortTo

Changes the position of the port rectangle of the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void MovePortTo (
   short leftGlobal,
   short topGlobal
);
```

Parameters

`leftGlobal`

The horizontal distance to move the port rectangle.

`topGlobal`

The vertical distance to move the port rectangle.

Discussion

The `MovePortTo` function is normally called only by the Window Manager. The `MovePortTo` function changes the position of the current graphics port's port rectangle: the `leftGlobal` and `topGlobal` parameters set the distance between the upper-left corner of the boundary rectangle and the upper-left corner of the new port rectangle.

This does not affect the screen; it merely changes the location at which subsequent drawing inside the graphics port appears. Like the `PortSize` function, `MovePortTo` doesn't change the clipping or visible region, nor does it affect the local coordinate system of the graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

StarMenu

Declared in

`QuickdrawAPI.h`

MoveTo

Moves the graphics pen to a particular location in the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void MoveTo (
   short h,
   short v
);
```

Parameters

`h`

The horizontal coordinate of the graphics pen's new position.

`v`

The vertical coordinate of the graphics pen's new position.

Discussion

The `MoveTo` function changes the graphics pen's current location to the new horizontal coordinate you specify in the `h` parameter and the new vertical coordinate you specify in the `v` parameter. Specify the new location in the local coordinates of the current graphics port. The `MoveTo` function performs no drawing.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CarbonQuartzDrawingWPrinting
GlyphaIVOld
Palette and GWorld
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

NewDeviceLoopDrawingUPP

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
DeviceLoopDrawingUPP NewDeviceLoopDrawingUPP (
   DeviceLoopDrawingProcPtr userRoutine
);
```


Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Declared in
`QuickdrawTypes.h`

NewGDevice

Creates a new `GDevice` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
GDHandle NewGDevice (
   short refNum,
   long mode
);
```

Parameters
`refNum`

Reference number of the graphics device for which you are creating a `GDevice` structure. For most video devices, this information is set at system startup.

`mode`

The device configuration mode. Used by the screen driver, this value sets the pixel depth and specifies color or black and white.

Return Value
A handle to the new `GDevice` (page 91) structure. If the request is unsuccessful, `NewGDevice` returns `NULL`.

Discussion
Generally, you do not need to use `NewGDevice`, because Color QuickDraw uses this function to create `GDevice` structures for your application automatically. When the system starts up, it allocates and initializes one handle to a `GDevice` structure for each video device it finds. When you use the `NewGWorld` function, QuickDraw automatically creates a `GDevice` structure for the new offscreen graphics world.

For the graphics device whose driver is specified in the `refNum` parameter and whose mode is specified in the `mode` parameter, the `NewGDevice` function allocates a new `GDevice` structure and all of its handles, and then calls the `InitGDevice` function to initialize the structure.

NewGDevice allocates the new GDevice structure and all of its handles in the system heap, and the NewGDevice function sets all attributes in the gdFlags field of the GDevice structure to FALSE. If your application creates a GDevice structure, use the SetDeviceAttribute (page 406) function to change the flag bits in the gdFlags field of the GDevice structure to TRUE. Your application should never directly change the gdFlags field of the GDevice structure. Instead, use only the SetDeviceAttribute function.

If your application creates a GDevice structure without a driver, set the mode parameter to –1. In this case, InitGDevice cannot initialize the GDevice structure, so your application must perform all initialization of the structure. A GDevice structure's default mode is defined as 128. This is assumed to be a black-and-white mode. If you specify a value other than 128 in the mode parameter, the structure's gdDevType bit in the gdFlags field of the GDevice structure is set to TRUE to indicate that the graphics device is capable of displaying color.

The NewGDevice function does not automatically insert the GDevice structure into the device list. In general, your application should not create GDevice structures, and if it ever does, it should never add them to the device list.

If your program uses NewGDevice to create a graphics device without a driver, InitGDevice does nothing; instead, your application must initialize all fields of the GDevice structure. After your application initializes the color table for the GDevice structure, call the Color Manager function MakeITable to build the inverse table for the graphics device.

Special Considerations
The NewGDevice function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QuickdrawAPI.h

NewGWorld

Creates an offscreen graphics world. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
QDErr NewGWorld (
   GWorldPtr *offscreenGWorld,
   short PixelDepth,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

`offscreenGWorld`

On return, a pointer to the offscreen graphics world created by this function. You use this pointer when referring to this new offscreen world in other QuickDraw functions.

`PixelDepth`

The pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. The default parameter (0) uses the pixel depth of the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify 0 in this parameter, `NewGWorld` also uses the `GDevice` structure from this device instead of creating a new `GDevice` structure for the offscreen world. If you use `NewGWorld` on a computer that supports only basic QuickDraw, you may specify only 0 or 1 in this parameter.

`boundsRect`

The boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if `NewGWorld` creates one. If you specify 0 in the `pixelDepth` parameter, `NewGWorld` interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. `NewGWorld` then uses the pixel depth, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

`cTable`

A handle to a `ColorTable` structure. If you pass `NULL` in this parameter, `NewGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter. If you set the `pixelDepth` parameter to 0, `NewGWorld` ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use `NewGWorld` on a computer that supports only basic QuickDraw, you may specify only `NULL` in this parameter.

`aGDevice`

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case `NewGWorld` attaches this `GDevice` structure to the new offscreen graphics world. If you set the `pixelDepth` parameter to 0, or if you do not set the `noNewDevice` flag, `NewGWorld` ignores the `aGDevice` parameter, so set it to `NULL`. If you set the `pixelDepth` parameter to 0, `NewGWorld` uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NULL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

`flags`

Options available to your application. You can set a combination of the flags `pixPurge`, `noNewDevice`, `useTempMem`, and `keepLocal`. If you don't wish to use any of these flags, specify 0 in this parameter to accept the default behavior for `NewGWorld`. The default behavior creates an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, it uses an existing `GDevice` structure (if you pass 0 in the `depth` parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. See "Graphics World Flags" (page 129) for a description of the values you can use here.

Return Value

A result code.

Discussion

Typically, you pass 0 in the `pixelDepth` parameter, a window's port rectangle in the `boundsRect` parameter, `NULL` in the `cTable` and `aGDevice` parameters, and in the `flags` parameter a 0. This provides your application with the default behavior of `NewGWorld`, and it supports computers running basic QuickDraw. This also allows QuickDraw to optimize the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions when your application copies the image in an offscreen graphics world into an onscreen graphics port.

The `NewGWorld` function allocates memory for an offscreen graphics port and its pixel map. On computers that support only basic QuickDraw, `NewGWorld` creates a 1-bit pixel map that your application can manipulate using other relevant functions described in this chapter. Your application can copy this 1-bit pixel map into basic graphics ports.

Unless you specify 0 in the `pixelDepth` parameter–or pass the `noNewDevice` flag in the `flags` parameter and supply a `GDevice` structure in the `aGDevice` parameter– `NewGWorld` also allocates a new offscreen `GDevice` structure.

When creating an image, use the `NewGWorld` function to create an offscreen graphics world that is optimized for an image's characteristics—for example, its best pixel depth. After creating the image, use the `CopyBits`, `CopyMask`, or `CopyDeepMask` function to copy that image to an onscreen graphics port. Color QuickDraw

automatically renders the image at the best available pixel depth for the screen. Creating an image in an offscreen graphics port and then copying it to the screen in this way prevents the visual choppiness that would otherwise occur if your application were to build a complex image directly onscreen.

The `NewGWorld` function initializes the offscreen graphics port by calling the `OpenCPort` function. The `NewGWorld` function sets the offscreen graphics port's visible region to a rectangular region coincident with its boundary rectangle. The `NewGWorld` function generates an inverse table with the Color Manager function `MakeITable`, unless one of the `GDevice` structures for the screens has the same color table as the `GDevice` structure for the offscreen world, in which case `NewGWorld` uses the inverse table from that `GDevice` structure.

The address of the offscreen pixel image is not directly accessible from the `PixMap` structure for the offscreen graphics world. However, you can use the `GetPixBaseAddr` (page 247) function to get a pointer to the beginning of the offscreen pixel image.

For purposes of estimating memory use, you can compute the size of the offscreen pixel image by using this formula:

rowBytes * (boundsRect.bottom – boundsRect.top)

In the `flags` parameter, you can specify several options. If you don't want to use any of these options, pass 0 in the `flags` parameter:

- If you specify the `pixPurge` flag, `NewGWorld` stores the offscreen pixel image in a purgeable block of memory. In this case, before drawing to or from the offscreen pixel image, your application should call the `LockPixels` (page 314) function and ensure that it returns `TRUE`. If `LockPixels` returns `FALSE`, the memory for the pixel image has been purged, and your application should either call `UpdateGWorld` (page 454) to reallocate it and then reconstruct the pixel image, or draw directly in a window instead of preparing the image in an offscreen graphics world. Never draw to or copy from an offscreen pixel image that has been purged without reallocating its memory and then reconstructing it.
- If you specify the `noNewDevice` flag, `NewGWorld` does not create a new offscreen `GDevice` structure. Instead, it uses the `GDevice` structure that you specify in the `aGDevice` parameter–and its associated pixel depth and color table–to create the offscreen graphics world. (If you set the `pixelDepth` parameter to 0, `NewGWorld` uses the `GDevice` structure for the screen with the greatest pixel depth among all the screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter–even if you specify the `noNewDevice` flag.) The `NewGWorld` function keeps a reference to the `GDevice` structure for the offscreen graphics world, and the `SetGWorld` (page 410) function uses that structure to set the current graphics device.
- If you set the `useTempMem` flag, `NewGWorld` creates the base address for an offscreen pixel image in temporary memory. You generally would not use this flag, because you should use temporary memory only for fleeting purposes and only with the `AllowPurgePixels` (page 149) function.

- If you specify the `keepLocal` flag, your offscreen pixel image is kept in Macintosh main memory and is not cached to a graphics accelerator card. use this flag carefully, as it negates the advantages provided by any graphics acceleration card that might be present.

If your application needs to change the pixel depth, boundary rectangle, or color table for an offscreen graphics world, use the `UpdateGWorld` (page 454) function.

Special Considerations

If you supply a handle to a `ColorTable` structure in the `cTable` parameter, `NewGWorld` makes a copy of the structure and stores its handle in the offscreen `PixMap` structure. It is your application's responsibility to make sure that the `ColorTable` structure you specify in the `cTable` parameter is valid for the offscreen graphics port's pixel depth.

If when using `NewGWorld` you specify a pixel depth, color table, or `GDevice` structure that differs from those used by the window into which you copy your offscreen image, the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions require extra time to complete.

To use a custom color table in an offscreen graphics world, you need to create the associated offscreen `GDevice` structure, because Color QuickDraw needs its inverse table.

The `NewGWorld` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

DesktopSprites
DesktopSprites.win
Inside Mac Movie TB Code
Palette and GWorld
vrscript.win

Declared in

`QDOffscreen.h`

NewGWorldFromPtr

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
QDErr NewGWorldFromPtr (
   GWorldPtr *offscreenGWorld,
   UInt32 PixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags,
   Ptr newBuffer,
   SInt32 rowBytes
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon GLSnapshot

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

QTtoCG

Declared in
`QDOffscreen.h`

NewPixMap

Creates a new, initialized `PixMap` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PixMapHandle NewPixMap (
   void
);
```


Return Value
A handle to the new `PixMap` structure.

Discussion
All fields of the `PixMap` structure are copied from the current device's `PixMap` structure except the color table. In System 7, the `hRes` and `vRes` fields are set to 72 dpi, no matter what values the current device's `PixMap` structure contains. A handle to the color table is allocated but not initialized.

Typically, you do not need to call this function because `PixMap` structures are created for you when you create a window using the Window Manager functions `NewCWindow` and `GetNewCWindow` and when you create an offscreen graphics world with the `NewGWorld` function.

If your application creates a pixel map, your application must initialize the `PixMap` structure's color table to describe the pixels. Use the `GetCTable` (page 230) function to read such a table from a resource file. Use the `DisposeCTable` (page 186) function to dispose of the `PixMap` structure's color table and replace it with the one returned by `GetCTable`.

Special Considerations
The `NewPixMap` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
NewCCursor

Declared in
`QuickdrawAPI.h`

NewPixPat

Creates a new pixel pattern. Generally, however, your application should create a pixel pattern in a `'ppat'` resource. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
PixPatHandle NewPixPat (
```



```
   void
);
```

Return Value

A handle to the new `PixPat` (page 106) structure created by the `NewPixPat` function.

Discussion

This function calls the `NewPixMap` (page 333) function to allocate the pattern's `PixMap` structure and initializes it to the same settings as the pixel map of the current `GDevice` structure. `NewPixPat` also sets the `pat1Data` field of the new `PixPat` structure to a 50 percent gray pattern. `NewPixPat` allocates new handles for the `PixPat` structure's data, expanded data, expanded map, and color table but does not initialize them instead, your application must initialize them.

Set the `rowBytes`, `bounds`, and `pixelSize` fields of the pattern's `PixMap` structure to the dimensions of the desired pattern. The `rowBytes` value should be equal to

```
(width of bounds) x pixelSize/8
```

The `rowBytes` value need not be even. The width and height of the bounds must be a power of 2. Each scan line of the pattern must be at least 1 byte in length—that is, ([width of bounds] x `pixelSize`) must be at least 8.

Your application can explicitly specify the color corresponding to each pixel value with a color table. The color table for the pattern must be placed in the `pmTable` field in the pattern's `PixMap` structure.

Including the `PixPat` structure itself, `NewPixPat` allocates a total of five handles. The sizes of the handles to the `PixPat` and `PixMap` structures are the sizes of their respective data structures. The other three handles are initially small in size. Once the pattern is drawn, the size of the expanded data is proportional to the size of the pattern data, but adjusted to the depth of the screen. The color table size is the size of the structure plus 8 bytes times the number of colors in the table.

When you are finished using the pixel pattern, use the `DisposePixPat` (page 189) function to make the memory used by the pixel pattern available again.

Special Considerations

The `NewPixPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QuickdrawAPI.h

NewRgn

Begins creating a new region. (Available in OS X v10.0 through OS X v10.6.)

```
RgnHandle NewRgn (
   void
);
```

Return Value
A handle to the new region.

Discussion
The NewRgn function allocates space for a new, variable-size region and initializes it to the empty region defined by the rectangle (0,0,0,0). This is the only function that creates a new region; other functions merely alter the size or shape of existing regions.

To begin defining a region, use the OpenRgn (page 350) function.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the SetPort function to change to the graphics port containing the points or rectangles, using the LocalGlobal function to convert their locations to global coordinates, using SetPort to return to your starting graphics port, and then using the GlobalToLocal function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations
The NewRgn function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Use the Memory Manager function MaxMem to determine whether the memory for the region is valid before using NewRgn.

Ensure that the memory for a region is valid before calling this function to manipulate that region if there isn't sufficient memory, the system may crash. Regions are limited to 32 KB in size in basic QuickDraw and 64 KB in color QuickDraw. Before defining a region, you can use the Memory Manager function MaxMem to determine whether the memory for the region is valid. You can determine the current size of an existing region by calling

the Memory Manager function `GetHandleSize`. When you record drawing operations in an open region, the resulting region description may overflow the 32 KB or 64 KB limit. Should this happen in color QuickDraw, the `QDError` function returns the result code `regionTooBigError`.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Carbon Porting Tutorial
CustomWindow
HTMLSample
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

NewScreenBuffer

Creates an offscreen `PixMap` structure and allocates memory for the base address of its pixel image. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
QDErr NewScreenBuffer (
   const Rect *globalRect,
   Boolean purgeable,
   GDHandle *gdh,
   PixMapHandle *offscreenPixMap
);
```

Parameters

`globalRect`

The boundary rectangle, in global coordinates, for the offscreen pixel map.

`purgeable`

A value of `TRUE` to make the memory block for the offscreen pixel map purgeable, or a value of `FALSE` to make it unpurgeable.

`gdh`

On return, a pointer to the handle to the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle specified in the `globalRect` parameter.

`offscreenPixMap`

On return, a pointer to a handle to the new offscreen `PixMap` structure.

Return Value

A result code.

Discussion

Applications generally do not need to use `NewScreenBuffer`. The `NewGWorld` (page 328) function uses the `NewScreenBuffer` function to create and allocate memory for an offscreen pixel image.

Special Considerations

The `NewScreenBuffer` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QDOffscreen.h`

NewTempScreenBuffer

Creates an offscreen `PixMap` structure and allocate temporary memory for the base address of its pixel image applications generally don't need to use `NewTempScreenBuffer`. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDErr NewTempScreenBuffer (
   const Rect *globalRect,
   Boolean purgeable,
   GDHandle *gdh,
   PixMapHandle *offscreenPixMap
);
```

Parameters

`globalRect`

The boundary rectangle, in global coordinates, for the offscreen pixel map.

`purgeable`

A value of `TRUE` to make the memory block for the offscreen pixel map purgeable, or a value of `FALSE` to make it unpurgeable.

`gdh`

On return, a pointer to the handle to the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle specified in the `globalRect` parameter.

`offscreenPixMap`

On return, a pointer to the handle to the new offscreen `PixMap` structure.

Return Value
A result code.

Discussion
The `NewTempScreenBuffer` function performs the same functions as `NewScreenBuffer` except that it creates the base address for the offscreen pixel image in temporary memory. When an application passes it the `useTempMem` flag, the `NewGWorld` function uses `NewTempScreenBuffer` instead of `NewScreenBuffer`.

Your application should not need to use this function.

Special Considerations
The `NewTempScreenBuffer` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QDOffscreen.h`

NoPurgePixels

Prevents the Memory Manager from purging the base address for an offscreen pixel image. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void NoPurgePixels (
   PixMapHandle pm
);
```


Parameters

`pm`

A handle to an offscreen pixel map.

Discussion

The `NoPurgePixels` function marks the base address for an offscreen pixel image as unpurgeable.

To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 238) function. Then supply this handle for the `pm` parameter of `NoPurgePixels`.

Special Considerations

The `NoPurgePixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QDOffscreen.h`

ObscureCursor

Hides the cursor until the next time the user moves the mouse. (Available in OS X v10.0 through OS X v10.6.)

```
void ObscureCursor (
   void
);
```

Discussion

Your application normally calls `ObscureCursor` when the user begins to type. Unlike `HideCursor` (page 276) , `ObscureCursor` has no effect on the cursor level and must not be balanced by a call to `ShowCursor`.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

OffscreenVersion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
SInt32 OffscreenVersion (
   void
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QDOffscreen.h`

OffsetPoly

Moves a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void OffsetPoly (
   PolyHandle poly,
   short dh,
   short dv
);
```

Parameters

`poly`

A handle to a polygon to move.

`dh`

The horizontal distance to move the polygon.

`dv`

The vertical distance to move the polygon.

Discussion

The `OffsetPoly` function moves the polygon whose handle you pass in the `poly` parameter by adding the value you specify in the `dh` parameter to the horizontal coordinates of its points, and by adding the value you specify in the `dv` parameter to the vertical coordinates of all points of its region boundary. If the values of `dh`

and dv are positive, the movement is to the right and down; if either is negative, the corresponding movement is in the opposite direction. The region retains its size and shape. This does not affect the screen unless you subsequently call a function to draw the region.

OffsetPoly is an especially efficient operation, because the data defining a polygon is stored relative to the first point of the polygon and so is not actually changed by OffsetPoly.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QuickdrawAPI.h

OffsetRect

Moves a rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void OffsetRect (
   Rect * r,
   short dh,
   short dv
);
```

Parameters

r
: A pointer to the rectangle to move.

dh
: The horizontal distance to move the rectangle.

dv
: The vertical distance to move the rectangle.

Discussion
The OffsetRect function moves the rectangle that you specify in the r parameter by adding the value you specify in the dh parameter to each of its horizontal coordinates and the value you specify in the dv parameter to each of its vertical coordinates. If the dh and dv parameters are positive, the movement is to the right and

down; if either is negative, the corresponding movement is in the opposite direction. The rectangle retains its shape and size; it is merely moved on the coordinate plane. The movement does not affect the screen unless you subsequently call a function to draw within the rectangle.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in OS X v10.0 through OS X v10.6.

Related Sample Code
GlyphaIVOld
HTMLSample
MovieBrowser
QTMusicToo
TypeServicesForUnicode

Declared in
`QuickdrawAPI.h`

OffsetRgn

Moves a region. (Available in OS X v10.0 through OS X v10.6.)

```
void OffsetRgn (
   RgnHandle rgn,
   short dh,
   short dv
);
```

Parameters

`rgn`
A handle to the region to move.

`dh`
The horizontal distance to move the region.

`dv`
The vertical distance to move the region.

Discussion

The `OffsetRgn` function moves the region whose handle you pass in the `rgn` parameter by adding the value you specify in the `dh` parameter to the horizontal coordinates of all points of its region boundary, and by adding the value you specify in the `dv` parameter to the vertical coordinates of all points of its region boundary. If the values of `dh` and `dv` are positive, the movement is to the right and down; if either is negative, the corresponding movement is in the opposite direction. The region retains its size and shape. This does not affect the screen unless you subsequently call a function to draw the region.

The `OffsetRgn` function is an especially efficient operation, because most of the data defining a region is stored relative to the `rgnBBox` field in its `Region` structure and so is not actually changed by `OffsetRgn`.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

CopyBits vs. CopyMask
FinderDragPro
QTMusicToo
qtskins
TE Over Background

Declared in

`QuickdrawAPI.h`

OpColor

Sets the maximum color values for the `addPin` and `subPin` arithmetic transfer modes and the weight color for the `blend` arithmetic transfer mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void OpColor (
   const RGBColor *color
);
```

Parameters

`color`

An `RGBColor` structure that defines a color.

Discussion

Specify the red, green, and blue values in the `RGBColor` structure and specify this structure in the `color` parameter.

If the current graphics port is defined by a `GrafPort` structure, `OpColor` has no effect.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qtdataref.win

QTMusicToo

vrscript

vrscript.win

Declared in

`QuickdrawAPI.h`

OpenCPicture

Begins defining a picture in extended version 2 format. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PicHandle OpenCPicture (
   const OpenCPicParams *newHeader
);
```

Parameters

`newHeader`

An `OpenCPicParams` structure.

Return Value

A handle to a new `Picture` structure. `OpenCPicture` collects your subsequent drawing commands in this structure. Use this handle when referring to the picture in subsequent functions, such as the `DrawPicture` function.

Discussion

When defining a picture, you can use all other QuickDraw drawing functions, with the exception of `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask`. You can also use the `PicComment` (page 365) function to include picture comments in your picture definition.

The `OpenCPicture` function creates a pictures in the extended version 2 format. This format permits your application to specify resolutions when creating images.

Use the `OpenCPicParams` (page 99) structure you pass in the `newHeader` parameter to specify the horizontal and vertical resolution for the picture, and specify an optimal bounding rectangle for displaying the picture at this resolution. When you later call the `DrawPicture` (page 192) function to play back the saved picture, supply a destination rectangle, and `DrawPicture` scales the picture so that it is completely aligned with the destination rectangle. To display a picture at a resolution other than that at which it was created, compute an appropriate destination rectangle by scaling its width and height by the following factor:

```
scale factor = destination resolution / source resolution
```

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle.

The `OpenCPicture` function calls the `HidePen` function, so no drawing occurs on the screen while the picture is open (unless you call the `ShowPen` function just after `OpenCPicture`, or you called `ShowPen` previously without balancing it by a call to `HidePen`).

After defining the picture, close it by using the `ClosePicture` (page 161) function.

After creating the picture, use the `GetPictInfo` function to gather information about it. The `PictInfo` structure returned by `GetPictInfo` returns the picture's resolution and optimal bounding rectangle.

Special Considerations

When creating a picture, use the `ClosePicture` function to finish it before you open the Printing Manager with the `PrOpen` function. There are two main reasons for this. First, you should allow the printing driver to use as much memory as possible. Second, the Printing Manager creates its own type of graphics port, one that replaces the standard QuickDraw drawing operations stored in the `grafProcs` field of a `CGrafPort` or `GrafPort` structure. To avoid unexpected results when creating a picture, draw into a graphics port created with QuickDraw instead of drawing into a printing port created by the Printing Manager.

After calling `OpenCPicture`, be sure to finish your picture definition by calling `ClosePicture` before you call `OpenCPicture` again. You cannot nest calls to `OpenCPicture`.

Always use the `ClipRect` procedure to specify a clipping region appropriate for your picture before you call `OpenCPicture`. If you do not use `ClipRect` to specify a clipping region, `OpenCPicture` uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you scale the picture when drawing it, the clipping region can become invalid when `DrawPicture` scales the clipping region—in which case, your picture will not be drawn.

The `OpenCPicture` function may move or purge memory; do not call at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

OpenCursorComponent

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSErr OpenCursorComponent (
   Component c,
   ComponentInstance *ci
);
```

Return Value

Carbon Porting Notes
This function is not implemented on Mac OS X.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

OpenPicture

Creates a picture which allows you to specify resolutions for your pictures. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PicHandle OpenPicture (
   const Rect *picFrame
);
```

Parameters

`picFrame`

The bounding rectangle for the picture. The `DrawPicture` function uses this rectangle to scale the picture if you draw it into a destination rectangle of a different size.

Return Value

A handle to a new `Picture` structure. `OpenPicture` collects your subsequent drawing commands in this structure. Use this handle when referring to the picture in subsequent functions, such as the `DrawPicture` function.

Discussion

The `OpenPicture` function, which was created for earlier versions of system software, is described here for completeness. Use the `OpenPicture` function to begin defining a picture.

The `OpenPicture` function calls the `HidePen` function, so no drawing occurs on the screen while the picture is open (unless you call the `ShowPen` function just after `OpenPicture` or you called `ShowPen` previously without balancing it by a call to `HidePen`).

The `OpenPicture` function creates pictures in the version 2 format on computers with Color QuickDraw when the current graphics port is a color graphics port. Pictures created in this format support color drawing operations at 72 dpi. On computers supporting only basic QuickDraw, or when the current graphics port is a basic graphics port, this function creates pictures in version 1 format. Pictures created in version 1 format support only black-and-white drawing operations at 72 dpi.

When defining a picture, you can use all other QuickDraw drawing functions, with the exception of `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask`. You can also use the `PicComment` (page 365) function to include picture comments in your picture definition.

After defining the picture, close it by using the `ClosePicture` (page 161) function. To draw the picture, use the `DrawPicture` (page 192) function.

Special Considerations

The version 2 and version 1 picture formats support only 72-dpi resolution. The `OpenCPicture` function creates pictures in the extended version 2 format. The extended version 2 format, which is created by the `OpenCPicture` function on all Macintosh computers running System 7, permits your application to specify additional resolutions when creating images.

Version 1 pictures are limited to 32 KB. You can determine the picture size while it is being formed by calling the Memory Manager function `GetHandleSize`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

MovieBrowser

qdmediamaker.win

qteffects.win

QTMusicToo

qtspritesplus

Declared in

`QuickdrawAPI.h`

OpenPoly

Begins defining a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
PolyHandle OpenPoly (
   void
);
```

Return Value

A handle to a new polygon.

Discussion

The `OpenPoly` function starts saving lines for processing as a polygon definition. While a polygon is open, all calls to the `Line` and `LineTo` functions affect the outline of the polygon. Only the line endpoints affect the polygon definition; the pattern mode, pattern, and size do not affect it. The `OpenPoly` function calls the

HidePen function, so no drawing occurs on the screen while the polygon is open (unless you call the ShowPen function just after calling OpenPoly, or you called ShowPen previously without balancing it by a call to HidePen).

A polygon should consist of a sequence of connected lines. The OpenPoly function stores the definition for a polygon in a Polygon structure.

When a polygon is open, the current graphics port's polySave field contains a handle to information related to the polygon definition. If you want to temporarily disable the polygon definition, you can save the current value of this field, set the field to NULL, and later restore the saved value to resume the polygon definition.

Even though the onscreen presentation of a polygon is clipped, the definition of a polygon is not; you can define a polygon anywhere on the coordinate plane.

When you are finished calling the line-drawing functions that define your polygon, use the ClosePoly (page 162) function.

Special Considerations
Do not call OpenPoly while a region or another polygon is already open.

Polygons are limited to 64 KB. You can determine the polygon size while it is being formed by calling the Memory Manager function GetHandleSize.

The OpenPoly function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
MouseTracking

Declared in
QuickdrawAPI.h

OpenRgn

Begins defining a region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void OpenRgn (
   void
);
```

Discussion

The `OpenRgn` function allocates temporary memory to start saving lines and framed shapes for processing as a region definition. Call `OpenRgn` only after initializing a region with the `NewRgn` function.

The `NewRgn` function stores the definition for a region in a `Region` structure.

While a region is open, all calls to `Line`, `LineTo`, and the functions that draw framed shapes (except arcs) affect the outline of the region. Only the line endpoints and shape boundaries affect the region definition—the pattern mode, pattern, and size do not affect it.

When you are finished defining the region, call the `CloseRgn` (page 162) function.

The `OpenRgn` function calls `HidePen`, so no drawing occurs on the screen while the region is open (unless you call `ShowPen` just after `OpenRgn`, or you called `ShowPen` previously without balancing it by a call to `HidePen`). Since the pen hangs below and to the right of the pen location, drawing lines with even the smallest pen changes pixels that lie outside the region you define.

The outline of a region is mathematically defined and infinitely thin, and it separates the bit or pixel image into two groups of pixels: those within the region and those outside it.

A region should consist of one or more closed loops. Each framed shape itself constitutes a loop. Any lines drawn with the `Line` or `LineTo` function should connect with each other or with a framed shape. Even if the onscreen presentation of a region is clipped, the definition of a region is not; you can define a region anywhere on the coordinate plane with complete disregard for the location of various graphics port entities on that plane.

When a region is open, the current graphics port's `rgnSave` field contains a handle to information related to the region definition. If you want to temporarily disable the collection of lines and shapes, you can save the current value of this field, set the field to `NULL`, and later restore the saved value to resume the region definition. Also, calling `SetPort` while a region is being formed discontinues formation of the region until another call to `SetPort` resets the region's original graphics port.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

Regions are limited to 32 KB in size in basic QuickDraw and 64 KB in Color QuickDraw. You can determine the current size of an existing region by calling the Memory Manager function `GetHandleSize`. When you structure drawing operations in an open region, the resulting region description may overflow the 32 KB or 64 KB limit. Should this happen in Color QuickDraw, the `QDError` function returns the result code `regionTooBigError`.

Do not call `OpenRgn` while another region or a polygon is already open. When you are finished constructing the region, use the `CloseRgn` function, which is described next.

The `OpenRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CTMClip

kcapApp

QTMusicToo

Declared in

`QuickdrawAPI.h`

PackBits

Compresses a data buffer stored in RAM. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PackBits (
   Ptr *srcPtr,
   Ptr *dstPtr,
   short srcBytes
);
```

Parameters

`srcPtr`

On entry, a pointer to the first byte of a buffer of data to be compressed. On exit, a pointer to the first byte following the bytes compressed.

`dstPtr`

On entry, a pointer to the first byte in which to store compressed data. On exit, a pointer to the first byte following the compressed data.

`srcBytes`

The number of bytes of uncompressed data to be compressed. In versions of software prior to version 6.0.2, this number must be 127 or less.

Discussion

You must allocate memory for the destination buffer itself. Allocate enough memory for a worst-case scenario where the destination buffer is 128 bytes long for each block of source data up to 127 bytes. Use the following formula to determine how much space to allocate for the destination buffer:

maxDstBytes := srcBytes + (srcBytes+126) DIV 127;

where `maxDstBytes` stands for the maximum number of destination bytes.

The `PackBits` algorithm is most effective on data buffers in which there are likely to be series of bytes containing the same value. For example, resources of many formats often contain many consecutive zeros. If you have a data buffer in which there are only likely to be a series of words or long words containing the same values, `PackBits` is unlikely to be effective.

Special Considerations

Because your application must allocate memory for the source and destination buffers, `PackBits` does not move relocatable blocks. Thus, you can call it at interrupt time.

Because `PackBits` changes the values of the `srcPtr` and `dstPtr` parameters, you should pass to `PackBits` only copies of the pointers to the source and destination buffers. This allows you to access the beginning of the source and destination buffers after `PackBits` returns. Also, if the source or destination buffer is stored in an unlocked, relocatable block, this technique prevents `PackBits` from changing the value of a master pointer, which would make the original handle invalid.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

PaintArc

Paints a wedge of the oval that fits inside a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void PaintArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

`r`

The rectangle that defines an oval's boundaries.

`startAngle`

The angle indicating the start of the arc.

`arcAngle`

The angle indicating the arc's extent.

Discussion

Using the pen pattern and pattern mode of the current graphics port, the `PaintArc` function draws a wedge of the oval bounded by the rectangle that you specify in the `r` parameter. As in the `FrameArc` function, use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

The pen location does not change.

Use `FillArc` (page 205) , to draw a wedge with a pattern different from that specified in the `pnPat` field of the current graphics port.

Special Considerations

The `PaintArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Tiler

Declared in
`QuickdrawAPI.h`

PaintOval

Paints an oval with the graphics pen's pattern and pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PaintOval (
   const Rect *r
);
```

Parameters
`r`

The rectangle that defines the oval's boundary.

Discussion
Using the pen pattern and pattern mode for the current graphics port, the `PaintOval` function draws the interior of an oval just inside the bounding rectangle that you specify in the `r` parameter. The pen location does not change.

Use `FillOval` (page 212) to draw the interior of an oval with a pen pattern different from that for the current graphics port.

Special Considerations
The `PaintOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CocoaVideoFrameToGWorld
Out of This GWorld
qdmediamaker.win

Declared in
`QuickdrawAPI.h`

PaintPoly

Paints a polygon with the graphics pen's pattern and pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void PaintPoly (
   PolyHandle poly
);
```

Parameters

`poly`

A handle to the polygon to paint. The `OpenPoly` (page 349) function returns this handle when you first create the polygon.

Discussion

Using the pen pattern and pattern mode for the current graphics port, the `PaintPoly` function draws the interior of a polygon whose handle you pass in the `poly` parameter. The pen location does not change.

This function temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 319) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Use the `FillPoly` (page 213) function to draw the interior of a polygon with a pattern different from that specified in the `pnPat` field of the current graphics port.

Special Considerations

Do not create a height or width for the polygon greater than 32,767 pixels, or `PaintPoly` will crash.

The `PaintPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

PaintRect

Paints a rectangle with the graphics pen's pattern and pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PaintRect (
   const Rect *r
);
```

Parameters
`r`

The rectangle to paint.

Discussion
The `PaintRect` function draws the interior of the rectangle that you specify in the `r` parameter with the pen pattern for the current graphics port, according to the pattern mode for the current graphics port. The pen location does not change.

Use the `FillRect` (page 214) to draw the interior of a rectangle with a pen pattern different from that for the current graphics port.

Special Considerations
The `PaintRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
CreateMovie
DrawSprocketTestOld
HTMLSample
QTMusicToo

Declared in
`QuickdrawAPI.h`

PaintRgn

Paints a region with the graphics pen's pattern and pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PaintRgn (
   RgnHandle rgn
);
```

Parameters
`rgn`
A handle to the region to paint.

Discussion
Using the pen pattern and pattern mode for the current graphics port, the `PaintRgn` function draws the interior of the region whose handle you pass in the `rgn` parameter. The pen location does not change.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Use `FillRgn` (page 215) to draw the interior of a region with a pen pattern different from that for the current graphics port.

Special Considerations
The `PaintRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

PaintRoundRect

Paints a rounded rectangle with the graphics pen's pattern and pattern mode. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PaintRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

`r`

The rectangle that defines the rounded rectangle's boundaries.

`ovalWidth`

The width of the oval defining the rounded corner.

`ovalHeight`

The height of the oval defining the rounded corner.

Discussion

Using the pattern and pattern mode of the graphics pen for the current graphics port, the `PaintRoundRect` function draws the interior of the rounded rectangle bounded by the rectangle that you specify in the `r` parameter. Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners of the rounded rectangle.

The pen location does not change.

Use `FillRoundRect` (page 216) to draw the interior of a rounded rectangle with a pen pattern different from that for the current graphics port.

Special Considerations

The `PaintRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

PenMode

Sets the pattern mode of the graphics pen in the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void PenMode (
   short mode
);
```

Parameters

`mode`

The pattern mode. See "Source, Pattern, and Arithmetic Transfer Mode Constants" (page 138).

Discussion

Using the pattern mode you specify in the `mode` parameter, the `PenMode` function sets the manner in which the pattern of the graphics pen is transferred onto the bitmap (or pixel map) when you draw lines or shapes in the current graphics port.

If you specify a source mode (such as one used with the `CopyBits` function) instead of a pattern mode, no drawing is performed.

The current pattern mode is stored in the `pnMode` field of the current graphics port. The initial pattern mode value is `patCopy`, in which the pen pattern is copied directly to the bitmap.

To use highlighting, add the `hilite` constant or its value to the source or pattern mode:

With highlighting, QuickDraw replaces the background color with the highlight color when your application draws or copies images between graphics ports. This has the visual effect of using a highlighting pen to select the object. (The global variable `HiliteRGB` is read from parameter RAM when the machine starts. Basic graphics ports use the color stored in the `HiliteRGB` global variable as the highlight color. Color graphics ports default to the `HiliteRGB` global variable, but can be overridden by the `HiliteColor` function.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Special Considerations

When your application draws with a pixel pattern, Color QuickDraw ignores the pattern mode and simply transfers the pattern directly to the pixel map without regard to the foreground and background colors.

The results of inverting a pixel are predictable only with direct pixels or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the color table. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
FinderDragPro
HTMLSample
Live Scroll
MovieBrowser
QTMusicToo

Declared in
`QuickdrawAPI.h`

PenNormal

Sets the size, pattern, and pattern mode of the graphics pen in the current graphics port to their initial values. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void PenNormal (
   void
);
```

Discussion
The `PenNormal` function restores the size, pattern, and pattern mode of the graphics pen in the current graphics port to their initial values: a size of 1 pixel by 1 pixel, a pattern mode of `patCopy`, and a pattern of `black`. The pen location does not change.

Special Considerations
The `PenNormal` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Fragment Tool
GlyphaIVOld
Icon Play
MovieBrowser
QTMusicToo

Declared in
`QuickdrawAPI.h`

PenPat

Sets the bit pattern to be used by the graphics pen in the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PenPat (
   const Pattern *pat
);
```

Parameters
`pat`

A bit pattern, as defined by a `Pattern` structure.

Discussion
The `PenPat` function sets the graphics pen to use the bit pattern defined in the `Pattern` (page 100) structure that you specify in the `pat` parameter. (The standard patterns `white`, `black`, `gray`, `ltGray`, and `dkGray` are predefined; the initial bit pattern for the pen is `black`.) This pattern is stored in the `pnPat` field of a `GrafPort` structure. The QuickDraw painting functions (such as `PaintRect`) also use the pen's pattern when drawing a shape.

The `PenPat` function also sets a bit pattern for the graphics pen in a color graphics port. The `PenPat` function creates a handle, of type `PixPatHandle`, for the bit pattern and stores this handle in the `pnPixPat` field of the `CGrafPort` structure. This pattern always uses the graphics port's current foreground and background colors.

To define your own patterns, you typically create pattern, 'PAT', or pattern list, 'PAT#', resources.

Special Considerations

The `PenPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CocoaVideoFrameToGWorld
DialogsToHIViews
HTMLSample
MovieBrowser
QTMusicToo

Declared in

`QuickdrawAPI.h`

PenPixPat

Sets the pixel pattern used by the graphics pen in the current color graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PenPixPat (
   PixPatHandle pp
);
```

Parameters

`pp`

A handle to the pixel pattern to use as the pen pattern.

Discussion

The `PenPixPat` function is similar to the basic QuickDraw function `PenPat`, except that you pass `PenPixPat` a handle to a multicolored pixel pattern rather than a bit pattern.

The `PenPixPat` function stores the handle to the pixel pattern in the `pnPixPat` field of the `CGrafPort` structure, therefore, you should not dispose of this handle since QuickDraw removes all references to your pattern from an existing graphics port when you dispose of it.

If you use `PenPixPat` to set a pixel pattern in a basic graphics port, the data in the `pat1Data` field of the `PixPat` (page 106) structure is placed into the `pnPat` field of the `GrafPort` structure.

To define your own pixel pattern, you can create a pixel pattern resource, which is described in `'ppat'`, or you can use the `NewPixPat` (page 334) function. To set the pen to use a bit pattern, you can also use the QuickDraw function `PenPat`.

Special Considerations
The `PenPixPat` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

PenSize

Sets the dimensions of the graphics pen in the current graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PenSize (
   short width,
   short height
);
```

Parameters
`width`
The pen width, as an integer from 0 to 32,767. If you set the value to 0, the pen does not draw. Values less than 0 are undefined.

`height`
The pen height, as an integer from 0 to 32,767. If you set the value to 0, the pen does not draw. Values less than 0 are undefined.

Discussion

The `PenSize` function sets the width that you specify in the `width` parameter and the height that you specify in the `height` parameter for the graphics pen in the current graphics port. All subsequent calls to the `Line` and `LineTo` functions and to the functions that draw framed shapes in the current graphics port use the new pen dimensions.

You can get the current pen dimensions from the `pnSize` field of the current graphics port, where the width and height are stored as a `Point` structure.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

MovieBrowser
Out of This GWorld
Palette and GWorld
QTMusicToo
qtspritesplus

Declared in

`QuickdrawAPI.h`

PicComment

Inserts a picture comment into a picture that you are defining or into your printing code. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PicComment (
   short kind,
   short dataSize,
   Handle dataHandle
);
```


Parameters

`kind`

The type of comment.

`dataSize`

Size of any additional data passed in the `dataHandle` parameter. If no additional data is used, specify 0 in this parameter.

`dataHandle`

A handle to additional data, if used. If no additional data is used, specify `NULL` in this parameter.

Discussion

When used after your application begins creating a picture with the `OpenCPicture` (or `OpenPicture`) function, the `PicComment` function inserts the specified comment into the `Picture` structure. When sent to a printer driver after your application uses the `PrOpenPage` function, `PicComment` passes the data or commands in the specified comment directly to the printer.

Picture comments contain data or commands for special processing by output devices, such as printers.

Usually printer drivers process picture comments, but applications can also do so. For your application to process picture comments, it must replace the `StdComment` function pointed to by the `commentProc` field of the `CQDProcs` or `QDProcs` structure, which in turn is pointed to by the `grafProcs` field of a `CGrafPort` or `GrafPort` structure. The default `StdComment` function provided by QuickDraw does no comment processing whatsoever. You can use the `SetStdCProcs` function to assist you in changing the `CQDProcs` structure, and you can use the `SetStdProcs` function to assist you in changing the `QDProcs` structure.

If you create and process your own picture comments, you should define comments so that they contain information that identifies your application (to avoid using the same comments as those used by Apple or by other third-party products). You should define a comment as an `ApplicationComment` comment type with a `kind` value of 100. The first 4 bytes of the data for the comment should specify your application's signature. You can use the next 2 bytes to identify the type of comment—that is, to specify a `kind` value to your own application.

Suppose your application signature were `'WAVE'`, and you wanted to use the value 128 to identify a `kind` value to your own application. You would supply values to the `kind` and `data` parameters to `PicComment` as follows:

`kind` = 100; `data` = `'WAVE'` [4 bytes] + 128 [2 bytes] + additional data [n bytes]

Your application can then parse the first 6 bytes of the comment to determine whether and how to process the rest of the data in the comment. It is up to you to publish information about your comments if you wish them to be understood and used by other applications.

Special Considerations

These former picture comments are now obsolete: `SetGrayLevel`, `ResourcePS`, `PostScriptFile`, and `TextIsPostScript`.

The `PicComment` function may move or purge memory.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

PixMap32Bit

Determines whether a pixel map requires 32-bit addressing mode for access to its pixel image. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean PixMap32Bit (
   PixMapHandle pmHandle
);
```

Parameters

`pmHandle`

A handle to an offscreen pixel map.

Return Value

`TRUE` if a pixel map requires 32-bit addressing mode for access to its pixel image. If your application is in 24-bit mode, you must change to 32-bit mode.

Discussion

To get a handle to an offscreen pixel map, first use the `GetGWorldPixMap` (page 238) function. Then supply this handle for the `pm` parameter of `PixMap32Bit`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Live Scroll

Declared in
`QDOffscreen.h`

PixPatChanged

Notifies QuickDraw that the content of a `PixPat` structure, including its `PixMap` structure or the image in its `patData` field, has been modified. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void PixPatChanged (
   PixPatHandle ppat
);
```

Parameters

`ppat`

A handle to the changed pixel pattern.

Discussion

The `PixPatChanged` function sets the `patXValid` field of the `PixPat` structure specified in the `ppat` parameter to –1 and notifies QuickDraw of the change.

If your application changes the `pmTable` field of a pixel pattern's `PixMap` structure, it should call `PixPatChanged`. However, if your application changes the content of the color table referenced by the `PixMap` structure's `pmTable` field, it should call both the `PixPatChanged` and the `CTabChanged` functions.

Your application should never need to directly modify a `PixPat` structure and use the `PixPatChanged` function; instead, your application should use the QuickDraw functions for manipulating the values in a `PixPat` structure.

Special Considerations

The `PixPatChanged` function may move or purge memory in the application heap; do not call the `PixPatChanged` function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QDOffscreen.h

PortChanged

Notifies QuickDraw that the content of a GrafPort structure or CGrafPort structure, including any of the data structures specified by handles within the structure, has been modified. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void PortChanged (
   GrafPtr port
);
```

Parameters
port
A pointer to the GrafPort structure that you have changed.

Discussion
If your application has changed a CGrafPort structure, it must coerce the CGrafPtr so it will point to a GrafPtr before passing the pointer in the port parameter.

You generally should not directly change any of the PixPat structures specified in a CGrafPort structure, but instead use the PenPixPat and BackPixPat functions. However, if your application does change the content of a PixPat structure, it should call the PixPatChanged function and the PortChanged function.

If your application changes the pmTable field of the PixMap structure specified in the graphics port, your application should call PortChanged. If your application changes the content of the ColorTable structure referenced by the pmTable field, it should call CTabChanged also.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QDOffscreen.h

PortSize

Changes the size of the port rectangle of the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void PortSize (
   short width,
   short height
);
```

Parameters

`width`

The width of the reset port rectangle.

`height`

The height of the reset port rectangle.

Discussion

The `PortSize` function is normally called only by the Window Manager. The `PortSize` function changes the size of the current graphics port's port rectangle. The upper-left corner of the port rectangle remains at its same location the width and height of the port rectangle are set to the given `width` and `height`. In other words, `PortSize` moves the lower-right corner of the port rectangle to a position relative to the upper-left corner.

The `PortSize` function doesn't change the clipping or visible region of the graphics port, nor does it affect the local coordinate system of the graphics port it changes only the width and height of the port rectangle. Remember that all drawing occurs only in the intersection of the boundary rectangle and the port rectangle, after being cropped to the visible region and the clipping region.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

AlwaysPreview

StarMenu

Declared in

`QuickdrawAPI.h`

ProtectEntry

Adds protection to or removes protection from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void ProtectEntry (
   short index,
   Boolean protect
);
```

Parameters

`index`

The index to the entry whose protection is to be changed.

`protect`

A Boolean value: specify `true` to protect the entry, `false` to remove protection.

Discussion

A protected entry can not be changed by other applications. `ProtectEntry` returns a protection error in `QDErr` if you attempt to protect an already protected entry. However, it can remove protection from any entry, even an already unprotected one.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

Pt2Rect

Determines the smallest rectangle that encloses two given points. (Available in OS X v10.0 through OS X v10.6.)

```
void Pt2Rect (
   Point pt1,
   Point pt2,
   Rect *dstRect
);
```


Parameters

`pt1`

The first of two points to enclose.

`pt2`

The second of two points to enclose.

`dstRect`

On return, a pointer to the smallest rectangle that can enclose them.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawAPI.h`

PtInRect

Determines whether a pixel below is enclosed in a rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean PtInRect (
   Point pt,
   const Rect * r
);
```

Parameters

`pt`

The point to test.

`r`

The rectangle to test.

Return Value

`TRUE` if the pixel below and to the right of the point specified in the `pt` parameter is enclosed in the rectangle specified in the `r` parameter. `FALSE` if it is not.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

Carbon Porting Tutorial

CarbonMDEF

Fragment Tool

MovieBrowser

QTMusicToo

Declared in

`QuickdrawAPI.h`

PtInRgn

Determines whether a pixel is within a region. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean PtInRgn (
   Point pt,
   RgnHandle rgn
);
```

Parameters

`pt`

The point whose pixel is to be checked.

`rgn`

A handle to the region to test.

Return Value

`TRUE` if the pixel below and to the right of the point specified in the `pt` parameter is within the region whose handle is specified in the `rgn` parameter. `FALSE` if it is not.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

CustomWindow
qteffects.win
qtskins
qtspritesplus
vrmovies.win

Declared in

`QuickdrawAPI.h`

PtToAngle

Calculates an angle between a vertical line pointing straight up from the center of a rectangle and a line from the center to a given point. (Available in OS X v10.0 through OS X v10.6.)

```
void PtToAngle (
   const Rect *r,
   Point pt,
   short *angle
);
```

Parameters

`r`

The rectangle to examine.

`pt`

The point to which an angle is to be calculated.

`angle`

On return, a pointer to the resulting angle.

The result returned in the `angle` parameter is specified in degrees from 0 to 359, measured clockwise from 12 o'clock, with 90 at 3 o'clock, 180 at 6 o'clock, and 270 at 9 o'clock. Other angles are measured relative to the rectangle. If the line to the given point goes through the upper-right corner of the rectangle, the angle returned is 45, even if the rectangle is not square if it goes through the lower-right corner, the angle is 135, and so on.

The angle returned can be used as input to one of the functions that manipulate arcs and wedges, in "Drawing Arcs and Wedges".

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawAPI.h`

QDDisplayWaitCursor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void QDDisplayWaitCursor (
   Boolean forceWaitCursor
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDDone

Determines whether QuickDraw has completed drawing in a given graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean QDDone (
   GrafPtr port
);
```

Parameters
`port`

The `GrafPort` structure for a graphics port in which your application has begun drawing; if you pass `NULL`, `QDDone` tests all open graphics ports.

Return Value
`TRUE` if all drawing operations have finished in the graphics port specified in the `port` parameter, `FALSE` if any remain to be executed. If you pass `NULL` in the `port` parameter, then `QDDone` returns `TRUE` only if drawing operations have completed in all ports.

Discussion
The `QDDone` function may be useful if a graphics accelerator is present and operating asynchronously. You can use it to ensure that all drawing is done before issuing new drawing commands, and to avoid the possibility that the new drawing operations might be overlaid by previously issued but unexecuted operations.

Special Considerations
If a graphics port draws a clock or some other continuously operating drawing process, `QDDone` may never return `TRUE`.

To determine whether all drawing in a color graphics port has completed, you must coerce its `CGrafPort` structure to a `GrafPort` structure, which you pass in the `port` parameter.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QDOffscreen.h`

QDError

Obtains a result code from the last applicable QuickDraw function that you called. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
short QDError (
   void
);
```

Return Value
The error result. On a system with only basic QuickDraw, `QDError` always returns `noErr`.

Discussion
The `QDError` function is helpful in determining whether insufficient memory caused a drawing operation - particularly those involving regions, polygons, pictures, and images copied with `CopyBits` - to fail.

Basic QuickDraw uses stack space for work buffers. For complex operations such as depth conversion, dithering, and image resizing, stack space may not be sufficient. QuickDraw attempts to get temporary memory from other parts of the system. If that is still not enough, `QDError` returns the `nsStackErr` error. If your application receives this result, reduce the memory required by the operation.

When you structure drawing operations in an open region, the resulting region description may overflow the 64 KB limit. In this case, `QDError` returns `regionTooBigError`. Since the resulting region is potentially corrupt, the `CloseRgn` function returns an empty region if it detects `QDError` has returned `regionTooBigError`. A similar error, `rgnTooBigErr`, occurs when using the `BitMapToRegion` function to convert a bitmap to a region.

The `BitMapToRegion` function also generates the `pixmapTooDeepErr` error if a `PixMap` structure is supplied that is greater than 1 bit per pixel. You may be able to recover from this problem by coercing your `PixMap` structure into a 1-bit `PixMap` structure and calling the `BitMapToRegion` function again.

Special Considerations
The `QDError` function does not report errors returned by basic QuickDraw.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDFlushPortBuffer

Calls the Quartz compositor to flush all new drawing in a Carbon window to the display. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void QDFlushPortBuffer (
   CGrafPtr port,
   RgnHandle region
);
```

Parameters

`port`

A window port. If the port has no back buffer, or if the port is an offscreen or printing port, this function does nothing.

`region`

An update region. Under normal conditions, you should pass `NULL` to avoid the overhead of additional region operations.

Discussion

In Mac OS X, drawing in a window port updates a back buffer associated with the window. Updates to this buffer are accumulated in a list called the dirty region.

The back buffer is automatically flushed to the display each time through the event loop. When the event loop does not get control soon enough—for example, during an animation sequence—you can call this function to flush the port buffer to the device immediately.

When you call this function, there are several different execution paths:

1. If the `region` parameter is `NULL`, the dirty region is flushed—along with any Quartz 2D drawing operations marked for update by calls to `CGContextSynchronize`—and the dirty region is set to empty.
2. If the `region` parameter specifies an update region, the intersection of the dirty region and the update region is flushed—along with any Quartz 2D drawing operations marked for update by calls to `CGContextSynchronize`—and the flushed region is subtracted from the dirty region.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ATSUICurveAccessDemo
Carbon Porting Tutorial
QTCarbonShell
Tiler
VRMakePano Library

Declared in
`QuickdrawAPI.h`

QDGetDirtyRegion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDGetDirtyRegion (
   CGrafPtr port,
   RgnHandle rgn
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDIsPortBufferDirty

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean QDIsPortBufferDirty (
   CGrafPtr port
);
```


Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
vrscript

Declared in
`QuickdrawAPI.h`

QDIsPortBuffered

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean QDIsPortBuffered (
   CGrafPtr port
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ElectricImageComponent.win

VRMakePano Library

vrscript

Declared in
`QuickdrawAPI.h`

QDRegionToRects

(Available in OS X v10.0 through OS X v10.6.)

```
OSStatus QDRegionToRects (
   RgnHandle rgn,
```



```
   QDRegionParseDirection dir,
   RegionToRectsUPP proc,
   void *userData
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Declared in
`QuickdrawAPI.h`

QDSetDirtyRegion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDSetDirtyRegion (
   CGrafPtr port,
   RgnHandle rgn
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

Random

Obtains a pseudorandom integer. (Available in OS X v10.0 through OS X v10.6. Use the Standard C Library `random(3)` function instead.)

```
short Random (
   void
);
```


Return Value
A pseudorandom integer, uniformly distributed in the range -32767 to 32767.

Discussion
The value `Random` returns depends solely on the global variable `randSeed`, which the QuickDraw `InitGraf` function initializes to 1. Each time the `Random` function executes, it uses a numerical algorithm to change the value of `randSeed` to prevent it from returning the same value each time it is called.

To prevent your application from generating the same sequence of pseudo-random numbers each time it is executed, initialize the `randSeed` global variable, when your application starts up, to a volatile long word variable such as the current date and time. If you would like to generate the same sequence of pseudo-random numbers twice, on the other hand, simply set `randSeed` to the same value before calling `Random` for each sequence.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DrawSprocketTestOld
FullScreen
GlyphaIVOld
ImageCompositing
SkyCreator

Declared in
`QuickdrawAPI.h`

RealColor

Determines whether a given `RGBColor` data structure exists in the current device's color table. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean RealColor (
   const RGBColor *color
);
```


Parameters

`color`

The `RGBColor` data structure to be tested.

Discussion

The `RealColor` function determines whether the color is available in the current `GDevice` data structure's CLUT, basing its search on the current resolution of the inverse table. For example, if the current value of the `iTabRes` field is 4, `RealColor` returns `true` if there exists a color that exactly matches the top 4 bits of red, green, and blue. (See the `iTabRes` field of the inverse table, `ITab` (page 96).)

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

RectInRgn

Determines whether a rectangle intersects a region. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean RectInRgn (
   const Rect *r,
   RgnHandle rgn
);
```

Parameters

`r`

The rectangle to check for intersection.

`rgn`

A handle to the region to check.

Return Value

`TRUE` if the rectangle specified in the `r` parameter intersects the region whose handle is specified in the `rgn` parameter. The `RectInRgn` function returns `TRUE` if the intersection encloses at least 1 bit or `FALSE` if it does not.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

The `RectInRgn` function sometimes returns `TRUE` when the rectangle merely intersects the region's bounding rectangle. If you need to know exactly whether a given rectangle intersects the actual region, use `RectRgn` (page 384) to set the rectangle to a region, and call `SectRgn` (page 396) to see whether the two regions intersect. If the result of `SectRgn` is an empty region, then the rectangle does not intersect the region.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

RectRgn

Changes the structure of an existing region to that of a rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void RectRgn (
   RgnHandle rgn,
   const Rect *r
);
```

Parameters

`rgn`

A handle to the region to restructure as a rectangle.

`r`

The rectangle structure to use.

Discussion

The `RectRgn` function destroys the previous structure of the `SetRectRgn` function, and it then sets the new structure to a rectangle that you specify in the `r` parameter.

As an alternative to the `RectRgn` function, use the `SetRectRgn` function, which accepts as parameters four coordinates instead of a rectangle.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations
The `RectRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
Fragment Tool
HackTV Carbon
hacktv.win
MovieBrowser
TE Over Background

Declared in
`QuickdrawAPI.h`

ReserveEntry

Reserves or removes reservation from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void ReserveEntry (
   short index,
   Boolean reserve
);
```


Parameters

`index`

The index to the entry.

`reserve`

`True` to reserve the entry, `false` to remove the reservation.

Discussion

A reserved entry cannot be matched by another application's search function, and `Color2Index` (or other functions that depend on it such as `RGBForeColor`, `RGBBackColor`, and `SetCPixel`) never return that entry to another client. You could use this function to selectively protect a color for color table animation.

The `ReserveEntry` function copies the low byte of the `gdID` field of the current `GDevice` data structure into the low byte of the `ColorSpec.value` field of the color table when reserving an entry, and leaves the high byte alone. `ReserveEntry` acts like selective protection and does not allow any changes if the current `gdID` field is different than the one in the `ColorSpec.value` field of the reserved entry. If a requested match is already reserved, `ReserveEntry` returns a protection error. It can remove reservation from any entry, even if a requested match is already not reserved.

Carbon Porting Notes

This function does nothing useful on Mac OS X.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

RestoreEntries

Restores a selection of color table entries. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void RestoreEntries (
   CTabHandle srcTable,
   CTabHandle dstTable,
```



```
   ReqListRec *selection
);
```

Parameters

`srcTable`

The color table containing entries to be restored.

`dstTable`

The color table in which to restore the entries. If `dstTable` is `NULL`, or points to the current `GDevice` data structure's color table, `RestoreEntries` changes the device's color table and the hardware CLUT to these new colors.

`selection`

A pointer to the `ReqListRec` (page 119) data structure. The entries to be restored are enumerated as offsets into a `ColorTable` data structure, not the contents of the `ColorSpec.value` field.

Discussion

The `RestoreEntries` function does not rebuild the inverse table.

If a request is beyond the end of the destination color table, `RestoreEntries` sets that position in the `requestList` data structure to `colReqErr`, and returns an error. `RestoreEntries` assumes that the color table specified by the `srcTable` parameter and the request list specified by the `selection` parameter have the same number of entries.

`RestoreEntries` does not change the color table's seed, so no invalidation occurs (which may cause `RGBForeColor` to act strangely). `RestoreEntries` ignores protection and reservation of color table entries.

You generally should use the Palette Manager to give your application its own set of colors; use of `RestoreEntries` should be limited to special-purpose applications. `RestoreEntries` allows you to change a color table without changing its `ctSeed` field. You can execute the application code and then use `RestoreEntries` to put the original colors back in. However, in some cases things in the background may appear in the wrong colors, since they were never redrawn. To avoid this, your application must build its own new inverse table and redraw the background. If you then use `RestoreEntries`, you should call the `CTabChanged` function to clean up correctly.

Carbon Porting Notes

This function does nothing useful on Mac OS X.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

RGBBackColor

Changes the background color. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void RGBBackColor (
   const RGBColor *color
);
```

Parameters
`color`
An `RGBColor` structure.

Discussion
If the current port is defined by a `CGrafPort` structure, QuickDraw supplies its `rgbBkColor` field with the RGB value that you specify in the `color` parameter, and places the pixel value most closely matching that color in the `bkColor` field. For indexed devices, the pixel value is an index to the current device's CLUT. F or direct devices, the value is the 16-bit or 32-bit equivalent to the RGB value.

If the current port is defined by a `GrafPort` structure, basic QuickDraw supplies its `fgColor` field with a color value determined by taking the high bit of each of the red, green, and blue components of the color that you supply in the `color` parameter. Basic QuickDraw uses that 3-bit number to select a color from its eight-color system.

You can also use Palette Manager functions to set the background color.

To determine the current background color, use the `GetBackColor` (page 226) function.

Because a pixel pattern already contains color, QuickDraw ignores the background color and foreground colors when your application draws with a pixel pattern. Use the `PenPixPat` function to assign a pixel pattern to the foreground pattern used by the graphics pen. Use the `BackPixPat` function to assign a pixel pattern as the background pattern for the current color graphics port. Use the `FillCRect`, `FillCOval`, `FillCRoundRect`, `FillCArc`, `FillCRgn`, and `FillCPoly` functions to fill shapes with a pixel pattern.

Special Considerations
The `RGBBackColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites
qteffects.win
qtspritesplus
qtwiredactions
qtwiredspritesjr.win

Declared in
`QuickdrawAPI.h`

RGBForeColor

Changes the color of the "ink" used for framing and painting. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void RGBForeColor (
   const RGBColor *color
);
```

Parameters
`color`

An `RGBColor` structure.

Discussion
If the current port is defined by a `CGrafPort` structure, QuickDraw supplies its `rgbFgColor` field with the RGB value that you specify in the `color` parameter, and places the pixel value most closely matching that color in the `fgColor` field. For indexed devices, the pixel value is an index to the current device's CLUT. For direct devices, the value is the 16-bit or 32-bit equivalent to the RGB value.

If the current port is defined by a `GrafPort` structure, basic QuickDraw supplies its `fgColor` field with a color value determined by taking the high bit of each of the red, green, and blue components of the color that you supply in the `color` parameter. Basic QuickDraw uses that 3-bit number to select a color from its eight-color system.

You can also use Palette Manager functions to set the foreground color.

To determine the current foreground color, use the `GetForeColor` (page 234) function.

QuickDraw ignores the foreground and background colors when your application draws with a pixel pattern. Assign a pixel pattern to the foreground pattern used by the graphics pen; by using the `BackPixPat` function to assign a pixel pattern as the background pattern for the current color graphics port; and by using the `FillCRect`, `FillCOval`, `FillCRoundRect`, `FillCArc`, `FillCRgn`, and `FillCPoly` functions to fill shapes with a pixel pattern.

Special Considerations
The `RGBForeColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ChromaKeyMovie
GlyphaIVOld
qtmovietrack.win
QTMusicToo
TE Over Background

Declared in
`QuickdrawAPI.h`

SaveEntries

Saves a selection of color table entries. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SaveEntries (
   CTabHandle srcTable,
   CTabHandle resultTable,
   ReqListRec *selection
);
```


Parameters

`srcTable`

The color table containing entries to be saved. If you supply `NULL`, `SaveEntries` uses the current device's color table as the source.

`resultTable`

The color table in which to save the entries.

`selection`

A pointer to the `ReqListRec` (page 119) data structure. The entries to be set are enumerated as offsets into a `ColorTable` data structure, not the contents of the `ColorSpec.value` field.) If an entry is not present in `srcTable`, then `SaveEntries` sets that position of the `selection` parameter to `colReqErr`, and that position of `resultTable` contains random values.

Discussion

If `SaveEntries` can not find one or more entries, then it posts an error code to `QDError`; however, for every entry in selection which is not `colReqErr`, the values in `resultTable` are valid. `SaveEntries` assumes that the color table specified by the `srcTable` parameter and the request list specified by the `selection` parameter have the same number of entries.

The output of `SaveEntries` is the same as the input for `RestoreEntries`, except for the order.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

ScalePt

Scales a height and width according to the proportions of two rectangles. (Available in OS X v10.0 through OS X v10.6.)

```
void ScalePt (
   Point *pt,
   const Rect *srcRect,
   const Rect *dstRect
);
```


Parameters

`pt`

On input, a pointer to an initial height and width (specified in the two fields of a `Point` structure) to scale; on return, vertical and horizontal scaling factors derived by multiplying the height and width by ratios of the height and width of the rectangle in the `srcRect` parameter to the height and width of the rectangle in the `dstRect` parameter.

You do not pass coordinates in this parameter. Instead, you pass an initial height to scale in the `v` (or vertical) field of the `Point` structure, and you pass an initial width to scale in the `h` (or horizontal) field.

The `ScalePt` function scales these measurements by multiplying the initial height by the ratio of the height of the rectangle you specify in the `dstRect` parameter to the height of the rectangle you specify in the `srcRect` parameter, and by multiplying the initial width by the ratio of the width of the `dstRect` rectangle to the width of the `srcRect` rectangle.

`srcRect`

A rectangle. The ratio of this rectangle's height to the height of the rectangle in the `dstRect` parameter provides the vertical scaling factor, and the ratio of this rectangle's width to the width of the rectangle in the `dstRect` parameter provides the horizontal scaling factor.

`dstRect`

A rectangle compared to the rectangle in the `srcRect` parameter to determine vertical and horizontal scaling factors.

Discussion

The `ScalePt` function produces horizontal and vertical scaling factors from the proportions of two rectangles. Use `ScalePt`, for example, to scale the dimensions of the graphics pen.

Where the width of the `dstRect` rectangle is twice the width of the `srcRect` rectangle, and its height is three times the height of `srcRect`, `ScalePt` scales the width of the graphics pen from 3 to 6 and scales its height from 2 to 6.

Special Considerations

The minimum value `ScalePt` returns is (1,1).

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in
QuickdrawAPI.h

ScreenRes

Determines the resolution of the main device. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void ScreenRes (
   short *scrnHRes,
   short *scrnVRes
);
```

Parameters

`scrnHRes`

On return, the number of horizontal pixels per inch displayed by the current device.

`scrnVRes`

On return, the number of vertical pixels per inch displayed by the current device.

Discussion

To determine the resolutions of all available graphics devices, examine their `GDevice` (page 91) structures. The horizontal and vertical resolutions for a graphics device are stored in the `hRes` and `vRes` fields, respectively, of the `PixMap` structure for the device's `GDevice` structure.

Currently, QuickDraw and the Printing Manager always assume a screen resolution of 72 dpi.

Do not use the actual screen resolution as a scaling factor when drawing into a printing graphics port. Instead, always use 72 dpi as the scaling factor. See the Printing Manager documentation for more information about drawing into a printing graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

QuickdrawAPI.h

ScrollRect

Scroll the pixels of a specified portion of a basic graphics port's bitmap (or a color graphics port's pixel map). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void ScrollRect (
   const Rect *r,
   short dh,
   short dv,
   RgnHandle updateRgn
);
```

Parameters

`r`

The pointer to the rectangle defining the area to be scrolled.

`dh`

The horizontal distance to be scrolled.

`dv`

The vertical distance to be scrolled.

`updateRgn`

A handle to the region of the window that needs to be updated.

Discussion

The `ScrollRect` function shifts pixels that are inside the specified rectangle of the current graphics port. No other pixels or the bits they represent are affected. The pixels are shifted a distance of `dh` horizontally and `dv` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified rectangle are not displayed, and the bits they represent are not saved. It is up to your application to save this data.

The empty area created by the scrolling is filled with the graphics port's background pattern, and the update region is changed to this filled area.

The `ScrollRect` function doesn't change the local coordinate system of the graphics port it simply moves the rectangle specified in the `r` parameter to different coordinates. Notice that `ScrollRect` doesn't move the graphics pen or the clipping region. However, because the document has moved, they're in different positions relative to the document.

By creating an update region for the window, `ScrollRect` forces an update event. After using `ScrollRect`, your application should use its own window-updating code to draw into the update region of the window.

The `ScrollRect` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CarbonMDEF

MovieBrowser

QTMusicToo

vrscript

vrscript.win

Declared in
`QuickdrawAPI.h`

SectRect

Determines whether two rectangles intersect. (Available in OS X v10.0 through OS X v10.6.)

```
Boolean SectRect (
   const Rect *src1,
   const Rect *src2,
   Rect *dstRect
);
```

Parameters
`src1`

The first of two rectangles to test for intersection.

`src2`

The second of two rectangles to test for intersection.

`dstRect`

On return, a pointer to the rectangle marking the intersection of the first two rectangles.

Return Value
`TRUE` if the specified rectangles intersect or `FALSE` if they do not.

Discussion

The `SectRect` function calculates the rectangle that delineates the intersection of the two rectangles you specify in the `src1` and `src2` parameters. Rectangles that touch at a line or a point are not considered intersecting, because their intersection rectangle (actually, in this case, an intersection line or point) does not enclose any pixels in the bit image.

If the rectangles do not intersect, the destination rectangle is set to (0,0,0,0). The `SectRect` function works correctly even if one of the source rectangles is also the destination.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

GlyphaIVOld
HTMLSample
MovieBrowser
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

SectRgn

Calculates the intersection of two regions. (Available in OS X v10.0 through OS X v10.6.)

```
void SectRgn (
   RgnHandle srcRgnA,
   RgnHandle srcRgnB,
   RgnHandle dstRgn
);
```

Parameters

`srcRgnA`

A handle to the first of two regions whose intersection is to be determined.

`srcRgnB`

A handle to the second of two regions whose intersection is to be determined.

`dstRgn`

On return, a handle to the region holding the intersection area. If the regions do not intersect, or one of the regions is empty, `SectRgn` sets the destination to the empty region defined by the rectangle (0,0,0,0).

The `SectRgn` function does not create a destination region; you must have already allocated memory for it by using the `NewRgn` (page 336) function.

The destination region may be one of the source regions, if desired.

Discussion

The SectRgn procedure calculates the intersection of the two regions whose handles you pass in the srcRgnA and srcRgnB parameters, and it places the intersection in the region whose handle you pass in the dstRgn parameter. If the regions do not intersect, or one of the regions is empty, SectRgn sets the destination to the empty region defined by the rectangle (0,0,0,0).

The SectRgn procedure does not create a destination region; you must have already allocated memory for it by using the NewRgn function.

The destination region may be one of the source regions, if desired.

Special Considerations

The `SectRgn` function may temporarily use heap space that's twice the size of the two input regions.

The `SectRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Password

QTMusicToo

TE Over Background

Declared in

`QuickdrawAPI.h`

SeedCFill

Determines how far filling will extend to pixels matching the color of a particular pixel. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SeedCFill (
   const BitMap *srcBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short seedH,
   short seedV,
   ColorSearchUPP matchProc,
   long matchData
);
```

Parameters

`srcBits`

The source image. If the image is in a pixel map, you must coerce its `PixMap` structure to a `BitMap` structure.

`dstBits`

On return, the destination mask.

`srcRect`

The rectangle of the source image.

`dstRect`

The rectangle of the destination image.

`seedH`

The horizontal position of the seed point.

`seedV`

The vertical position of the seed point.

`matchProc`

An optional color search function.

`matchData`

Data for the optional color search function.

Discussion

The `SeedCFill` function generates a mask showing where the pixels in an image can be filled from a starting point, like the paint pouring from the MacPaint paint-bucket tool. This mask is a bitmap filled with 1's to indicate all pixels adjacent to a seed point whose colors do not exactly match the `RGBColor` structure for the pixel at the seed point. You can then use this mask with the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions.

You specify a source image in the `srcBits` parameter and, in the `srcRect` parameter, specify a rectangle within that source image. You specify where to begin seeding in the `seedH` and `seedV` parameters, which must be the horizontal and vertical coordinates of a point in the local coordinate system of the source bitmap. By default, the 1's returned in the mask indicate all pixels adjacent to the seed point whose pixel values do not exactly match the pixel value of the pixel at the seed point. To use this default, set the `matchProc` and `matchData` parameters to 0.

In generating the mask, `SeedCFill` uses the `CopyBits` function to convert the source image to a 1-bit mask. The `SeedCFill` function installs a default color search function that returns 0 if the pixel value matches that of the seed point all other pixel values return 1's.

The `SeedCFill` function does not scale so the source and destination rectangles must be the same size. Calls to `SeedCFill` are not clipped to the current port and are not stored into QuickDraw pictures.

To customize `SeedCFill`, write your own color search function and point to it in the `matchProc` parameter; `SeedCFill` will then use your function instead of the default.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

SeedFill

Determines how far filling will extend from a seeding point. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SeedFill (
   const void *srcPtr,
   void *dstPtr,
   short srcRow,
   short dstRow,
```



```
   short height,
   short words,
   short seedH,
   short seedV
);
```

Parameters

`srcPtr`

A pointer to the source bit image.

`dstPtr`

On input, a pointer to the destination bit image; upon return, a pointer to the bitmap containing the resulting mask.

`srcRow`

Row width of the source bitmap.

`dstRow`

Row width of the destination bitmap.

`height`

Height (in pixels) of the fill rectangle.

`words`

Width (in words) of the fill rectangle.

`seedH`

The horizontal offset (in pixels) at which to begin filling the destination bit image.

`seedV`

The vertical offset (in pixels) at which to begin filling the destination bit image.

Discussion

The `SeedFill` function produces a mask showing where bits in an image can be filled from a starting point, like the paint pouring from the MacPaint paint-bucket tool. The `SeedFill` returns this mask in the `dstPtr` parameter. This mask is a bitmap filled with 1's only where the pixels in the source image can be filled. You can then use this mask with the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions.

Point to the bit image you want to fill with the `srcPtr` parameter, which can point to the image's base address or a word boundary within the image. Specify a pixel height and word width with the `height` and `words` parameters to define a fill rectangle that delimits the area you want to fill. The fill rectangle can be the entire bit image or a subset of it. Point to a destination image with the `dstPtr` parameter. Specify the row widths of the source and destination bitmaps (their `rowBytes` values) with the `srcRow` and `dstRow` parameters. (The bitmaps can be different sizes, but they must be large enough to contain the fill rectangle at the origins specified by the `srcPtr` and `dstPtr` parameters.)

You specify where to begin filling with the `seedH` and `seedV` parameters: they specify a horizontal and vertical offset in pixels from the origin of the image pointed to by the `srcPtr` parameter. The `SeedFill` function calculates contiguous pixels from that point out to the boundaries of the fill rectangle, and it stores the result in the bit image pointed to by the `dstPtr` parameter.

Calls to `SeedFill` are not clipped to the current port and are not stored into QuickDraw pictures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetCCursor

Specifies a color cursor for display on the screen. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetCCursor (
   CCrsrHandle cCrsr
);
```

Parameters
`cCrsr`
A handle to the color cursor to be displayed.

Discussion
At the time the cursor is set, it is expanded to the current screen depth so that it can be drawn rapidly. You must call `GetCCursor` before you call `SetCCursor`; however, you can make several subsequent calls to `SetCCursor` once `GetCCursor` creates the `CCrsr` structure.

If your application has changed the cursor's data or its color table, it must also invalidate the `crsrXValid` and `crsrID` fields of the `CCrsr` structure before calling `SetCCursor`.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
NewCCursor
qtshoweffect
qtshoweffect.win

Declared in
`QuickdrawAPI.h`

SetClientID

Sets the `gdID` field in the current `GDevice` data structure to identify this client program to its search and complement functions. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetClientID (
   short id
);
```

Parameters
`id`
The ID to be set in the device data structure.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetClip

Changes the clipping region of the current graphics port (basic or color) to a region you specify. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetClip (
```



```
   RgnHandle rgn
);
```

Parameters

`rgn`

A handle to a region. The `SetClip` function makes this region the clipping region of the current graphics port. The `SetClip` function doesn't change the region handle, but instead affects the clipping region itself.

Discussion

Since `SetClip` copies the specified region into the current graphics port's clipping region, any subsequent changes you make to the region specified in the `rgn` parameter do not affect the clipping region of the graphics port.

The initial clipping region of a graphics port is an arbitrarily large rectangle. You can set the clipping region to any arbitrary region, to aid you in drawing inside the graphics port—for example, to avoid drawing over scroll bars when drawing into a window, you could define a clipping region that excludes the scroll bars.

You can use the `GetClip` and `SetClip` functions to preserve the current clipping region: use `GetClip` to save the current port's clipping region, and use `SetClip` to restore it.

All other system software functions preserve the current clipping region.

The `SetClip` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

AlwaysPreview
HTMLSample
HTMLUserPane
Live Scroll
TE Over Background

Declared in

`QuickdrawAPI.h`

SetCPixel

Sets the color of an individual pixel to the color that most closely matches the RGB color that you specify in the `cPix` *parameter. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)*

```
void SetCPixel (
   short h,
   short v,
   const RGBColor *cPix
);
```

Parameters

`h`

The horizontal coordinate of the point at the upper-left corner of the pixel.

`v`

The vertical coordinate of the point at the upper-left corner of the pixel.

`cPix`

An `RGBColor` structure.

Discussion

On an indexed color system, the `SetCPixel` function sets the pixel value to the index of the best-matching color in the current device's CLUT. In a direct environment, the `SetCPixel` function sets the pixel value to a 16-bit or 32-bit direct pixel value.

To determine the color of an individual pixel, use the `GetCPixel` (page 229) function.

Special Considerations

The `SetCPixel` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

SetCursor

Sets the current cursor. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetCursor (
   const Cursor * crsr
);
```

Parameters

`crsr`

A `Cursor` (page 86) structure for the cursor to be displayed.

Discussion

If the cursor is hidden, it remains hidden and attains its new appearance only when it's uncovered. If the cursor is already visible, it changes to the new appearance immediately.

You need to use the `InitCursor` (page 279) function to initialize the standard arrow cursor and make it visible on the screen before you call `SetCursor` to change the cursor's appearance.

To display a color cursor, use the `SetCCursor` (page 401) function.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

CopyBits vs. CopyMask
NewCCursor
Password
QTMusicToo
TE Over Background

Declared in

`QuickdrawAPI.h`

SetCursorComponent

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
OSErr SetCursorComponent (
   ComponentInstance ci
);
```

Return Value

Carbon Porting Notes
This function is not implemented on Mac OS X.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetDeviceAttribute

Sets the attribute bits of a `GDevice` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetDeviceAttribute (
   GDHandle gdh,
   short attribute,
   Boolean value
);
```

Parameters

`gdh`

A handle to a `GDevice` structure.

`attribute`

One of the specific constants, which represent bits in the `gdFlags` field of a `GDevice` structure. See GDevice (page 91) for the values you can use in this parameter.

`value`

A value of either 0 or 1 for the flag bit specified in the `attribute` parameter.

Discussion

For the graphics device specified in the `gdh` parameter, the `SetDeviceAttribute` function sets the flag bit specified in the `attribute` parameter to the value specified in the `value` parameter.

Your application should never directly change the `gdFlags` field of the `GDevice` structure; instead, use only the `SetDeviceAttribute` function.

Special Considerations

The `SetDeviceAttribute` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Play Video Sample

Declared in

`QuickdrawAPI.h`

SetEmptyRgn

Sets an existing region to be empty. (Available in OS X v10.0 through OS X v10.6.)

```
void SetEmptyRgn (
   RgnHandle rgn
);
```

Parameters

`rgn`

A handle to the region to be made empty.

Discussion

The `SetEmptyRgn` function destroys the previous structure of the region whose handle you pass in the `rgn` parameter; it then sets the new structure to the empty region defined by the rectangle (0,0,0,0).

Special Considerations

The `SetEmptyRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CarbonTransparentWindow

CustomWindow

kcapApp

TypeServicesForUnicode

Declared in
`QuickdrawAPI.h`

SetEntries

Sets a group of color table entries for the current `GDevice` data structure. This function is used by system software and your application should not need to call it. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetEntries (
   short start,
   short count,
   CSpecArray aTable
);
```

Parameters
`start`

The index of the first entry to be changed.

`count`

The number of entries to be changed. Note that all values are zero-based; for example, to set three entries, pass 2 in the count parameter.

`aTable`

An array of `ColorSpec` data structures containing the colors to be used. Directly specify a `cSpecArray` structure, not the beginning of a color table. The `ColorSpec.value` fields of the entries must be in the logical range for the target device's assigned pixel depth. Thus, with a 4-bit pixel size, the `ColorSpec.value` fields should be in the range 1 to 15. With an 8-bit pixel size, the range is 0 to 255.

Discussion
Instead of using `SetEntries`, you should use the Palette Manager function `SetEntryColor` to allow your application to run in a multiscreen or multitasking environment.

The SetEntries positional information works in logical space rather than in the actual memory space used by the hardware. Requesting a change at the fourth position in the color table may not modify the fourth color table entry in the hardware, but it does correctly change the color on the screen for any pixels with a value of 4 in the video card. The SetEntries mode characterized by a start position and a length is called sequence mode. In this case, SetEntries sequentially loads new colors into the hardware in the same order as they appear in the aTable parameter, copies the clientID fields for changed color table entries from the current GDevice data structure's gdID field, and ignores the ColorSpec.value fields.

The other SetEntries mode is called index mode. It allows the cSpecArray structure to specify where the data will be installed on an entry-by-entry basis. To use this mode, pass –1 for the start position, with a valid count and a pointer to the cSpecArray data structure. Each entry is installed into the color table at the position specified by the ColorSpec.value field of each entry in the cSpecArray data structure. In the current GDevice data structure's color table, the ColorSpec.value fields of all changed entries are assigned the GDevice data structure's gdID value.

When the Color Manager changes color table entries, it invalidates all cached fonts, and changes the color table's seed number so that the next drawing operation triggers the Color Manager to rebuild the inverse table. If any of the requested entries are protected or out of range, the Color Manager returns a protection error, and nothing happens. The Color Manager changes a reserved entry only if the current gdID field of the current GDevice data structure matches the low byte of the intended ColorSpec.value field in the color table.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
QuickdrawAPI.h

SetGDevice

Sets a GDevice structure as the current device. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetGDevice (
   GDHandle gd
);
```


Parameters

`gd`

A handle to a `GDevice` structure.

Discussion

Your application won't generally need to use this function, because when your application draws into a window on one or more screens, Color QuickDraw automatically switches `GDevice` structures as appropriate; and when your application needs to draw into an offscreen graphics world, it can use the `SetGWorld` function to set the graphics port as well as the `GDevice` structure for the offscreen environment. However, if your application uses the `SetPort` function instead of the `SetGWorld` function to set the graphics port to or from an offscreen graphics world, then your application must use `SetGDevice` in conjunction with `SetPort`.

A handle to the currently active device is kept in the global variable `TheGDevice`.

Special Considerations

The `SetGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

DrawSprocketTestOld

GlyphaIVOld

Declared in

`QuickdrawAPI.h`

SetGWorld

Changes the current graphics port (basic, color, or offscreen). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetGWorld (
   CGrafPtr port,
   GDHandle gdh
);
```


Parameters

`port`

A pointer to an offscreen graphics world, color graphics port, or basic graphics port. Specify values of type `GrafPtr`, `CGrafPtr`, or `GWorldPtr`, depending on whether you want to set the current graphics port to be a basic graphics port, color graphics port, or offscreen graphics world. Any drawing your application performs then occurs in this graphics port.

`gdh`

A handle to a `GDevice` structure. If you pass a pointer to an offscreen graphics world in the `port` parameter, set this parameter to `NULL`, because `SetGWorld` ignores this parameter and sets the current device to the device attached to the offscreen graphics world.

Discussion

The `SetGWorld` function sets the current graphics port to the one specified by the `port` parameter and—unless you set the current graphics port to be an offscreen graphics world—sets the current device to that specified by the `gdh` parameter.

Special Considerations

The `SetGWorld` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qteffects

qteffects.win

qtspritesplus

qtwiredactions

qtwiredspritesjr.win

Declared in

`QDOffscreen.h`

SetOrigin

Changes the coordinates of the window origin of the port rectangle of the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void SetOrigin (
   short h,
   short v
);
```

Parameters

`h`

The horizontal coordinate of the upper-left corner of the port rectangle.

`v`

The vertical coordinate of the upper-left corner of the port rectangle.

Discussion

The `SetOrigin` function changes the coordinates of the upper-left corner of the current graphics port's port rectangle to the values supplied by the `h` and `v` parameters. All other points in the current graphics port's local coordinate system are calculated from this point. All subsequent drawing and calculation functions use the new coordinate system.

The `SetOrigin` function does not affect the screen; it does, however, affect where subsequent drawing inside the graphics port appears. The `SetOrigin` function does not offset the coordinates of the clipping region or the graphics pen, which therefore change position on the screen (unlike the boundary rectangle, port rectangle, and visible region, which don't change position onscreen).

Because `SetOrigin` does not move the window's clipping region, use the `GetClip` function to store your clipping region immediately after your first call to `SetOrigin`—if you use clipping regions in your windows. Before calling your own window-drawing function, use the `ClipRect` function to define a new clipping region—to avoid drawing over your scroll bars, for example. After calling your own window-drawing function, use the `SetClip` function to restore the original clipping region. You can then call `SetOrigin` again to restore the window origin to a horizontal coordinate of 0 and a vertical coordinate of 0 with your original clipping region intact.

All other functions in the Macintosh Toolbox and Operating System preserve the local coordinate system of the current graphics port. The `SetOrigin` function is useful for readjusting the coordinate system after a scrolling operation.

Note that the Window Manager and Control Manager always assume the window's upper-left point has a horizontal coordinate of 0 and a vertical coordinate of 0 when they draw in a window. Therefore, if you use `SetOrigin` to change the window origin, be sure to use `SetOrigin` again to return the window origin to a horizontal coordinate of 0 and a vertical coordinate of 0 before using any Window Manager or Control Manager functions.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CarbonQuartzDrawingWPrinting
HTMLSample
Password
QTMusicToo
TE Over Background

Declared in
`QuickdrawAPI.h`

SetPenState

Restores the state of the graphics pen that was saved with the `GetPenState` function. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetPenState (
   const PenState *pnState
);
```

Parameters
`pnState`

A `PenState` structure previously created with the `GetPenState` function. The `SetPenState` function sets the graphics pen's location, size, pattern, and pattern mode in the current graphics port to the values stored in this structure.

Discussion
This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DialogsToHIViews
FinderDragPro
HTMLSample
qtmovietrack.win

Declared in
`QuickdrawAPI.h`

SetPixelsState

Restores an offscreen pixel image to the state that you saved with the `GetPixelsState` function. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPixelsState (
   PixMapHandle pm,
   GWorldFlags state
);
```

Parameters

`pm`

A handle to an offscreen pixel map.

`state`

Flags, which you usually save with the `GetPixelsState` function. You can use either of the constants `pixelsPurgeable` or `pixelsLocked here.`

Because only an unlocked memory block can be purged, `SetPixelsState` calls the `UnlockPixels` (page 451) and `AllowPurgePixels` (page 149) functions if the `state` parameter specifies the `pixelsPurgeable` flag. If the `state` parameter does not specify the `pixelsPurgeable` flag, `SetPixelsState` makes the base address for the offscreen pixel image unpurgeable.

If the `state` parameter does not specify the `pixelsLocked` flag, `SetPixelsState` allows the base address for the offscreen pixel image to be moved.

Discussion
The `SetPixelsState` function changes the state of the memory allocated for an offscreen pixel image to the state indicated in the `state` parameter.

After using `GetPixelsState` and before using `SetPixelsState`, your application can temporarily alter the offscreen graphics world by using the `AllowPurgePixels` (page 149) function to temporarily mark the memory block for its offscreen pixel map as purgeable, the `NoPurgePixels` (page 339) function to make it unpurgeable, the `LockPixels` (page 314) function to prevent it from being moved, and the `UnlockPixels` (page 451) function to unlock it.

Special Considerations
The `SetPixelsState` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QDOffscreen.h`

SetPort

Changes the current graphics port (basic or color). (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
virtual void SetPort (
   void *port
);
```

Parameters
`port`

A pointer to a `GrafPort` structure. Typically, you pass a pointer to a `GrafPort` structure that you previously saved with the `GetPort` function. The `SetPort` function sets this structure to be the current graphics port.

Discussion
All QuickDraw drawing functions affect the bitmap of, and use the local coordinate system of, the current graphics port. Each graphics port has its own graphics pen and text characteristics, which remain unchanged when the graphics port isn't selected as the current graphics port.

When your application runs in Color QuickDraw or uses offscreen graphics worlds, it should use the `SetGWorld` function instead of `SetPort`. The `SetGWorld` function restores the current graphics port for basic and color graphics ports as well as offscreen graphics worlds.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
Fragment Tool
GlyphaIVOld
HTMLUserPane
QTMusicToo

Declared in
`QuickdrawAPI.h`

SetPortBackPixPat

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortBackPixPat (
   CGrafPtr port,
   PixPatHandle backPattern
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortBits

Sets the bitmap for the current basic graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetPortBits (
   const BitMap *bm
);
```

Parameters

`bm`

A pointer to the `BitMap` structure to set for the current graphics port. Be sure to prepare all fields of the `BitMap` structure before you call `SetPortBits`.

Discussion

You should never need to use this function. This function, created for early versions of QuickDraw, allows you to perform all normal drawing and calculations on a buffer other than the screen—for example, copying a small offscreen image onto the screen with the `CopyBits` function. However, instead of using `SetPortBits`, you should use the more powerful offscreen graphics capabilities.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

GlyphaIVOld

Declared in

`QuickdrawAPI.h`

SetPortBounds

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetPortBounds (
   CGrafPtr port,
   const Rect *rect
);
```


Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortClipRegion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortClipRegion (
   CGrafPtr port,
   RgnHandle clipRgn
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CGDrawPicture

Declared in
`QuickdrawAPI.h`

SetPortCustomXFerProc

(Available in OS X v10.0 through OS X v10.6.)


```
OSErr SetPortCustomXFerProc (
   CGrafPtr port,
   CustomXFerProcPtr proc,
   UInt32 flags,
   UInt32 refCon
);
```

Return Value

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortFillPixPat

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortFillPixPat (
   CGrafPtr port,
   PixPatHandle penPattern
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortFracHPenLocation

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void SetPortFracHPenLocation (
   CGrafPtr port,
   short pnLocHFrac
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortGrafProcs

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortGrafProcs (
   CGrafPtr port,
   CQDProcsPtr procs
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites
DesktopSprites.win
qteffects.win
qtspritesplus
qtwiredspritesjr.win

Declared in
`QuickdrawAPI.h`

SetPortOpColor

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortOpColor (
   CGrafPtr port,
   const RGBColor *opColor
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortPenMode

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortPenMode (
   CGrafPtr port,
   SInt32 penMode
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CarbonCustomList

Declared in
`QuickdrawAPI.h`

SetPortPenPixPat

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortPenPixPat (
   CGrafPtr port,
   PixPatHandle penPattern
);
```

Carbon Porting Notes
Use this new accessor function in place of direct access to structures.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortPenSize

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortPenSize (
   CGrafPtr port,
   Point penSize
);
```


Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

SetPortPix

Sets the pixel map for the current color graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortPix (
   PixMapHandle pm
);
```

Parameters

`pm`

A handle to the `PixMap` structure.

Discussion

The `SetPortPix` function replaces the `portPixMap` field of the current `CGrafPort` structure with the handle you specify in the `pm` parameter.

Typically, your application does not need to call this function.

The `SetPortPix` function is analogous to the basic QuickDraw function `SetPortBits`, which sets the bitmap for the current basic graphics port. The `SetPortPix` function has no effect when used with a basic graphics port. Similarly, `SetPortBits` has no effect when used with a color graphics port.

Both `SetPortPix` and `SetPortBits` allow you to perform drawing and calculations on a buffer other than the screen. However, instead of using these functions, use the offscreen graphics capabilities.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Inside Mac Movie TB Code

Declared in
`QuickdrawAPI.h`

SetPortVisibleRegion

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortVisibleRegion (
   CGrafPtr port,
   RgnHandle visRgn
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPt

Assigns two coordinates to a point. (Available in OS X v10.0 through OS X v10.6.)

```
void SetPt (
   Point *pt,
   short h,
   short v
);
```

Parameters
`pt`

A pointer to the point to be given new coordinates. On return, this point is assigned the horizontal coordinate you specify in the `h` parameter and the vertical coordinate you specify in the `v` parameter.

h

The horizontal value of the new coordinates.

v

The vertical value of the new coordinates.

Discussion

The SetPt procedure assigns the horizontal coordinate specified in the h parameter and the vertical coordinate specified in the v parameter to the point returned in the pt parameter.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

Carbon Porting Tutorial

CopyBits vs. CopyMask

FinderDragPro

Fragment Tool

MoreOSL

Declared in

`QuickdrawAPI.h`

SetQDError

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetQDError (
   OSErr err
);
```

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

SetQDGlobalsArrow

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetQDGlobalsArrow (
   const Cursor *arrow
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetQDGlobalsRandomSeed

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetQDGlobalsRandomSeed (
   long randomSeed
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetRect

Assigns coordinates to a rectangle. (Available in OS X v10.0 through OS X v10.6.)


```
void SetRect (
   Rect * r,
   short left,
   short top,
   short right,
   short bottom
);
```

Parameters

`r`

A pointer to the rectangle to set.

`left`

The horizontal coordinate of the new upper-left corner of the rectangle.

`top`

The vertical coordinate of the new upper-left corner of the rectangle.

`right`

The horizontal coordinate of the new lower-right corner of the rectangle.

`bottom`

The vertical coordinate of the new lower-right corner of the rectangle.

Discussion

The `SetRect` function assigns the coordinates you specify in the `left`, `top`, `right`, and `bottom` parameters to the rectangle that you specify in the `r` parameter. This function is provided to help you shorten your program text. If you want a more readable text, at the expense of source text length, you can instead assign integers (or points) directly into the fields of a `Rect` structure.

You can use a rectangle to specify locations and sizes for various graphics operations.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

Carbon Porting Tutorial

GlyphaIVOld

HTMLSample

Out of This GWorld
QTMusicToo

Declared in
`QuickdrawAPI.h`

SetRectRgn

Changes the structure of an existing region to that of a rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void SetRectRgn (
   RgnHandle rgn,
   short left,
   short top,
   short right,
   short bottom
);
```

Parameters

`rgn`
A handle to the region to restructure as a rectangle.

`left`
The horizontal coordinate of the upper-left corner of the rectangle to set as the new region.

`top`
The vertical coordinate of the upper-left corner of the rectangle to set as the new region.

`right`
The horizontal coordinate of the lower-right corner of the rectangle to set as the new region.

`bottom`
The vertical coordinate of the lower-right corner of the rectangle to set as the new region.

Discussion
The `SetRectRgn` function destroys the previous structure of the region whose handle you pass in the `rgn` parameter, and it then sets the new structure to the rectangle that you specify in the `left`, `top`, `right`, and `bottom` parameters. If you specify an empty rectangle (that is, `right` is greater than or equal to `left` or `bottom = top`), the `SetRectRgn` function sets the region to the empty region defined by the rectangle (0,0,0,0).

As an alternative to the `SetRectRgn` function, you can change the structure of an existing region to that of a rectangle by using the `RectRgn` (page 384) function, which accepts as a parameter a rectangle instead of four coordinates.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

The `SetRectRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Password

TE Over Background

Declared in

`QuickdrawAPI.h`

SetStdCProcs

Obtains a `CQDProcs` structure with fields that point to QuickDraw's standard low-level functions, which you can modify to change QuickDraw's standard low-level behavior. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetStdCProcs (
   CQDProcs *procs
);
```

Parameters

`procs`

Upon completion, a `CQDProcs` structure with fields that point to QuickDraw's standard low-level functions. You can change one or more fields to point to your own functions and then set the color graphics port to use this modified `CQDProcs` (page 83) structure.

Discussion

For each shape that QuickDraw can draw, certain functions perform basic graphics operations on the shape: framing, painting, erasing, inverting, and filling. These functions, in turn, call a low-level drawing function for the shape.

The `grafProcs` field determines which low-level functions are called. If that field contains a value of `NULL`, the standard functions are called. You can set the `grafProcs` field to point to a structure of pointers to your own functions, and either completely override the standard ones or call them after your functions have modified their parameters as necessary.

The `SetStdCProcs` function sets all the fields of the `CQDProcs` structure to point to the standard functions. You can then reset the ones with which you are concerned.

The functions you install in the `CDQProcs` structure must have the same calling sequences as the standard basic QuickDraw functions.

When drawing in a color graphics port, your application must always use `SetStdCProcs` instead of `SetStdProcs`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

DesktopSprites

DesktopSprites.win

MovieSprites

qtspritesplus

WiredSprites

Declared in

`QuickdrawAPI.h`

SetStdProcs

Obtains a `QDProcs` structure with fields that point to basic QuickDraw's standard low-level functions, which you can modify to point to your own functions. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void SetStdProcs (
   QDProcs *procs
);
```

Parameters

`procs`

On return, a pointer to a `QDProcs` structure with fields that point to basic QuickDraw's standard low-level functions. You can change one or more fields of this structure to point to your own functions and then set the basic graphics port to use this modified `QDProcs` structure. By changing these pointers, you can install your own functions, and either completely override the standard ones or call them after your functions have modified their parameters as necessary.

Discussion

The functions you install in this `QDProcs` structure must have the same calling sequences as the standard functions.

Special Considerations

The Color QuickDraw function `SetStdCProcs` is analogous to the `SetStdProcs` function, which you should use with computers that support only basic QuickDraw. When drawing in a color graphics port, your application must always use `SetStdCProcs` instead of `SetStdProcs`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

TE Over Background

Declared in

`QuickdrawAPI.h`

ShieldCursor

Hides the cursor in a rectangle. (Available in OS X v10.0 through OS X v10.6.)

```
void ShieldCursor (
   const Rect *shieldRect,
   Point offsetPt
);
```


Parameters

`shieldRect`

A rectangle in which the cursor is hidden whenever the cursor intersects the rectangle. The rectangle may be specified in global or local coordinates. If you are using global coordinates, pass (0,0) in the `offsetPt` parameter. If you are using the local coordinates of a graphics port, pass the coordinates for the upper-left corner of the graphics port's boundary rectangle in the `offsetPt` parameter.

`offsetPt`

A point value for the offset of the rectangle. Like the basic QuickDraw function `LocalToGlobal`, the `ShieldCursor` function offsets the coordinates of the rectangle by the coordinates of this point.

Discussion

If the cursor and the given rectangle intersect, `ShieldCursor` hides the cursor. If they do not intersect, the cursor remains visible while the mouse is not moving, but is hidden when the mouse moves. Use this function with a feature such as QuickTime to display content in a specified rectangle. When a QuickTime movie is animating, the cursor should not be visible in front of the movie.

The `ShieldCursor` function decrements the cursor level and should be balanced by a call to the `ShowCursor` function.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

ShowCursor

Displays a cursor hidden by the `HideCursor` or `ShieldCursor` functions. (Available in OS X v10.0 through OS X v10.6.)

```
void ShowCursor (
   void
);
```

Discussion

`ShowCursor` increments the cursor level, which has been decremented by the `HideCursor` (page 276) or `ShieldCursor` (page 431) function and displays the cursor on the screen when the level is 0. A call to the `ShowCursor` function balances each previous call to the `HideCursor` or `ShieldCursor` function. The level is not incremented beyond 0, so extra calls to `ShowCursor` have no effect.

Low-level interrupt-driven functions link the cursor with the mouse position, so that if the cursor level is 0 and visible, the cursor automatically follows the mouse.

If the cursor has been changed with the `SetCursor` (page 405) function while hidden, `ShowCursor` displays the new cursor.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
Carbon Porting Tutorial
GlyphaIVOld
Simple DrawSprocket

Declared in
`QuickdrawAPI.h`

ShowPen

Changes the ink of a graphics pen from invisible to visible, making pen drawing appear on the screen. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void ShowPen (
   void
);
```

Discussion
`ShowPen` is called by the functions `CloseRgn` (page 162) , `ClosePoly` (page 162) , and `ClosePicture`.

The `ShowPen` function increments the `pnVis` field of the current graphics port. For 0 or positive values, the pen drawing shows on the screen.

For example, if you have used the `HidePen` function to decrement the `pnVis` field from 0 to –1, use the `ShowPen` function to make its value 0 so that QuickDraw resumes drawing on the screen. Subsequent calls to `ShowPen` increment `pnVis` beyond 0, so every call to `ShowPen` should be balanced by a call to `HidePen`.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
MouseTracking

Declared in
`QuickdrawAPI.h`

SlopeFromAngle

Converts an angle value to a slope value. (Available in OS X v10.0 through OS X v10.6.)

```
Fixed SlopeFromAngle (
   short angle
);
```

Parameters
`angle`

The angle, expressed in clockwise degrees from 12 o'clock and treated `MOD` 180. (90 degrees is thus at 3 o'clock and –90 degrees is at 9 o'clock.

Return Value
The slope corresponding to the angle specified in the `angle` parameter. Slopes are defined as Dx/Dy, the horizontal change divided by the vertical change between any two points on a line with the given angle. The negative y-axis is defined as being at 12 o'clock, and the positive y-axis at 6 o'clock. The x-axis is defined as usual, with the positive side defined as being at 3 o'clock.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

StdArc

QuickDraw's standard low-level function for drawing an arc or a wedge. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdArc (
   GrafVerb verb,
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

`verb`

The action to perform. See "Verb Constants" (page 144).

`r`

The rectangle to contain the arc.

`startAngle`

The beginning angle.

`arcAngle`

The ending angle.

Discussion

Using the action specified in the `verb` parameter, the `StdArc` function draws an arc or wedge of the oval that fits inside the rectangle specified in the `r` parameter. The arc or wedge is bounded by the radii specified in the `startAngle` and `arcAngle` parameters.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

StdBits

QuickDraw's standard low-level function for transferring bits and pixels. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void StdBits (
   const BitMap *srcBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn
);
```

Parameters

`srcBits`

A pointer to a bitmap or pixel map containing the image to copy.

`srcRect`

A pointer to the source rectangle.

`dstRect`

The destination rectangle.

`mode`

The source mode for the copy.

`maskRgn`

A handle to a region acting as a mask for the transfer.

Discussion

The `StdBits` function transfers a bit or pixel image between the bitmap or pixel map specified in the `srcBits` parameter and bitmap of the current graphics port, just as if the `CopyBits` function were called with the same parameters and with a destination bitmap equal to `(* thePort).portBits`.

You should only call this low-level function from your customized QuickDraw functions.

See `CopyBits` (page 165) for a discussion of the destination bitmap and of the `srcBits`, `srcRect`, `dstRect`, `mode`, and `maskRgn` parameters

Special Considerations

The `StdBits` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
DesktopSprites
qteffects.win
qtspritesplus
qtwiredactions
qtwiredspritesjr.win

Declared in
`QuickdrawAPI.h`

StdComment

QuickDraw's standard low-level function for processing a picture comment. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdComment (
   short kind,
   short dataSize,
   Handle dataHandle
);
```

Parameters

`kind`

The type of comment.

`dataSize`

The size of additional data, in bytes.

`dataHandle`

A handle to additional data.

Discussion

If there's no additional data for the comment, the value of the `dataHandle` parameter is `NULL` and the value of the `dataSize` parameter is 0. The `StdComment` function simply ignores the comment.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdComment` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

StdGetPic

QuickDraw's standard low-level function for retrieving information from the definition of a picture. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void StdGetPic (
   void *dataPtr,
   short byteCount
);
```

Parameters
`dataPtr`

On return, a pointer to the collected picture data.

`byteCount`

The size of the picture data.

Discussion
The `StdGetPic` function retrieves from the definition of the currently open picture the next number of bytes as specified in the `byteCount` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

StdLine

QuickDraw's standard low-level function for drawing a line. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdLine (
   Point newPt
);
```

Parameters

`newPt`

The point to which to draw the line.

Discussion

The `StdLine` function draws a line from the current pen location to the location (in local coordinates) specified in the `newPt` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdLine` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

ReKeyTrans

Declared in

`QuickdrawAPI.h`

StdOpcode

(Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdOpcode (
   const Rect *fromRect,
   const Rect *toRect,
```



```
   UInt16 opcode,
   SInt16 version
);
```

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

StdOval

QuickDraw's standard low-level function for drawing an oval. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdOval (
   GrafVerb verb,
   const Rect *r
);
```

Parameters
`verb`
The action to perform. See "Verb Constants" (page 144).

`r`
The rectangle to contain the oval.

Discussion
The `StdOval` function draws an oval inside the given rectangle specified in the `r` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations
The `StdOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

StdPoly

QuickDraw's standard low-level function for drawing a polygon. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdPoly (
   GrafVerb verb,
   PolyHandle poly
);
```

Parameters

`verb`

The action to perform. See "Verb Constants" (page 144).

`poly`

A handle to the polygon data.

Discussion

The `StdPoly` function draws the polygon specified in the `poly` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

StdPutPic

QuickDraw's standard low-level function for saving information as the definition of a picture. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdPutPic (
   const void *dataPtr,
   short byteCount
);
```

Parameters

`dataPtr`

A pointer to the collected picture data.

`byteCount`

The size of the picture data.

Discussion

The `StdPutPic` function saves as the definition of the currently open picture the drawing commands stored in the data structure pointed to by the `dataPtr` parameter, starting with the first byte and continuing for the next number of bytes as specified in the `byteCount` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdPutPic` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

StdRect

QuickDraw's standard low-level function for drawing a rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void StdRect (
   GrafVerb verb,
   const Rect *r
);
```

Parameters

`verb`

The action to perform. See "Verb Constants" (page 144).

`r`

The rectangle to draw.

Discussion

The `StdRect` function draws the rectangle specified in the `r` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
TE Over Background

Declared in
`QuickdrawAPI.h`

StdRgn

QuickDraw's standard low-level function for drawing a region. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdRgn (
   GrafVerb verb,
   RgnHandle rgn
);
```


Parameters

`verb`

The action to perform. See "Verb Constants" (page 144).

`rgn`

A handle to the region data.

Discussion

The `StdRgn` function draws the region specified in the `rgn` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

TE Over Background

Declared in

`QuickdrawAPI.h`

StdRRect

QuickDraw's standard low-level function for drawing a rounded rectangle. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StdRRect (
   GrafVerb verb,
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```


Parameters

`verb`

The action to perform. See "Verb Constants" (page 144).

`r`

The rectangle to draw.

`ovalWidth`

The width diameter for the corner oval.

`ovalHeight`

The height diameter for the corner oval.

Discussion

The `StdRRect` function draws the rounded rectangle specified in the `r` parameter according to the action specified in the `verb` parameter. The `ovalWidth` and `ovalHeight` parameters specify the diameters of curvature for the corners.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdRRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

StuffHex

Sets byte values into memory. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void StuffHex (
   void *thingPtr,
   ConstStr255Param s
);
```


Parameters

`thingPtr`

A pointer to any data structure in memory. If `thingPtr` is an odd address, then `thingPtr` is interpreted as pointing to the next word boundary.

`s`

A string of characters representing hexadecimal digits. All characters in this string must be hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Otherwise, `StuffHex` may set bytes in the data structure pointed to by `thingPtr` to arbitrary values. If there are an odd number of characters in the string, the last character is ignored.

Discussion

The `StuffHex` function sets bytes in memory beginning with that byte specified by the parameter `thingPtr`. The total number of bytes set is equivalent to half the length of the string, ignoring the last character if the number of characters is odd.

Each byte to be set corresponds to two characters in the string. These characters should represent hexadecimal digits. For example, the string ‘`D41A`’ results in 2 bytes being set to the values $D4 and $1A, respectively.

To copy a range of bytes from one memory location to another, you should ordinarily use the Memory Manager function, `BlockMove`.

Special Considerations

The `StuffHex` function does no range checking to ensure that bytes being set are within the bounds of a certain data structure. If you do not use `StuffHex` carefully, you may change memory in the partition of your application or another application in unpredictable ways.

Although the `StuffHex` function sets the value of individual bytes, it does not move relocatable blocks. Thus, you can call it at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

SubPt

Subtracts the coordinates of one point from another. (Available in OS X v10.0 through OS X v10.6.)


```
void SubPt (
   Point src,
   Point *dst
);
```

Parameters

`src`

A point, the coordinates of which are to be subtracted from the coordinates of the point specified in the `dst` parameter.

`dst`

The address of a point. Upon completion, the coordinates of this point contain the differences between the coordinates of the two points specified in the entry parameters.

If you pass `NULL` in the `dst` parameter, this function sets the `QDError` result code to `paramErr` and returns.

Availability

Available in OS X v10.0 through OS X v10.6.

Declared in

`QuickdrawAPI.h`

SyncCGContextOriginWithPort

Synchronizes the origin in a Quartz context with the lower-left corner of the associated graphics port. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus SyncCGContextOriginWithPort (
   CGContextRef inContext,
   CGrafPtr port
);
```

Parameters

`context`

A Quartz context associated with a graphics port. You can obtain such a context by calling `QDBeginCGContext` (page 59).

`port`

The graphics port associated with the context.

Return Value

A result code. If `noErr`, the context's origin was successfully changed.

Discussion

If you're using Quartz 2D to draw in a graphics port and `SetOrigin` (page 411) is called to change the port's origin, you can call this function to maintain the correspondence between the context's origin and the lower-left corner of the `portBounds` rectangle.

When you call this function:

1. The current transformation matrix (CTM) is reset to its default values. Any changes you made to the CTM prior to calling this function are lost.
2. The CTM is translated to establish the new origin, taking the port's current origin into account.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

ATSUIDirectAccessDemo

CarbonQuartzDrawingWPrinting

Declared in

`QuickdrawAPI.h`

TestDeviceAttribute

Determines whether the flag bit for an attribute has been set in the `gdFlags` field of a `GDevice` structure. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean TestDeviceAttribute (
   GDHandle gdh,
   short attribute
);
```

Parameters

`gdh`

A handle to a `GDevice` structure.

`attribute`

One of the specific constants, which represent bits in the `gdFlags` field of a `GDevice` structure. See "Device Attribute Constants" (page 125) for a description of the values you can use in this parameter.

Return Value

`TRUE` if the bit of the graphics device attribute specified in the `attribute` parameter is set to 1. Otherwise, `TestDeviceAttribute` returns `FALSE`.

Discussion

Use the `SetDeviceAttribute` (page 406) function to change any of the flags tested by the `TestDeviceAttribute` function.

Special Considerations

The `TestDeviceAttribute` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

Declared in

`QuickdrawAPI.h`

UnionRect

Calculates the smallest rectangle that encloses two rectangles. (Available in OS X v10.0 through OS X v10.6.)

```
void UnionRect (
   const Rect * src1,
   const Rect * src2,
   Rect * dstRect
);
```

Parameters

`src1`

The first of two rectangles to enclose.

`src2`

The second of two rectangles to enclose.

`dstRect`

On return, a pointer to the smallest rectangle that encloses both of the rectangles you specify in the `src1` and `src2` parameters. One of the source rectangles may also be the destination.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in OS X v10.0 through OS X v10.6.

Related Sample Code

MovieBrowser

QTMusicToo

Declared in

`QuickdrawAPI.h`

UnionRgn

Calculates the union of two regions. (Available in OS X v10.0 through OS X v10.6.)

```
void UnionRgn (
   RgnHandle srcRgnA,
   RgnHandle srcRgnB,
   RgnHandle dstRgn
);
```

Parameters

`srcRgnA`

A handle to the first of two regions whose union is to be determined.

`srcRgnB`

A handle to the second of two regions whose union is to be determined.

`dstRgn`

On return, a handle to the region holding the resulting union area. If both regions are empty, `UnionRgn` sets the destination to the empty region defined by the rectangle (0,0,0,0).

The `UnionRgn` function does not create the destination region; you must have already allocated memory for it by using the `NewRgn` (page 336) function.

The destination region may be one of the source regions, if desired.

Discussion

The UnionRgn procedure calculates the union of the two regions whose handles you pass in the srcRgnA and srcRgnB parameters, and it places the union in the region whose handle you pass in the dstRgn parameter. If both regions are empty, UnionRgn sets the destination to the empty region defined by the rectangle (0,0,0,0).

Special Considerations

The `UnionRgn` function may temporarily use heap space that's twice the size of the two input regions.

The `UnionRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code

Inside Mac Movie TB Code

MovieBrowser

QTMusicToo

Declared in

`QuickdrawAPI.h`

UnlockPixels

Allows the Memory Manager to move the base address for the offscreen pixel map that you specify in the `pm` parameter. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void UnlockPixels (
   PixMapHandle pm
);
```


Parameters

`pm`

A handle to an offscreen pixel map. Pass the same handle that you passed previously to the `LockPixels` function.

Discussion

To ensure the integrity of the data in a pixel image, call `LockPixels` before drawing into or copying from a pixel map; then, to prevent heap fragmentation, call `UnlockPixels` as soon as your application finishes drawing to and copying from the offscreen pixel map.

The `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer (which is what the `baseAddr` field for an onscreen pixel map contains). The `LockPixels` function dereferences the `PixMap` handle into a pointer. When you use the `UnlockPixels` function, the handle is recovered.

You don't need to call `UnlockPixels` if `LockPixels` returns `FALSE`, because `LockPixels` doesn't lock the memory for a pixel image if that memory has been purged. However, calling `UnlockPixels` on purged memory does no harm.

Special Considerations

The `UnlockPixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Palette and GWorld

vrmakepano

VRMakePano Library

vrscript

vrscript.win

Declared in

`QDOffscreen.h`

UnlockPortBits

Releases a previously acquired lock on the back buffer for a Carbon window. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
OSErr UnlockPortBits (
   GrafPtr port
);
```

Parameters

`port`

A window port specified in a previous call to `LockPortBits` (page 316).

Return Value

A result code. If `noErr`, the corresponding lock is released.

Discussion

For more information about this function, see `LockPortBits` (page 316).

In Mac OS 9, this function does nothing and returns `noErr`.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared in

`QuickdrawAPI.h`

UnpackBits

Decompresses a data buffer containing data compressed by `PackBits`. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void UnpackBits (
   Ptr *srcPtr,
   Ptr *dstPtr,
   short dstBytes
);
```


Parameters

`srcPtr`

On entry, a pointer to the first byte of a buffer of data to be decompressed. On exit, a pointer to the first byte following the compressed data.

`dstPtr`

On entry, a pointer to the first byte in which to store decompressed data. On exit, a pointer to the first byte following the decompressed data.

`dstBytes`

The number of bytes of the data before compression. Use `PackBits` to compress data structures of a fixed size that you can then pass in this parameter to `UnpackBits`, or store with the compressed data the original size of the uncompressed data.

Discussion

Because your application must allocate memory for the source and destination buffers, `UnpackBits` does not move relocatable blocks. Thus, you can call it at interrupt time.

Because `UnpackBits` changes the values of the `srcPtr` and `dstPtr` parameters, you should pass to `UnpackBits` only copies of the pointers to the source and destination buffers. This allows you to access the beginning of the source and destination buffers after `UnpackBits` returns. Also, if the source or destination buffer is stored in an unlocked, relocatable block, this technique prevents `UnpackBits` from changing the value of a master pointer, which would make the original handle invalid.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

UpdateGWorld

Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world. (Available in OS X v10.0 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
GWorldFlags UpdateGWorld (
   GWorldPtr *offscreenGWorld,
```



```
   short pixelDepth,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

`offscreenGWorld`

On input, a pointer to an existing offscreen graphics world; upon completion, the pointer to the updated offscreen graphics world.

`pixelDepth`

The pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. If you specify 0 in this parameter, `UpdateGWorld` rescans the device list and uses the depth of the screen with the greatest pixel depth among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify 0 in this parameter, `UpdateGWorld` also copies the `GDevice` structure from this device to create an offscreen `GDevice` structure. The `UpdateGWorld` function ignores the value you supply for this parameter if you specify a `GDevice` structure in the `aGDevice` parameter.

`boundsRect`

The boundary rectangle and port rectangle for the offscreen pixel map. This also becomes the boundary rectangle for the `GDevice` structure, if `NewGWorld` creates one. If you specify 0 in the `pixelDepth` parameter, `NewGWorld` interprets the boundaries in global coordinates, with which it determines which screens intersect the rectangle. (`NewGWorld` then uses the pixel depth, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle.) Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

`cTable`

A handle to a `ColorTable` structure. If you pass `NULL` in this parameter, `UpdateGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter; if you set the `pixelDepth` parameter to 0, `UpdateGWorld` copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. The `UpdateGWorld` function ignores the value you supply for this parameter if you specify a `GDevice` structure in the `aGDevice` parameter.

`aGDevice`

As an option, a handle to a `GDevice` structure whose pixel depth and color table you want to use for the offscreen graphics world. To use the pixel depth and color table that you specify in the `pixelDepth` and `cTable` parameters, set this parameter to `NULL`.

`flags`

Options available to your application. You can set a combination of the flags `clipPix`, `stretchPix`, and `ditherPix`. If you don't wish to use any of these flags, specify 0. However, you should pass either `clipPix` or `stretchPix` to ensure that the pixel map is updated to reflect the new color table. See GWorldFlags (page 96) for a description of the values you can use here.

Return Value

`UpdateGWorld` returns the `gwFlagErr` flag if `UpdateGWorld` was unsuccessful; in this case, the offscreen graphics world is left unchanged. Use the `QDError` function to help you determine why `UpdateGWorld` failed.

Discussion

You should call `UpdateGWorld` after every update event and whenever your windows move or change size.

If the `LockPixels` (page 314) function reports that the Memory Manager has purged the base address for the offscreen pixel image, use `UpdateGWorld` to reallocate its memory. Then, reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.

The `UpdateGWorld` function uses the following algorithm when updating the offscreen pixel image:

1. If the color table that you specify in the `cTable` parameter is different from the previous color table, or if the color table associated with the `GDevice` structure that you specify in the `aGDevice` parameter is different, Color QuickDraw maps the pixel values in the offscreen pixel map to the new color table.
2. If the value you specify in the `pixelDepth` parameter differs from the previous pixel depth, Color QuickDraw translates the pixel values in the offscreen pixel image to those for the new pixel depth.
3. If the rectangle you specify in the `boundsRect` parameter differs from, but has the same size as, the previous boundary rectangle, QuickDraw realigns the pixel image to the screen for optimum performance for the `CopyBits` function.
4. If the rectangle you specify in the `boundsRect` parameter is smaller than the previous boundary rectangle and you specify the `clipPix` flag, the pixel image is clipped along the bottom and right edges.
5. If the rectangle you specify in the `boundsRect` parameter is bigger than the previous boundary rectangle and you specify the `clipPix` flag, the bottom and right edges of the pixel image are undefined.
6. If the rectangle you specify in the `boundsRect` parameter is smaller than the previous boundary rectangle and you specify the `stretchPix` flag, the pixel image is reduced to the new size.
7. If the rectangle you specify in the `boundsRect` parameter is bigger than the previous boundary rectangle and you specify the `stretchPix` flag, the pixel image is stretched to the new size.
8. If the Memory Manager purged the base address for the offscreen pixel image, `UpdateGWorld` reallocates the memory, but the pixel image is lost. You must reconstruct it.

Special Considerations

The `UpdateGWorld` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in OS X v10.0 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

StarMenu

TE Over Background

Declared in

`QDOffscreen.h`

XorRgn

Calculates the difference between the union and the intersection of two regions. (Available in OS X v10.0 through OS X v10.6.)

```
void XorRgn (
   RgnHandle srcRgnA,
   RgnHandle srcRgnB,
   RgnHandle dstRgn
);
```

Parameters

`srcRgnA`

A handle to the first of two regions to compare.

`srcRgnB`

A handle to the second of two regions to compare.

`dstRgn`

On return, a handle to the region holding the result.

This does not create the destination region; you must have already allocated memory for it by using the `NewRgn` (page 336) function.

If the regions are coincident, `XorRgn` sets the destination region to the empty region defined by the rectangle (0,0,0,0).

Discussion
The XorRgn procedure calculates the difference between the union and the intersection of the regions whose handles you pass in the srcRgnA and srcRgnB parameters and places the result in the region whose handle you pass in the dstRgn parameter.

Special Considerations
The `XorRgn` function may temporarily use heap space that's twice the size of the two input regions.

The `XorRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in OS X v10.0 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

Available in OS X v10.1 through OS X v10.6

IsPortClipRegionEmpty

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean IsPortClipRegionEmpty (
   CGrafPtr port
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

IsPortVisibleRegionEmpty

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean IsPortVisibleRegionEmpty (
   CGrafPtr port
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDAddRectToDirtyRegion

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDAddRectToDirtyRegion (
   CGrafPtr inPort,
   const Rect *inBounds
);
```

Return Value

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDAddRegionToDirtyRegion

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDAddRegionToDirtyRegion (
   CGrafPtr inPort,
   RgnHandle inRegion
);
```

Return Value

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDDisposeRegionBits

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDDisposeRegionBits (
   QDRegionBitsRef regionBits
);
```

Return Value

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDGetPatternOrigin

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void QDGetPatternOrigin (
   Point *origin
);
```

Availability

Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in

`QuickdrawAPI.h`

QDPictCreateWithProvider

Creates a QDPict picture, using QuickDraw picture data supplied with a Quartz data provider. (Available in OS X v10.1 through OS X v10.6.)

```
QDPictRef QDPictCreateWithProvider (
   CGDataProviderRef provider
);
```

Parameters

`provider`

A Quartz data provider that supplies QuickDraw picture data. The picture data must begin at either the first byte or the 513th byte in the data provider. The picture bounds must not be an empty rectangle.

QuickDraw retains the data provider you pass in, and you may safely release it after this function returns.

Return Value

A new QDPict picture, or `NULL` if the picture data is not valid. The initial retain count is 1. After you finish using the picture, you should release it by calling `QDPictRelease` (page 465).

Discussion

This function creates a QDPict picture that you can draw in a Quartz context. For general information about QDPict pictures, see `QDPictRef` (page 113).

Availability
Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QDPictToCGContext.h`

QDPictCreateWithURL

Creates a QDPict picture, using QuickDraw picture data specified with a Core Foundation URL. (Available in OS X v10.1 through OS X v10.6.)

```
QDPictRef QDPictCreateWithURL (
   CFURLRef url
);
```

Parameters
`url`

A Core Foundation URL that specifies a PICT file containing the QuickDraw picture data. The picture header data must begin at either the first byte or the 513th byte in the PICT file. The picture bounds must not be an empty rectangle.

Return Value
A new QDPict picture, or `NULL` if the picture data is not valid. The initial retain count is 1. After you finish using the picture, you should release it by calling `QDPictRelease` (page 465).

Discussion
This function creates a QDPict picture that you can draw in a Quartz context. For general information about QDPict pictures, see `QDPictRef` (page 113).

Availability
Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CGDrawPicture

Declared in
`QDPictToCGContext.h`

QDPictDrawToCGContext

Draws a QuickDraw picture in a Quartz context. (Available in OS X v10.1 through OS X v10.6.)

```
OSStatus QDPictDrawToCGContext (
   CGContextRef ctx,
   CGRect rect,
   QDPictRef pictRef
);
```

Parameters

`context`

The Quartz context in which to draw.

`rect`

The rectangular area in which to draw the picture. You should specify the origin and size of this rectangle in user space units. The origin is the lower left corner of the picture when drawn. If necessary, the picture is scaled to fit inside this rectangle. To get unscaled results, you should pass the rectangle returned by `QDPictGetBounds` (page 464). For additional information about scaling, see the discussion below.

`picture`

A QDPict picture.

Return Value

A result code. A non-zero result indicates that the picture was not successfully drawn.

Discussion

This function converts the picture data in a QDPict picture into an equivalent sequence of Quartz 2D graphics operations. Conceptually this is the same processing path taken when an application running in Mac OS X draws into a QuickDraw printing port.

When drawing a QDPict picture in a Quartz context, there are two ways to change the horizontal or vertical scale of the picture:

- Construct the drawing rectangle (see the `rect` parameter) by applying the change of scale to the bounds rectangle returned by `QDPictGetBounds` (page 464). In this case, QuickDraw scales all the graphic elements in the picture except for patterns—the same behavior as `DrawPicture` (page 192).
- Prior to calling `QDPictDrawToCGContext`, apply the change of scale to the current transformation matrix in the Quartz context—for example, by calling `CGContextScaleCTM`. In this case, QuickDraw scales the entire picture including patterns.

In a bitmap-based context, the picture is rendered into the bitmap. In a PDF-based context, the picture is converted into a PDF content stream. If the picture uses transfer modes such as `srcXor` that do not have an analog in Quartz 2D, the PDF representation may not match the original exactly.

Availability
Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CGDrawPicture

Declared in
`QDPictToCGContext.h`

QDPictGetBounds

Returns the intended location and size of a QDPict picture. (Available in OS X v10.1 through OS X v10.6.)

```
CGRect QDPictGetBounds (
   QDPictRef pictRef
);
```

Parameters
`picture`
A QDPict picture.

Return Value
A Quartz rectangle that represents the intended location and size of the picture. The rectangle is in default user space with one unit = 1/72 inch, and the origin is the lower-left corner of the picture.

Discussion
If the native resolution in the picture data is not 72 pixels per inch, the bounding rectangle returned by this function is scaled as follows:

```
width = width in pixels * 72 / horizontal resolution
height = height in pixels * 72 / vertical resolution
```

Availability
Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CGDrawPicture

Declared in
`QDPictToCGContext.h`

QDPictGetResolution

Returns the horizontal and vertical resolution of a QDPict picture. (Available in OS X v10.1 through OS X v10.6.)

```
void QDPictGetResolution (
   QDPictRef pictRef,
   float *xRes,
   float *yRes
);
```

Parameters

`picture`

A QDPict picture.

`xRes`

A pointer to your storage for a return value. Upon completion, the value is the picture's horizontal resolution in pixels per inch.

`yRes`

A pointer to your storage for a return value. Upon completion, the value is the picture's vertical resolution in pixels per inch.

Discussion

This function returns resolution data that you can use—together with the rectangle returned by `QDPictGetBounds` (page 464)—to compute the picture's size in pixels.

Availability

Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Declared in

`QDPictToCGContext.h`

QDPictRelease

Releases a QDPict picture. (Available in OS X v10.1 through OS X v10.6.)


```
void QDPictRelease (
   QDPictRef pictRef
);
```

Parameters

`picture`

A QDPict picture which you created or retained.

Discussion

After you finish using a QDPict picture that you created or retained, you should call this function to release the picture. If the picture's retain count becomes 0, this function frees the picture and any associated resources such as the picture's data provider.

Availability

Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Related Sample Code
CGDrawPicture

Declared in

`QDPictToCGContext.h`

QDPictRetain

Retains a QDPict picture. (Available in OS X v10.1 through OS X v10.6.)

```
QDPictRef QDPictRetain (
   QDPictRef pictRef
);
```

Parameters

`picture`

A QDPict picture.

Return Value

The retained picture.

Discussion
You should call this function when you obtain a QDPict picture that you did not create and you want to retain the picture for later use. When you no longer need the retained picture, you should call `QDPictRelease` (page 465) to release it.

Availability
Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QDPictToCGContext.h`

QDRestoreRegionBits

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDRestoreRegionBits (
   RgnHandle region,
   QDRegionBitsRef regionBits
);
```

Return Value

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSaveRegionBits

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
QDRegionBitsRef QDSaveRegionBits (
```



```
   RgnHandle region
);
```

Return Value

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSetPatternOrigin

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void QDSetPatternOrigin (
   Point origin
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSwapPort

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean QDSwapPort (
   CGrafPtr inNewPort,
```



```
   CGrafPtr *outOldPort
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CustomWindowWidget
ElectricImageComponent.win
QTCarbonShell
StarMenu

Declared in
`QuickdrawAPI.h`

RgnToHandle

(Available in OS X v10.1 through OS X v10.6.)

```
void RgnToHandle (
   RgnHandle region,
   Handle flattenedRgnDataHdl
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SectRegionWithPortClipRegion

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SectRegionWithPortClipRegion (
   CGrafPtr port,
```



```
   RgnHandle ioRegion
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SectRegionWithPortVisibleRegion

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SectRegionWithPortVisibleRegion (
   CGrafPtr port,
   RgnHandle ioRegion
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortTextFace

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
void SetPortTextFace (
   CGrafPtr port,
   StyleParameter face
);
```


Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortTextFont

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetPortTextFont (
   CGrafPtr port,
   short txFont
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortTextMode

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetPortTextMode (
   CGrafPtr port,
   short mode
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SetPortTextSize

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void SetPortTextSize (
   CGrafPtr port,
   short txSize
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SwapPortPicSaveHandle

(Available in OS X v10.1 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Handle SwapPortPicSaveHandle (
   CGrafPtr port,
   Handle inPicSaveHdl
);
```

Availability
Available in OS X v10.1 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

Available in OS X v10.2 through OS X v10.6

QDGlobalToLocalPoint

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Point * QDGlobalToLocalPoint (
   CGrafPtr port,
   Point *point
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CalendarView
CarbonSketch
Custom_HIView_Tutorial
MouseTracking

Declared in
`QuickdrawAPI.h`

QDGlobalToLocalRect

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Rect * QDGlobalToLocalRect (
   CGrafPtr port,
```



```
   Rect *bounds
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDGlobalToLocalRegion

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
RgnHandle QDGlobalToLocalRegion (
   CGrafPtr port,
   RgnHandle region
);
```

Return Value

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDIsNamedPixMapCursorRegistered

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Boolean QDIsNamedPixMapCursorRegistered (
```



```
   const char name[128]
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDLocalToGlobalPoint

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Point * QDLocalToGlobalPoint (
   CGrafPtr port,
   Point *point
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDLocalToGlobalRect

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Rect * QDLocalToGlobalRect (
   CGrafPtr port,
   Rect *bounds
);
```


Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared in
`QuickdrawAPI.h`

QDLocalToGlobalRegion

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
RgnHandle QDLocalToGlobalRegion (
   CGrafPtr port,
   RgnHandle region
);
```

Return Value

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDRegisterNamedPixMapCursor

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDRegisterNamedPixMapCursor (
   PixMapHandle crsrData,
   PixMapHandle crsrMask,
```



```
   Point hotSpot,
   const char name[128]
);
```

Return Value

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSetCursorScale

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDSetCursorScale (
   float scale
);
```

Return Value

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSetNamedPixMapCursor

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
OSStatus QDSetNamedPixMapCursor (
```



```
   const char name[128]
);
```

Return Value

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSwapPortTextFlags

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
UInt32 QDSwapPortTextFlags (
   CGrafPtr port,
   UInt32 newFlags
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDSwapTextFlags

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
UInt32 QDSwapTextFlags (
```



```
   UInt32 newFlags
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SwapPortPolySaveHandle

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Handle SwapPortPolySaveHandle (
   CGrafPtr port,
   Handle inPolySaveHdl
);
```

Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

SwapPortRegionSaveHandle

(Available in OS X v10.2 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Handle SwapPortRegionSaveHandle (
   CGrafPtr port,
   Handle inRegionSaveHdl
);
```


Availability
Available in OS X v10.2 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

Available in OS X v10.3 through OS X v10.6

QDGetCGDirectDisplayID

Returns the Quartz display ID that corresponds to a QuickDraw graphics device. (Available in OS X v10.3 through OS X v10.6.)

```
CGDirectDisplayID QDGetCGDirectDisplayID (
   GDHandle inGDevice
);
```

Parameters
`inGDevice`
A QuickDraw graphics device.

Return Value
A Quartz display ID, or `NULL` if the `inGDevice` parameter does not represent a display. For information about using a display ID, see *Quartz Display Services Reference*.

Availability
Available in OS X v10.3 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDGetCursorData

(Available in OS X v10.3 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
OSStatus QDGetCursorData (
   Boolean contextCursor,
   PixMapHandle *crsrData,
   Point *hotSpot
);
```

Return Value

Availability
Available in OS X v10.3 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDGetPictureBounds

(Available in OS X v10.3 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)

```
Rect * QDGetPictureBounds (
   PicHandle picH,
   Rect *outRect
);
```

Availability
Available in OS X v10.3 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

QDUnregisterNamedPixMapCursor

(Available in OS X v10.3 through OS X v10.6. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
OSStatus QDUnregisterNamedPixMapCursor (
   const char name[128]
);
```

Return Value

Availability
Available in OS X v10.3 through OS X v10.6.

Deprecated in OS X v10.4.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

Available in OS X v10.4 through OS X v10.6

IsValidRgnHandle

(Available in OS X v10.4 through OS X v10.6.)

```
Boolean IsValidRgnHandle (
   RgnHandle rgn
);
```

Availability
Available in OS X v10.4 through OS X v10.6.

Not available to 64-bit applications.

Declared in
`QuickdrawAPI.h`

Deprecated in OS X v10.4

DisposeColorComplementUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers .)


```
void DisposeColorComplementUPP (
   ColorComplementUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeColorSearchUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeColorSearchUPP (
   ColorSearchUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeDragGrayRgnUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeDragGrayRgnUPP (
   DragGrayRgnUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDArcUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDArcUPP (
   QDArcUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDBitsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDBitsUPP (
   QDBitsUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Related Sample Code
DesktopSprites
qteffects.win
qtspritesplus
qtwiredactions
qtwiredspritesjr.win

Declared in
`Quickdraw.h`

DisposeQDCommentUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDCommentUPP (
   QDCommentUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDGetPicUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDGetPicUPP (
   QDGetPicUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDJShieldCursorUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDJShieldCursorUPP (
   QDJShieldCursorUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDLineUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDLineUPP (
   QDLineUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDOpcodeUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDOpcodeUPP (
   QDOpcodeUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDOvalUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
void DisposeQDOvalUPP (
   QDOvalUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDPolyUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDPolyUPP (
   QDPolyUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDPutPicUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDPutPicUPP (
   QDPutPicUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDRectUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDRectUPP (
   QDRectUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDRgnUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDRgnUPP (
   QDRgnUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDRRectUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDRRectUPP (
```



```
   QDRRectUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDStdGlyphsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDStdGlyphsUPP (
   QDStdGlyphsUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDTextUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDTextUPP (
   QDTextUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeQDTxMeasUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeQDTxMeasUPP (
   QDTxMeasUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

DisposeRegionToRectsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void DisposeRegionToRectsUPP (
   RegionToRectsUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeColorComplementUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean InvokeColorComplementUPP (
   RGBColor *rgb,
   ColorComplementUPP userUPP
);
```


Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeColorSearchUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
Boolean InvokeColorSearchUPP (
   RGBColor *rgb,
   long *position,
   ColorSearchUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeDragGrayRgnUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeDragGrayRgnUPP (
   DragGrayRgnUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDArcUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDArcUPP (
   GrafVerb verb,
   const Rect *r,
   short startAngle,
   short arcAngle,
   QDArcUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDBitsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDBitsUPP (
   const BitMap *srcBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn,
   QDBitsUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDCommentUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDCommentUPP (
   short kind,
   short dataSize,
   Handle dataHandle,
   QDCommentUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDGetPicUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDGetPicUPP (
   void *dataPtr,
   short byteCount,
   QDGetPicUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDJShieldCursorUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDJShieldCursorUPP (
   short left,
```



```
   short top,
   short right,
   short bottom,
   QDJShieldCursorUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDLineUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDLineUPP (
   Point newPt,
   QDLineUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDOpcodeUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDOpcodeUPP (
   const Rect *fromRect,
   const Rect *toRect,
   UInt16 opcode,
   SInt16 version,
   QDOpcodeUPP userUPP
);
```


Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDOvalUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDOvalUPP (
   GrafVerb verb,
   const Rect *r,
   QDOvalUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDPolyUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDPolyUPP (
   GrafVerb verb,
   PolyHandle poly,
   QDPolyUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDPutPicUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDPutPicUPP (
   const void *dataPtr,
   short byteCount,
   QDPutPicUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDRectUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDRectUPP (
   GrafVerb verb,
   const Rect *r,
   QDRectUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDRgnUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDRgnUPP (
   GrafVerb verb,
```



```
   RgnHandle rgn,
   QDRgnUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDRRectUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDRRectUPP (
   GrafVerb verb,
   const Rect *r,
   short ovalWidth,
   short ovalHeight,
   QDRRectUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDStdGlyphsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
OSStatus InvokeQDStdGlyphsUPP (
   void *dataStream,
   ByteCount size,
   QDStdGlyphsUPP userUPP
);
```


Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDTextUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
void InvokeQDTextUPP (
   short byteCount,
   const void *textBuf,
   Point numer,
   Point denom,
   QDTextUPP userUPP
);
```

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeQDTxMeasUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
short InvokeQDTxMeasUPP (
   short byteCount,
   const void *textAddr,
   Point *numer,
   Point *denom,
   FontInfo *info,
   QDTxMeasUPP userUPP
);
```


Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

InvokeRegionToRectsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
OSStatus InvokeRegionToRectsUPP (
   UInt16 message,
   RgnHandle rgn,
   const Rect *rect,
   void *refCon,
   RegionToRectsUPP userUPP
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewColorComplementUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
ColorComplementUPP NewColorComplementUPP (
   ColorComplementProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewColorSearchUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
ColorSearchUPP NewColorSearchUPP (
   ColorSearchProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewDragGrayRgnUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
DragGrayRgnUPP NewDragGrayRgnUPP (
   DragGrayRgnProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDArcUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDArcUPP NewQDArcUPP (
   QDArcProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDBitsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDBitsUPP NewQDBitsUPP (
   QDBitsProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Related Sample Code
DesktopSprites

qteffects.win

qtspritesplus

qtwiredactions

qtwiredspritesjr.win

Declared in
`Quickdraw.h`

NewQDCommentUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDCommentUPP NewQDCommentUPP (
   QDCommentProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDGetPicUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDGetPicUPP NewQDGetPicUPP (
   QDGetPicProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDJShieldCursorUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDJShieldCursorUPP NewQDJShieldCursorUPP (
```



```
   QDJShieldCursorProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDLineUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDLineUPP NewQDLineUPP (
   QDLineProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDOpcodeUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDOpcodeUPP NewQDOpcodeUPP (
   QDOpcodeProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDOvalUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDOvalUPP NewQDOvalUPP (
   QDOvalProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDPolyUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDPolyUPP NewQDPolyUPP (
   QDPolyProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDPutPicUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDPutPicUPP NewQDPutPicUPP (
   QDPutPicProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDRectUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDRectUPP NewQDRectUPP (
   QDRectProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDRgnUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDRgnUPP NewQDRgnUPP (
   QDRgnProcPtr userRoutine
);
```


Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDRRectUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDRRectUPP NewQDRRectUPP (
   QDRRectProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDStdGlyphsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDStdGlyphsUPP NewQDStdGlyphsUPP (
   QDStdGlyphsProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDTextUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDTextUPP NewQDTextUPP (
   QDTextProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewQDTxMeasUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)

```
QDTxMeasUPP NewQDTxMeasUPP (
   QDTxMeasProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

NewRegionToRectsUPP

(Deprecated in OS X v10.4. Use Quartz 2D instead; see Quartz Programming Guide for QuickDraw Developers.)


```
RegionToRectsUPP NewRegionToRectsUPP (
   RegionToRectsProcPtr userRoutine
);
```

Return Value

Availability
Available in OS X v10.0 and later.

Deprecated in OS X v10.4.

Declared in
`Quickdraw.h`

Document Revision History

This table describes the changes to *QuickDraw Reference*.

Date	Notes
2007-06-29	Made minor format and editorial changes. Added `kNativeEndianPixMap` to the list of constants in "`Graphics World Flags`" (page 129).
2006-07-24	Added information about deprecated functions. Documented the QDGetCGDirectDisplayID function.
2005-08-11	Updated description of the function InitCursor.
2004-06-28	Added documentation for the functions and data types in `QDPictToCGContext.h`.
2004-02-26	Added or changed the documentation for the following functions: `QDBeginCGContext` (page 59), `QDEndCGContext` (page 61), `ClipCGContextToRegion` (page 158), `SyncCGContextOriginWithPort` (page 447), `CreateCGContextForPort` (page 175), `LockPortBits` (page 316), `UnlockPortBits` (page 453), `QDFlushPortBuffer` (page 378), `CreateNewPortForCGDisplayID` (page 177), `GetIndPattern` (page 239), `DeltaPoint` (page 181), `SubPt` (page 446).
2003-02-01	Updated to include information about Mac OS X availability.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010