
NSGraphicsContext Class Reference

Graphics & Animation





Apple Inc.
© 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Cocoa, iChat, Mac, Mac OS, and Quartz are trademarks of Apple Inc., registered in the United States and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

NSGraphicsContext Class Reference 5

Overview	5
Tasks	6
Creating a Graphics Context	6
Managing the Current Context	6
Managing the Graphics State	6
Testing the Drawing Destination	6
Getting Information About a Context	7
Flushing Graphics to the Context	7
Managing the Focus Stack	7
Configuring Rendering Options	7
Getting the Core Image Context	8
Managing the Color Rendering Intent	8
Class Methods	8
currentContext	8
currentContextDrawingToScreen	8
graphicsContextWithAttributes:	9
graphicsContextWithBitmapImageRep:	9
graphicsContextWithGraphicsPort:flipped:	10
graphicsContextWithWindow:	10
restoreGraphicsState	11
saveGraphicsState	11
setCurrentContext:	12
setGraphicsState:	12
Instance Methods	13
attributes	13
CIContext	13
colorRenderingIntent	14
compositingOperation	14
flushGraphics	15
focusStack	15
graphicsPort	15
imageInterpolation	16
isDrawingToScreen	16
isFlipped	16
patternPhase	17
restoreGraphicsState	17
saveGraphicsState	18
setColorRenderingIntent:	18
setCompositingOperation:	19
setFocusStack:	19

- setImageInterpolation: 20
- setPatternPhase: 20
- setShouldAntialias: 21
- shouldAntialias 21
- Constants 22
 - Attribute dictionary keys 22
 - Representation format attribute keys 22
 - NSImageInterpolation 22
 - NSColorRenderingIntent 23

Document Revision History 25

Index 27

NSGraphicsContext Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AppKit.framework
Availability	Available in Mac OS X v10.0 and later.
Companion guide	Cocoa Drawing Guide
Declared in	NSGraphics.h NSGraphicsContext.h
Related sample code	FunHouse Quartz EB QuickLookSketch Reducer Sketch-112

Overview

The `NSGraphicsContext` class is the programmatic interface to objects that represent graphics contexts. A context can be thought of as a destination to which drawing and graphics state operations are sent for execution. Each graphics context contains its own graphics environment and state.

The `NSGraphicsContext` class is an abstract superclass for destination-specific graphics contexts. You obtain instances of concrete subclasses with the class methods [currentContext](#) (page 8), [graphicsContextWithAttributes:](#) (page 9), [graphicsContextWithBitmapImageRep:](#) (page 9), [graphicsContextWithGraphicsPort:flipped:](#) (page 10), and [graphicsContextWithWindow:](#) (page 10).

At any time there is the notion of the current context. The current context for the current thread may be set using [setCurrentContext:](#) (page 12).

Graphics contexts are maintained on a stack. You push a graphics context onto the stack by sending it a [saveGraphicsState](#) (page 18) message, and pop it off the stack by sending it a [restoreGraphicsState](#) (page 17) message. By sending [restoreGraphicsState](#) (page 17) to an `NSGraphicsContext` object you remove it from the stack, and the next graphics context on the stack becomes the current graphics context.

Tasks

Creating a Graphics Context

- + [graphicsContextWithAttributes:](#) (page 9)
Instantiates and returns an instance of `NSGraphicsContext` using the specified attributes.
- + [graphicsContextWithBitmapImageRep:](#) (page 9)
Instantiates and returns a new graphics context using the supplied `NSBitmapImageRep` object as the context destination.
- + [graphicsContextWithGraphicsPort:flipped:](#) (page 10)
Instantiates and returns a new graphics context from the given graphics port.
- + [graphicsContextWithWindow:](#) (page 10)
Creates and returns a new graphics context for drawing into a window.

Managing the Current Context

- + [currentContext](#) (page 8)
Returns the current graphics context of the current thread.
- + [setCurrentContext:](#) (page 12)
Sets the current graphics context of the current thread.
- [graphicsPort](#) (page 15)
Returns the low-level, platform-specific graphics context represented by the receiver.

Managing the Graphics State

- + [setGraphicsState:](#) (page 12)
Makes the graphics context of the specified graphics state current, and resets graphics state.
- + [restoreGraphicsState](#) (page 11)
Pops a graphics context from the per-thread stack, makes it current, and sends the context a [restoreGraphicsState](#) (page 17) message.
- [restoreGraphicsState](#) (page 17)
Removes the receiver's graphics state from the top of the graphics state stack and makes the next graphics state the current graphics state.
- + [saveGraphicsState](#) (page 11)
Saves the graphics state of the current graphics context.
- [saveGraphicsState](#) (page 18)
Saves the current graphics state and creates a new graphics state on the top of the stack.

Testing the Drawing Destination

- + [currentContextDrawingToScreen](#) (page 8)
Returns a Boolean value that indicates whether the current context is drawing to the screen.

- [isDrawingToScreen](#) (page 16)
Returns a Boolean value that indicates whether the drawing destination is the screen.

Getting Information About a Context

- [attributes](#) (page 13)
Returns the receiver's attributes.
- [isFlipped](#) (page 16)
Returns a Boolean value that indicates the receiver's flipped state.

Flushing Graphics to the Context

- [flushGraphics](#) (page 15)
Forces any buffered operations or data to be sent to the receiver's destination.

Managing the Focus Stack

- [focusStack](#) (page 15)
Returns the object used by the context to track the hierarchy of views with locked focus.
- [setFocusStack:](#) (page 19)
Sets the object used by the receiver to track the hierarchy of views with locked focus.

Configuring Rendering Options

- [setCompositingOperation:](#) (page 19)
Sets the receiver's global compositing operation.
- [compositingOperation](#) (page 14)
Returns the receiver's global compositing operation setting.
- [setImageInterpolation:](#) (page 20)
Sets the receiver's interpolation behavior.
- [imageInterpolation](#) (page 16)
Returns a constant that specifies the receiver's interpolation behavior.
- [setShouldAntialias:](#) (page 21)
Sets whether the receiver should use antialiasing.
- [shouldAntialias](#) (page 21)
Returns a Boolean value that indicates whether the receiver uses antialiasing.
- [setPatternPhase:](#) (page 20)
Sets the amount to offset the pattern color when filling the receiver.
- [patternPhase](#) (page 17)
Returns the amount to offset the pattern color when filling the receiver.

Getting the Core Image Context

- [CIContext](#) (page 13)

Returns a `CIContext` object that you can use to render into the receiver.

Managing the Color Rendering Intent

- [colorRenderingIntent](#) (page 14)

Returns the current rendering intent in the receiver's graphics state.

- [setColorRenderingIntent:](#) (page 18)

Sets the rendering intent in the receiver's graphics state.

Class Methods

currentContext

Returns the current graphics context of the current thread.

```
+ (NSGraphicsContext *)currentContext
```

Return Value

The current graphics context of the current thread.

Discussion

Returns an instance of a concrete subclass of `NSGraphicsContext`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

FunHouse

Quartz EB

QuickLookSketch

Sketch+Accessibility

Sketch-112

Declared In

`NSGraphicsContext.h`

currentContextDrawingToScreen

Returns a Boolean value that indicates whether the current context is drawing to the screen.

```
+ (BOOL)currentContextDrawingToScreen
```

Return Value

YES if the current context is drawing to the screen, otherwise NO.

Discussion

This convenience method is equivalent to sending `isDrawingToScreen` (page 16) to the result of `currentContext` (page 8).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSGraphicsContext.h`

graphicsContextWithAttributes:

Instantiates and returns an instance of `NSGraphicsContext` using the specified attributes.

```
+ (NSGraphicsContext *)graphicsContextWithAttributes:(NSDictionary *)attributes
```

Parameters

attributes

A dictionary of values associated with the keys described in “Attribute dictionary keys” (page 22). The attributes specify such things as representation format and destination.

Return Value

A new `NSGraphicsContext` object or `nil` if the object could not be created.

Discussion

Use this method to create a graphics context for a window or bitmap destination. If you want to create a graphics context for a PDF or PostScript destination, do not use this method; instead, use the `NSPrintOperation` class to set up the printing environment needed to generate that type of information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSGraphicsContext.h`

graphicsContextWithBitmapImageRep:

Instantiates and returns a new graphics context using the supplied `NSBitmapImageRep` object as the context destination.

```
+ (NSGraphicsContext *)graphicsContextWithBitmapImageRep:(NSBitmapImageRep *)bitmapRep
```

Parameters

bitmapRep

The `NSBitmapImageRep` object to use as the destination.

Return Value

The created `NSGraphicsContext` object or `nil` if the object could not be created.

Discussion

This method accepts only single plane `NSBitmapImageRep` instances. It is the equivalent of using `graphicsContextWithAttributes:` (page 9) and passing *bitmapRep* as the value for the dictionary’s `NSGraphicsContextDestinationAttributeName` key.

Availability

Available in Mac OS X v10.4 and later.

See Also

+ [graphicsContextWithAttributes:](#) (page 9)

Related Sample Code

AnimatedTableView

Reducer

Declared In

NSGraphicsContext.h

graphicsContextWithGraphicsPort:flipped:

Instantiates and returns a new graphics context from the given graphics port.

```
+ (NSGraphicsContext *)graphicsContextWithGraphicsPort:(void *)graphicsPort
  flipped:(BOOL)initialFlippedState
```

Parameters

graphicsPort

The graphics port used to create the graphics-context object. Typically *graphicsPort* is a CGContextRef (opaque type) object.

initialFlippedState

Specifies the receiver's initial flipped state. This is the value returned by [isFlipped](#) (page 16) when no view has focus.

Return Value

The created NSGraphicsContext object or nil if the object could not be created.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

CIAnnotation

CIVideoDemoGL

FunHouse

iChatTheater

QuickLookSketch

Declared In

NSGraphicsContext.h

graphicsContextWithWindow:

Creates and returns a new graphics context for drawing into a window.

```
+ (NSGraphicsContext *)graphicsContextWithWindow:(NSWindow *)aWindow
```

Parameters*aWindow*

The window object representing the window to use for drawing.

Return Value

The created `NSGraphicsContext` object or `nil` if the object could not be created.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

JAWTExample

Declared In

`NSGraphicsContext.h`

restoreGraphicsState

Pops a graphics context from the per-thread stack, makes it current, and sends the context a [restoreGraphicsState](#) (page 17) message.

```
+ (void)restoreGraphicsState
```

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QuickLookSketch

Reducer

Sketch+Accessibility

Sketch-112

TrackBall

Declared In

`NSGraphicsContext.h`

saveGraphicsState

Saves the graphics state of the current graphics context.

```
+ (void)saveGraphicsState
```

Discussion

This method sends the current graphics context a [saveGraphicsState](#) (page 18) message and pushes the context onto the per-thread stack.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QuickLookSketch

Reducer

Sketch+Accessibility

Sketch-112

TrackBall

Declared In

NSGraphicsContext.h

setCurrentContext:

Sets the current graphics context of the current thread.

```
+ (void)setCurrentContext:(NSGraphicsContext *)context
```

Parameters

context

The graphics-context object to set as the current one. This must be an instance of a concrete subclass of NSGraphicsContext.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIAnnotation

CIVideoDemoGL

FunHouse

iChatTheater

QuickLookSketch

Declared In

NSGraphicsContext.h

setGraphicsState:

Makes the graphics context of the specified graphics state current, and resets graphics state.

```
+ (void)setGraphicsState:(NSInteger)graphicsState
```

Discussion

The *graphicsState* identifier must be created in the calling thread.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSGraphicsContext.h

Instance Methods

attributes

Returns the receiver's attributes.

- (NSDictionary *)attributes

Return Value

The receiver's attributes, if any.

Discussion

Screen-based graphics contexts do not store attributes, even if you create them using [graphicsContextWithAttributes:](#) (page 9).

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSGraphicsContext.h

CImageContext

Returns a CImageContext object that you can use to render into the receiver.

- (CIContext *)CIContext

Return Value

A CImageContext object or nil if the object could not be created.

Discussion

The CImageContext object is created on demand and remains in existence for the lifetime of its owning NSGraphicsContext object. A CImageContext object is an evaluation context for rendering a CIImage object through Quartz 2D or OpenGL. You use CImageContext objects in conjunction with CIFilter, CIImage, CIVector, and CIColor objects to take advantage of the built-in Core Image filters when processing images.

For more on CImageContext objects and related Core Image objects, see *Core Image Programming Guide*.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

AnimatedTableView

CIHazeFilterSample

FunHouse

Reducer

Declared In

NSGraphicsContext.h

colorRenderingIntent

Returns the current rendering intent in the receiver's graphics state.

- (NSColorRenderingIntent)colorRenderingIntent

Return Value

An “[Creating a Graphics Context](#)” (page 6) value that specifies the rendering intent currently used by the receiver. For possible values see “[Color Rendering Intent Constants](#)” (page 23).

Discussion

The rendering intent specifies how Cocoa should handle colors that are not located within the gamut of the destination color space of a graphics context.

Availability

Available in Mac OS X v10.5 and later.

See Also

- [setColorRenderingIntent:](#) (page 18)

Declared In

NSGraphicsContext.h

compositingOperation

Returns the receiver's global compositing operation setting.

- (NSCompositingOperation)compositingOperation

Return Value

The receiver's global compositing operation setting. See `NSCompositingOperation` for valid constants.

Discussion

The compositing operation is a global attribute of the graphics context and affects drawing operations that do not take an explicit compositing operation parameter. For methods that do take an explicit compositing operation parameter, the value of that parameter supersedes the global value.

The compositing operations are related to (but different from) the blend mode settings used in Quartz. Only the default compositing operation (`NSCompositeCopy`) is supported for PDF or PostScript content.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setCompositingOperation:](#) (page 19)

Related Sample Code

ImageMap

ImageMapExample

Declared In

NSGraphicsContext.h

flushGraphics

Forces any buffered operations or data to be sent to the receiver's destination.

```
- (void)flushGraphics
```

Discussion

Graphics contexts use buffers to queue pending operations but for efficiency reasons may not always empty those buffers immediately. This method forces the buffers to be emptied.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CocoaAUHost

iChatTheater

Declared In

NSGraphicsContext.h

focusStack

Returns the object used by the context to track the hierarchy of views with locked focus.

```
- (void *)focusStack
```

Return Value

The object used by the context to track the hierarchy of views with locked focus.

Discussion

You should never need to get or modify the focus stack information. The use of focus stacks may be deprecated in a future release.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.5.

Declared In

NSGraphicsContext.h

graphicsPort

Returns the low-level, platform-specific graphics context represented by the receiver.

```
- (void *)graphicsPort
```

Discussion

In Mac OS X, this is the Core Graphics context, a `CGContextRef` object (opaque type).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIAnnotation

FunHouse

ImageApp
MyPhoto
Quartz EB

Declared In

NSGraphicsContext.h

imageInterpolation

Returns a constant that specifies the receiver's interpolation behavior.

- (NSImageInterpolation)imageInterpolation

Return Value

The receiver's interpolation (image smoothing) behavior.

Discussion

The `NSImageInterpolation` constants are described in [NSImageInterpolation](#) (page 22).

Availability

Available in Mac OS X v10.0 and later.

See Also

- [setImageInterpolation:](#) (page 20)

Declared In

NSGraphicsContext.h

isDrawingToScreen

Returns a Boolean value that indicates whether the drawing destination is the screen.

- (BOOL)isDrawingToScreen

Return Value

YES if the drawing destination is the screen, otherwise NO.

Discussion

A return value of NO may mean that the drawing destination is a printer, but the destination may also be a PDF or EPS file. If this method returns NO, you can call [attributes](#) (page 13) to see if additional information is available about the drawing destination.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSGraphicsContext.h

isFlipped

Returns a Boolean value that indicates the receiver's flipped state.

- (BOOL)isFlipped

Return Value

YES if the receiver is flipped, otherwise NO.

Discussion

The state is determined by sending `isFlipped` to the receiver's view that has focus. If no view has focus, returns NO unless the receiver is instantiated using `graphicsContextWithGraphicsPort:flipped:` (page 10) specifying YES as the flipped parameter.

Availability

Available in Mac OS X v10.4 and later.

See Also

+ [graphicsContextWithGraphicsPort:flipped:](#) (page 10)

Declared In

NSGraphicsContext.h

patternPhase

Returns the amount to offset the pattern color when filling the receiver.

- (NSPoint)patternPhase

Return Value

The amount to offset the pattern color when filling the receiver.

Discussion

The pattern phase is a translation (width, height) applied before a pattern is drawn in the current context and is part of the saved graphics state of the context. The default pattern phase is (0,0). Setting the pattern phase has the effect of temporarily changing the pattern matrix of any pattern you decide to draw. For example, setting the pattern phase to (2,3) has the effect of moving the start of pattern cell tiling to the point (2,3) in default user space.

Availability

Available in Mac OS X v10.2 and later.

See Also

- [setPatternPhase:](#) (page 20)

Declared In

NSGraphicsContext.h

restoreGraphicsState

Removes the receiver's graphics state from the top of the graphics state stack and makes the next graphics state the current graphics state.

- (void)restoreGraphicsState

Discussion

This method must have been preceded with a [saveGraphicsState](#) (page 18) message to add the graphics state to the stack. Invocations of `saveGraphicsState` and `restoreGraphicsState` methods may be nested.

Restoring the graphics state restores such attributes as the current drawing style, transformation matrix, color, and font of the original graphics state.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSGraphicsContext.h

saveGraphicsState

Saves the current graphics state and creates a new graphics state on the top of the stack.

```
- (void)saveGraphicsState
```

Discussion

The new graphics state is a copy of the previous state that can be modified to handle new drawing operations.

Saving the graphics state saves such attributes as the current drawing style, transformation matrix, color, and font. To set drawing style attributes, use the methods of `NSBezierPath`. Other attributes are accessed through appropriate objects such as `NSAffineTransform`, `NSColor`, and `NSFont`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSGraphicsContext.h

setColorRenderingIntent:

Sets the rendering intent in the receiver's graphics state.

```
- (void)setColorRenderingIntent:(NSColorRenderingIntent)renderingIntent
```

Parameters

renderingIntent

An “[Creating a Graphics Context](#)” (page 6) value that specifies the rendering intent to be used. For possible values see “[NSColorRenderingIntent](#)” (page 23).

Discussion

The rendering intent specifies how Cocoa should handle colors that are not located within the gamut of the destination color space of a graphics context. If you do not explicitly set the rendering intent, and sampled images are being drawn, `NSGraphicsContext` uses perceptual rendering intent. Otherwise, `NSGraphicsContext` uses relative colorimetric rendering intent.

Availability

Available in Mac OS X v10.5 and later.

See Also

- [colorRenderingIntent](#) (page 14)

Declared In

NSGraphicsContext.h

setCompositingOperation:

Sets the receiver's global compositing operation.

```
- (void)setCompositingOperation:(NSCompositingOperation)operation
```

Parameters

operation

A constant that specifies a compositing operation. See `NSCompositingOperation` for valid constants.

Discussion

The compositing operation is a global attribute of the graphics context and affects drawing operations that do not take an explicit compositing operation parameter. For methods that do take an explicit compositing operation parameter, the value of that parameter supersedes the global value.

The compositing operations are related to (but different from) the blend mode settings used in Quartz. Only the default compositing operation (`NSCompositeCopy`) is supported when rendering PDF or PostScript content.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [compositingOperation](#) (page 14)

Related Sample Code

ImageMap

ImageMapExample

Declared In

NSGraphicsContext.h

setFocusStack:

Sets the object used by the receiver to track the hierarchy of views with locked focus.

```
- (void)setFocusStack:(void *)stack
```

Parameters

stack

The object used by the graphics context for view-hierarchy tracking.

Discussion

You should never need to get or modify the focus stack information. The use of focus stacks may be deprecated in a future release.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.5.

Declared In

NSGraphicsContext.h

setImageInterpolation:

Sets the receiver's interpolation behavior.

- (void)setImageInterpolation:(NSImageInterpolation)*interpolation***Parameters***interpolation*A constant specifying the image-interpolation behavior. The NSImageInterpolation constants are described in [NSImageInterpolation](#) (page 22).**Discussion**Note that this value is not part of the graphics state, so it cannot be reset using [restoreGraphicsState](#) (page 17).**Availability**

Available in Mac OS X v10.0 and later.

See Also- [imageInterpolation](#) (page 16)**Related Sample Code**

WebKitDOMElementPlugIn

Declared In

NSGraphicsContext.h

setPatternPhase:

Sets the amount to offset the pattern color when filling the receiver.

- (void)setPatternPhase:(NSPoint)*phase***Parameters***phase*

A point specifying the offset.

Discussion

Use this method when you need to line up the pattern color with another pattern, such as the pattern in a superview.

The pattern phase is a translation (width, height) applied before a pattern is drawn in the current context and is part of the saved graphics state of the context. The default pattern phase is (0,0). Setting the pattern phase has the effect of temporarily changing the pattern matrix of any pattern you decide to draw. For example, setting the pattern phase to (2,3) has the effect of moving the start of pattern cell tiling to the point (2,3) in default user space.

Availability

Available in Mac OS X v10.2 and later.

See Also

- [patternPhase](#) (page 17)

Declared In

NSGraphicsContext.h

setShouldAntialias:

Sets whether the receiver should use antialiasing.

- (void)setShouldAntialias:(BOOL)*antialias*

Parameters

antialias

YES if the receiver should use antialiasing, otherwise NO.

Discussion

This value is part of the graphics state and is restored by [restoreGraphicsState](#) (page 17).

Availability

Available in Mac OS X v10.0 and later.

See Also

- [shouldAntialias](#) (page 21)

Related Sample Code

Cocoa OpenGL

Cropped Image

From A View to A Movie

From A View to A Picture

Declared In

NSGraphicsContext.h

shouldAntialias

Returns a Boolean value that indicates whether the receiver uses antialiasing.

- (BOOL)shouldAntialias

Return Value

YES if the receiver uses antialiasing, otherwise NO.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [setShouldAntialias:](#) (page 21)

Declared In

NSGraphicsContext.h

Constants

Attribute dictionary keys

These constants are dictionary keys used by [graphicsContextWithAttributes:](#) (page 9) and [attributes](#) (page 13).

```
NSString *NSGraphicsContextDestinationAttributeName;
NSString *NSGraphicsContextRepresentationFormatAttributeName;
```

Constants

`NSGraphicsContextDestinationAttributeName`

Can be an instance of `NSWindow` or `NSBitmapImageRep` when creating a graphics context.

When determining the type of a graphics context, this value can be an `NSMutableData`, `NSString`, or `NSURL` object.

Available in Mac OS X v10.0 and later.

Declared in `NSGraphicsContext.h`.

`NSGraphicsContextRepresentationFormatAttributeName`

Specifies the destination file format.

This value should be retrieved only and not used to create a graphics context.

Available in Mac OS X v10.0 and later.

Declared in `NSGraphicsContext.h`.

Representation format attribute keys

These constants are possible values for the `NSGraphicsContextRepresentationFormatAttributeName` key in a graphic context's attribute dictionary.

```
NSString *NSGraphicsContextPSFormat;
NSString *NSGraphicsContextPDFFormat;
```

Constants

`NSGraphicsContextPDFFormat`

Destination file format is PDF.

Available in Mac OS X v10.0 and later.

Declared in `NSGraphicsContext.h`.

`NSGraphicsContextPSFormat`

Destination file format is PostScript.

Available in Mac OS X v10.0 and later.

Declared in `NSGraphicsContext.h`.

NSImageInterpolation

These interpolations are used by [imageInterpolation](#) (page 16) and [setImageInterpolation:](#) (page 20).

```
enum {
    NSImageInterpolationDefault,
    NSImageInterpolationNone,
    NSImageInterpolationLow,
    NSImageInterpolationMedium,
    NSImageInterpolationHigh
};
typedef NSUInteger NSImageInterpolation;
```

Constants

`NSImageInterpolationDefault`
Use the context's default interpolation.
 Available in Mac OS X v10.0 and later.
 Declared in `NSGraphicsContext.h`.

`NSImageInterpolationNone`
No interpolation.
 Available in Mac OS X v10.0 and later.
 Declared in `NSGraphicsContext.h`.

`NSImageInterpolationLow`
Fast, low-quality interpolation.
 Available in Mac OS X v10.0 and later.
 Declared in `NSGraphicsContext.h`.

`NSImageInterpolationMedium`
Medium quality, slower than `NSImageInterpolationLow`.
 Available in Mac OS X v10.6 and later.
 Declared in `NSGraphicsContext.h`.

`NSImageInterpolationHigh`
Slower, higher-quality interpolation.
 Available in Mac OS X v10.0 and later.
 Declared in `NSGraphicsContext.h`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSGraphicsContext.h`

NSColorRenderingIntent

These constants specify how Cocoa should handle colors that are not located within the destination color space of a graphics context. These constants are used by the methods [setColorRenderingIntent:](#) (page 18) and [colorRenderingIntent](#) (page 14).

```
enum {
    NSColorRenderingIntentDefault,
    NSColorRenderingIntentAbsoluteColorimetric,
    NSColorRenderingIntentRelativeColorimetric,
    NSColorRenderingIntentPerceptual,
    NSColorRenderingIntentSaturation
};
typedef NSInteger NSColorRenderingIntent;
```

Constants

`NSColorRenderingIntentDefault`

Use the default rendering intent for the graphics context.

Available in Mac OS X v10.5 and later.

Declared in `NSGraphics.h`.

`NSColorRenderingIntentAbsoluteColorimetric`

Map colors outside of the gamut of the output device to the closest possible match inside the gamut of the output device.

This operation can produce a clipping effect, where two different color values in the gamut of the graphics context are mapped to the same color value in the output device's gamut. Unlike the relative colorimetric, absolute colorimetric does not modify colors inside the gamut of the output device.

Available in Mac OS X v10.5 and later.

Declared in `NSGraphics.h`.

`NSColorRenderingIntentRelativeColorimetric`

Map colors outside of the gamut of the output device to the closest possible match inside the gamut of the output device.

This operation can produce a clipping effect, where two different color values in the gamut of the graphics context are mapped to the same color value in the output device's gamut. The relative colorimetric shifts all colors (including those within the gamut) to account for the difference between the white point of the graphics context and the white point of the output device.

Available in Mac OS X v10.5 and later.

Declared in `NSGraphics.h`.

`NSColorRenderingIntentPerceptual`

Preserve the visual relationship between colors by compressing the gamut of the graphics context to fit inside the gamut of the output device.

Perceptual intent is good for photographs and other complex, detailed images.

Available in Mac OS X v10.5 and later.

Declared in `NSGraphics.h`.

`NSColorRenderingIntentSaturation`

Preserve the relative saturation value of the colors when converting into the gamut of the output device.

The result is an image with bright, saturated colors. Saturation intent is good for reproducing images with low detail, such as presentation charts and graphs.

Available in Mac OS X v10.5 and later.

Declared in `NSGraphics.h`.

Document Revision History

This table describes the changes to *NSGraphicsContext Class Reference*.

Date	Notes
2009-06-24	Reformatted constants.
2009-06-02	Updated for Mac OS X v10.6. Added NSImageInterpolationMedium.
2007-03-01	Updated for Mac OS version 10.5.
2006-11-07	Documented the CGContext instance method.
2006-05-23	First publication of this content as a separate document.

REVISION HISTORY

Document Revision History

Index

A

Attribute dictionary keys [22](#)
attributes [instance method 13](#)

C

CIColorContext [instance method 13](#)
colorRenderingIntent [instance method 14](#)
compositingOperation [instance method 14](#)
currentContext [class method 8](#)
currentContextDrawingToScreen [class method 8](#)

F

flushGraphics [instance method 15](#)
focusStack [instance method 15](#)

G

graphicsContextWithAttributes: [class method 9](#)
graphicsContextWithBitmapImageRep: [class method 9](#)
graphicsContextWithGraphicsPort:flipped: [class method 10](#)
graphicsContextWithWindow: [class method 10](#)
graphicsPort [instance method 15](#)

I

imageInterpolation [instance method 16](#)
isDrawingToScreen [instance method 16](#)
isFlipped [instance method 16](#)

N

NSColorRenderingIntent [23](#)
NSColorRenderingIntentAbsoluteColorimetric [constant 24](#)
NSColorRenderingIntentDefault [constant 24](#)
NSColorRenderingIntentPerceptual [constant 24](#)
NSColorRenderingIntentRelativeColorimetric [constant 24](#)
NSColorRenderingIntentSaturation [constant 24](#)
NSGraphicsContextDestinationAttributeName [constant 22](#)
NSGraphicsContextPDFFormat [constant 22](#)
NSGraphicsContextPSFormat [constant 22](#)
NSGraphicsContextRepresentationFormatAttributeName [constant 22](#)
NSImageInterpolation [data type 22](#)
NSImageInterpolationDefault [constant 23](#)
NSImageInterpolationHigh [constant 23](#)
NSImageInterpolationLow [constant 23](#)
NSImageInterpolationMedium [constant 23](#)
NSImageInterpolationNone [constant 23](#)

P

patternPhase [instance method 17](#)

R

Representation format attribute keys [22](#)
restoreGraphicsState [class method 11](#)
restoreGraphicsState [instance method 17](#)

S

saveGraphicsState [class method 11](#)
saveGraphicsState [instance method 18](#)

setColorRenderingIntent: **instance method** [18](#)
setCompositingOperation: **instance method** [19](#)
setCurrentContext: **class method** [12](#)
setFocusStack: **instance method** [19](#)
setGraphicsState: **class method** [12](#)
setImageInterpolation: **instance method** [20](#)
setPatternPhase: **instance method** [20](#)
setShouldAntialias: **instance method** [21](#)
shouldAntialias **instance method** [21](#)