Quartz Composer Reference Collection

Graphics & Animation

2007-01-25

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

Contents

Tables

The Quartz Composer Reference Collection

Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Header file directories	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework/Headers
Declared in	QCComposition.h QCCompositionLayer.h QCCompositionParameterView.h QCCompositionPickerPanel.h QCCompositionPickerView.h QCCompositionRepository.h QCPlugIn.h QCPlugInViewController.h QCRenderer.h QCView.h

The Quartz Composer Reference Collection defines Objective-C classes that, in one way or another, work with compositions built using the Quartz Composer development tool. The classes support the following programming tasks:

- Load, play, and control compositions stored as Quartz Composer files (`qtz` extension). See the `QCView` and `QCRenderer` classes.
- Access and render compositions that are stored in a system-wide repository. See the `QCCompositionXXX` classes.
- Load and play a composition in a Core Animation layer. See the `QCCompositionLayer` class.
- Create a custom patch that can be used from the Quartz Composer development tool. See the `QCPlugInXXX` classes.

The Quartz Composer Reference Collection

Classes

PART I

Classes

QCComposition Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCComposition.h
Companion guide	Quartz Composer Programming Guide
Related sample code	Quartz Composer SlideShow

Overview

The `QCComposition` class represents a Quartz Composer composition that either:

- comes from the system-wide composition repository (`/Library/Compositions` and `~/Library/Compositions`) where it can be accessed by any application through the methods of the `QCCompositionRepository` class
- is created from an arbitrary source (typically a file on disk) using one of its methods

This class cannot be subclassed.

A `QCComposition` object has the following information associated with it and that you can obtain by using the appropriate method of the `QCComposition` class:

- Attributes include the name and description of the composition, copyright information, and whether or not its provided by Mac OS X (built-in).
- The protocols that the composition conforms to. A **composition protocol** defines a set of required and optional input parameters and output results.

Many methods of the `QCRenderer`, `QCCompositionLayer`, and `QCView` classes take a `QCComposition` object as a parameter.

Tasks

Creating a Composition

`+ compositionWithFile:` (page 15)
Returns an autoreleased composition object initialized with a Quartz Composer composition file.

`+ compositionWithData:` (page 14)
Returns an autoreleased composition object initialized with the contents of a Quartz Composer composition file.

Getting Information About a Composition

`- attributes` (page 15)
Returns the attributes of the composition.

`- protocols` (page 16)
Returns the list of protocols to which the composition conforms.

`- identifier` (page 15)
Returns the unique and persistent identifier for the composition from the composition repository.

Getting Port Keys

`- inputKeys` (page 16)
Returns an array listing the keys that identify the input ports of the root patch of the composition.

`- outputKeys` (page 16)
Returns an array listing the keys that identify the output ports of the root patch of the composition.

Class Methods

compositionWithData:

Returns an autoreleased composition object initialized with the contents of a Quartz Composer composition file.

```
+ (QCComposition*) compositionWithData:(NSData*)data;
```

Parameters

data
The contents of a file created with the Quartz Composer developer tool.

Return Value
A Quartz Composer composition object or `nil` if there is an error.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCComposition.h`

compositionWithFile:

Returns an autoreleased composition object initialized with a Quartz Composer composition file.

```
+ (QCComposition*) compositionWithFile:(NSString*)path;
```

Parameters

path

A path to a file created with the Quartz Composer developer tool (`.qtz` extension).

Return Value
A Quartz Composer composition object or `nil` if there is an error.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCComposition.h`

Instance Methods

attributes

Returns the attributes of the composition.

```
- (NSDictionary*) attributes
```

Return Value
A dictionary of composition attributes. See "`Attribute Keys`" (page 17) for the attributes that can be returned.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCComposition.h`

identifier

Returns the unique and persistent identifier for the composition from the composition repository.

```
- (NSString*) identifier
```

Return Value
The unique identifier for the composition if it comes from the composition repository; `nil` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCCompositionRepository.h

inputKeys

Returns an array listing the keys that identify the input ports of the root patch of the composition.

```
- (NSArray*) inputKeys
```

Return Value
An array of input keys.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCComposition.h

outputKeys

Returns an array listing the keys that identify the output ports of the root patch of the composition.

```
- (NSArray*) outputKeys
```

Return Value
An array of output keys.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCComposition.h

protocols

Returns the list of protocols to which the composition conforms.

```
- (NSArray*) protocols
```

Return Value
A list of protocols. See "Standard Protocols" (page 21).

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCComposition.h

Constants

Attribute Keys

Attributes of a composition.

```
extern NSString* const QCCompositionAttributeNameKey;
extern NSString* const QCCompositionAttributeDescriptionKey;
extern NSString* const QCCompositionAttributeCopyrightKey;
extern NSString* const QCCompositionAttributeBuiltInKey;
extern NSString* const QCCompositionAttributeTimeDependentKey;
extern NSString* const QCCompositionAttributeHasConsumersKey;
extern NSString* const QCCompositionAttributeCategoryKey;
```

Constants

`QCCompositionAttributeNameKey`
The key for the composition name. The associated value is an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `QCComposition.h`.

`QCCompositionAttributeDescriptionKey`
The key for the composition description. The associated value is an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `QCComposition.h`.

`QCCompositionAttributeCopyrightKey`
The key for composition copyright information. The associated value is an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `QCComposition.h`.

`QCCompositionAttributeBuiltInKey`
The key for the composition origin. The associated value is an `NSNumber` object that contains a Boolean value. `YES` indicates the composition is built-in (provided by Mac OS X).

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionAttributeTimeDependentKey`
The key for the composition time dependency. The associated value is an `NSNumber` object that contains a Boolean value. `YES` indicates that the composition is time dependent.

`QCCompositionAttributeHasConsumersKey`
The key for a composition that has consumer patches. The associated value is an `NSNumber` object that contains a Boolean value. `YES` indicates that the composition has consumers.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionAttributeCategoryKey`
The composition category. The associated value is a category constant. See "Composition Categories" (page 18).

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

Declared In
QCComposition.h

Composition Categories

Categories for compositions.

```
extern NSString* const QCCompositionCategoryDistortion;
extern NSString* const QCCompositionCategoryStylize;
extern NSString* const QCCompositionCategoryUtility;
```

Constants

QCCompositionCategoryDistortion
: A composition that produces a distortion effect.

  Available in Mac OS X v10.5 and later.

  Declared in QCComposition.h.

QCCompositionCategoryStylize
: A composition that produces a stylize effect.

  Available in Mac OS X v10.5 and later.

  Declared in QCComposition.h.

QCCompositionCategoryUtility
: A utility composition.

  Available in Mac OS X v10.5 and later.

  Declared in QCComposition.h.

Declared In
QCComposition.h

Standard Protocol Input Keys

Input ports of a composition.

```
extern NSString* const QCCompositionInputImageKey;
extern NSString* const QCCompositionInputSourceImageKey;
extern NSString* const QCCompositionInputDestinationImageKey;
extern NSString* const QCCompositionInputRSSFeedURLKey;
extern NSString* const QCCompositionInputRSSArticleDurationKey;
extern NSString* const QCCompositionInputPreviewModeKey;
extern NSString* const QCCompositionInputXKey;
extern NSString* const QCCompositionInputYKey;
extern NSString* const QCCompositionInputScreenImageKey;
extern NSString* const QCCompositionInputAudioPeakKey;
extern NSString* const QCCompositionInputAudioSpectrumKey;
extern NSString* const QCCompositionInputTrackPositionKey;
extern NSString* const QCCompositionInputTrackInfoKey;
extern NSString* const QCCompositionInputTrackSignalKey;
extern NSString* const QCCompositionInputPrimaryColorKey;
extern NSString* const QCCompositionInputSecondaryColorKey;
extern NSString* const QCCompositionInputPaceKey;
```

Constants

`QCCompositionInputImageKey`
An image input port whose key is `inputImage`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputSourceImageKey`
An image input port whose key is `inputSourceImage`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputDestinationImageKey`
An image input port whose key is `inputDestinationImage`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputRSSFeedURLKey`
A string input port whose key is `inputRSSFeedURL`. This port must be passed an http or feed scheme URL.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputRSSArticleDurationKey`
A number input port whose key is `inputRSSArticleDuration`. The value must be expressed in seconds.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputPreviewModeKey`
A Boolean input port whose key is `inputPreviewMode`. When the value of this input port is set to `TRUE`, the composition that provides this port must be able to run in a low-quality mode that produces a preview of the composition.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputXKey`
A number input port whose key is `inputX`. The value must be normalized to the image width with the origin on the left.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputYKey`
A number input port whose key is `inputY`. The value must be normalized to the image height with the origin at the bottom.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputScreenImageKey`
An image input port whose key is `inputScreenImage`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputAudioPeakKey`
A number input port whose key is `inputAudioPeak`. The value must be in the `[0,1]` range as a mono signal with no decay applied.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputAudioSpectrumKey`
A structure input port whose key is `inputAudioSpectrum`. The structure must contain 16 values in the `[0,1]` range representing 16 spectrum bands of the mono signal from low to high frequencies with no decay applied.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputTrackPositionKey`
A number input port whose key is `inputTrackPosition`. The value must be expressed in seconds.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputTrackInfoKey`
A structure input port whose key is `inputTrackInfo`. The structure contains optional entries, such as “name”, “artist”, "album", "duration", "artwork", and so on.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputTrackSignalKey`
A Boolean input port whose key is `inputTrackSignal`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputPrimaryColorKey`
A color input port whose key is `inputPrimaryColor`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputSecondaryColorKey`
A color input port whose key is `inputSecondaryColor`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionInputPaceKey`
A number input port whose key is `inputPace`. The value must be in the `[0,1]` range.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

Declared In
`QCComposition.h`

Standard Protocol Output Keys

Output ports of a composition.

```
extern NSString* const QCCompositionOutputImageKey;
extern NSString* const QCCompositionOutputWebPageURLKey;
```

Constants

`QCCompositionOutputImageKey`

An image output port whose key is `outputImage`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionOutputWebPageURLKey`

A string output port whose key is `outputWebPageURL`.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

Declared In

`QCComposition.h`

Standard Protocols

Protocols for a composition.

```
extern NSString* const QCCompositionProtocolGraphicAnimation;
extern NSString* const QCCompositionProtocolGraphicTransition;
extern NSString* const QCCompositionProtocolImageFilter;
extern NSString* const QCCompositionProtocolImageCompositor;
extern NSString* const QCCompositionProtocolImageTransition;
extern NSString* const QCCompositionProtocolScreenSaverRSS;
```

Constants

`QCCompositionProtocolGraphicAnimation`

A composition that renders a generic graphical animation. It has the option to use `QCCompositionInputPrimaryColorKey` (page 20) for the primary color of the animation, `QCCompositionInputSecondaryColorKey` (page 20) for the secondary color of the animation, `QCCompositionInputPaceKey` (page 20) for the global pace of the animation, and `QCCompositionInputPreviewModeKey` (page 19) to indicate if the animation should run in lower-quality for preview purposes.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionProtocolGraphicTransition`

A composition that performs a transition between two images, using a transition time in range of `0` to `1`. A conforming composition must use the input keys `QCCompositionInputSourceImageKey` (page 19) for the starting image and `QCCompositionInputDestinationImageKey` (page 19) for the image to transition to. The composition can optionally use `QCCompositionInputPreviewModeKey` (page 19) to indicate if the animation should run in lower-quality for preview purposes.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionProtocolImageFilter`

A composition that applies an effect to a source image. A conforming composition must use the input key `QCCompositionInputImageKey` (page 19) for the source image and `QCCompositionOutputImageKey` (page 21) for the output image. The composition can optionally use `QCCompositionInputXKey` (page 19) to specify the X position of the center point of the effect, `QCCompositionInputYKey` (page 19) to specify the Y position of the center point of the effect, and`QCCompositionInputPreviewModeKey` (page 19) to indicate if the animation should run in lower-quality for preview purposes.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionProtocolScreenSaver`

A composition that can be used as a screen saver. The composition has the option to use `QCCompositionInputScreenImageKey` (page 19) for a screenshot image of the screen that the screen saver runs on, `QCCompositionInputPreviewModeKey` (page 19) to indicate if the animation should run in lower-quality for preview purposes, and `QCCompositionOutputWebPageURLKey` (page 21) for a URL to open in the default web browser when screen saver exits (only allowed if screen saver password is disabled).

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionProtocolImageTransition`

A composition that performs a transition between two images, using a parametric time value to drives the transition from start (at time `0`) to end (at time `1`). A conforming composition must use the input keys `QCCompositionInputImageKey` (page 19) for the starting image and `QCCompositionInputDestinationImageKey` (page 19) for the ending image. The composition can optionally use `QCCompositionInputPreviewModeKey` (page 19) to indicate if the animation should run in lower-quality for preview purposes.

`QCCompositionProtocolRSSVisualizer`

A composition that acts as a visualizer for an RSS feed. A conforming composition must use the input key `QCCompositionInputRSSFeedURLKey` (page 19) for the URL to use for the RSS feed. It can optionally use `QCCompositionInputRSSArticleDurationKey` (page 19) to specify the duration of each feed article.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

`QCCompositionProtocolMusicVisualizer`

A composition that acts as a visualizer for music. A conforming composition must use the input key `QCCompositionInputAudioPeakKey` (page 20) for the instantaneous audio peak and the `QCCompositionInputAudioSpectrumKey` (page 20) for the instantaneous audio spectrum. It can optionally use the `QCCompositionInputTrackInfoKey` (page 20) to indicate it receives information about the current track and the `QCCompositionInputTrackSignalKey` (page 20) to indicate the start of a new track.

Available in Mac OS X v10.5 and later.

Declared in `QCComposition.h`.

Declared In

`QCComposition.h`

QCCompositionLayer Class Reference

Inherits from	CAOpenGLLayer : CALayer : NSObject
Conforms to	QCCompositionRenderer NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCCompositionLayer.h
Companion guides	Core Animation Programming Guide Quartz Composer Programming Guide
Related sample code	CALayerEssentials CocoaSlides

Overview

The `QCCompositionLayer` class loads, plays, and controls Quartz Composer compositions in a Core Animation layer hierarchy. The composition tracks the Core Animation layer time and is rendered directly at the current dimensions of the `QCCompositionLayer` object.

An archived `QCCompositionLayer` object saves the composition that's loaded at the time the layer is archived. It detects layer usage and pauses or resumes the composition appropriately. A `QCCompositionLayer` object starts rendering the composition automatically when the layer is placed in a visible layer hierarchy. The layer stops rendering when it is hidden or removed from the visible layer hierarchy.

You can pass data to the input ports, or retrieve data from the output ports, of the root patch of a composition by accessing the `patch` attribute of the `QCCompositionLayer` instance using methods provided by the `QCCompositionRenderer` protocol.

Note: You must not modify the `asynchronous` property of the superclass `CAOpenGLLayer`.

Tasks

Creating the Layer

`+ compositionLayerWithFile:` (page 25)
Creates and returns an instance of a composition layer using the Quartz Composer composition in the specified file.

`+ compositionLayerWithComposition:` (page 24)
Creates and returns an instance of a composition layer using the provided Quartz Composer composition.

`- initWithFile:` (page 26)
Initializes and returns a composition layer using the Quartz Composer composition in the specified file.

`- initWithComposition:` (page 25)
Initializes and returns a composition layer using the provided Quartz Composer composition.

Getting the Composition

`- composition` (page 25)
Returns the composition associated with the layer.

Class Methods

compositionLayerWithComposition:

Creates and returns an instance of a composition layer using the provided Quartz Composer composition.

```
+ (QCCompositionLayer*)compositionLayerWithComposition:(QCComposition*)composition
```

Parameters

composition
The Quartz Composer composition to use as content.

Return Value
An autoreleased, initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ compositionLayerWithFile:` (page 25)

Declared In
QCCompositionLayer.h

compositionLayerWithFile:

Creates and returns an instance of a composition layer using the Quartz Composer composition in the specified file.

```
+ (QCCompositionLayer*)compositionLayerWithFile:(NSString*)path
```

Parameters

path
A string that specifies the location of a Quartz Composer composition.

Return Value
An autoreleased, initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ compositionLayerWithComposition:` (page 24)

Related Sample Code
CALayerEssentials
CocoaSlides

Declared In
QCCompositionLayer.h

Instance Methods

composition

Returns the composition associated with the layer.

```
- (QCComposition*) composition
```

Return Value
The composition object associated with the layer or `nil` if there is none.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCCompositionLayer.h

initWithComposition:

Initializes and returns a composition layer using the provided Quartz Composer composition.

```
- (id)initWithComposition:(QCComposition*)composition
```

Parameters

composition

The Quartz Composer composition to use as content.

Return Value

The initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- initWithFile:` (page 26)

Declared In

`QCCompositionLayer.h`

initWithFile:

Initializes and returns a composition layer using the Quartz Composer composition in the specified file.

```
- (id)initWithFile:(NSString*)path
```

Parameters

path

A string that specifies the location of a Quartz Composer composition.

Return Value

The initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- initWithComposition:` (page 25)

Declared In

`QCCompositionLayer.h`

QCCompositionParameterView Class Reference

Inherits from	NSView : NSResponder : NSObject
Conforms to	NSAnimatablePropertyContainer (NSView) NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCCompositionParameterView.h
Companion guide	Quartz Composer Programming Guide

Overview

The `QCCompositionParameterView` class allows allows users to edit, in real time, the input parameters of a composition. The composition can be rendering in any of the following objects: `QCRenderer`, `QCView`, or `QCCompositionLayer`.

Tasks

Getting and Setting the Renderer

- `setCompositionRenderer:` (page 30)
  Sets the composition parameter view for editing the input parameters of the provided renderer object.
- `compositionRenderer` (page 28)
  Returns the renderer object associated with the composition parameter view.

Checking for Input Parameters

- `hasParameters` (page 29)
  Checks whether the composition that is currently edited by the composition parameter view has any input parameters.

Setting and Retrieving the Delegate

- `setDelegate:` (page 30)
  Sets the composition parameter view delegate.
- `delegate` (page 29)
  Returns the composition parameter view delegate.

Managing Background Drawing

- `setDrawsBackground:` (page 31)
  Sets whether the composition parameter view draws its background.
- `drawsBackground` (page 29)
  Returns whether the composition parameter view draws its background.

Setting and Getting the Background Color

- `setBackgroundColor:` (page 30)
  Sets the background color of the composition parameter view.
- `backgroundColor` (page 28)
  Retrieves the background color of the composition parameter view.

Instance Methods

backgroundColor

Retrieves the background color of the composition parameter view.

```
- (NSColor*) backgroundColor;
```

Return Value
The color of the background.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionParameterView.h`

compositionRenderer

Returns the renderer object associated with the composition parameter view.

```
- (id<QCCompositionRenderer>) compositionRenderer
```

Return Value
A renderer object or `nil`, if the composition parameter view is not set to a renderer object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setCompositionRenderer:` (page 30)

Declared In
`QCCompositionParameterView.h`

delegate

Returns the composition parameter view delegate.

```
- (id) delegate;
```

Return Value
The composition parameter view delegate.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionParameterView.h`

drawsBackground

Returns whether the composition parameter view draws its background.

```
- (BOOL) drawsBackground;
```

Return Value
`YES` if the view draws its background; otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionParameterView.h`

hasParameters

Checks whether the composition that is currently edited by the composition parameter view has any input parameters.

```
- (BOOL) hasParameters
```

Return Value
`YES` if the composition has any input parameters.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionParameterView.h`

setBackgroundColor:

Sets the background color of the composition parameter view.

```
- (void) setBackgroundColor:(NSColor*)color;
```

Parameters

color

The color to set.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionParameterView.h`

setCompositionRenderer:

Sets the composition parameter view for editing the input parameters of the provided renderer object.

```
- (void) setCompositionRenderer:(id<QCCompositionRenderer>)renderer
```

Parameters

renderer

A `QCCompositionRenderer` object, either `QCView`, `QCRenderer`, or `QCCompositionLayer`. Pass `nil` to unset this renderer.

Discussion
If the renderer is a `QCView` object, the view track the composition.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- compositionRenderer` (page 28)

Declared In
`QCCompositionParameterView.h`

setDelegate:

Sets the composition parameter view delegate.

```
- (void) setDelegate:(id)delegate;
```

Parameters

delegate

The delegate for the composition parameter view.

Availability

Available in Mac OS X v10.5 and later.

Declared In

QCCompositionParameterView.h

setDrawsBackground:

Sets whether the composition parameter view draws its background.

```
- (void) setDrawsBackground:(BOOL)flag;
```

Parameters

flag

YES for the view to draw its background; otherwise NO.

Availability

Available in Mac OS X v10.5 and later.

Declared In

QCCompositionParameterView.h

QCCompositionPickerPanel Class Reference

Inherits from	NSPanel : NSWindow : NSResponder : NSObject
Conforms to	NSUserInterfaceValidations (NSWindow) NSAnimatablePropertyContainer (NSWindow) NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCCompositionPickerPanel.h
Companion guide	Quartz Composer Programming Guide

Overview

The `QCCompositionPickerPanel` class represents a utility window that allows users to browse compositions that are in the Quartz Composer composition repository and, if supported, preview the composition. The `QCCompositionPickerPanel` class cannot be subclassed.

Tasks

Creating the Utility Window for Browsing Compositions

`+ sharedCompositionPickerPanel` (page 34)
Returns the shared instance of the composition picker panel.

Getting the Picker Panel View

`- compositionPickerView` (page 34)
Returns the composition picker view used by the panel so that it can be configured.

Class Methods

sharedCompositionPickerPanel

Returns the shared instance of the composition picker panel.

```
+ (QCCompositionPickerPanel*) sharedCompositionPickerPanel
```

Return Value
The shared `QCCompositionPickerPanel` object.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionPickerPanel.h`

Instance Methods

compositionPickerView

Returns the composition picker view used by the panel so that it can be configured.

```
- (QCCompositionPickerView*) compositionPickerView;
```

Return Value
The `QCCompositionPickerView` used by the composition picker panel.

Discussion
After you retrieve the view, you can configure it.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionPickerPanel.h`

Notifications

QCCompositionPickerPanelDidSelectCompositionNotification

Posted when the user chooses a composition.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionPickerPanel.h`

QCCompositionPickerView Class Reference

Inherits from	NSView : NSResponder : NSObject
Conforms to	NSAnimatablePropertyContainer (NSView) NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCCompositionPickerView.h
Companion guide	Quartz Composer Programming Guide

Overview

The `QCCompositionPickerView` class allows users to browse compositions that are in the Quartz Composer composition repository, and to preview them. You can set the default input parameters for a composition preview by using the method `setDefaultValue:forInputKey:`.

Note that the composition picker view does not automatically refresh its content when the composition repository is updated. It's your responsibility to perform any necessary updating.

Tasks

Setting and Getting the Background Color

- `setBackgroundColor:` (page 42)
  Sets the background color for the composition picker view.
- `backgroundColor` (page 38)
  Returns the background color of the composition picker view.

Managing Background Drawing

- `setDrawsBackground:` (page 44)
  Sets whether the composition picker view draws its background.

- `drawsBackground` (page 39)
  Returns whether the composition picker view draws its background.

Setting Composition Input Parameters

- `setDefaultValue:forInputKey:` (page 43)
  Sets the default value to use for a composition input parameter.
- `resetDefaultInputValues` (page 41)
  Clears all previously set default values for composition input parameters.

Managing Animation

- `startAnimation:` (page 46)
  Starts animating the composition in the composition picker view.
- `stopAnimation:` (page 46)
  Stops animating the composition that is currently animating in the composition picker view.
- `isAnimating` (page 39)
  Returns whether or not the composition picker view is currently animating its composition.
- `setMaxAnimationFrameRate:` (page 44)
  Sets the maximum frame rate for animating compositions.
- `maxAnimationFrameRate` (page 40)
  Retrieves the maximum frame rate for animating compositions.

Controlling Display of Composition Names

- `setShowsCompositionNames:` (page 45)
  Enables the display of composition names in the composition picker view.
- `showsCompositionNames` (page 46)
  Retrieves whether composition names can be shown in the composition picker view.

Setting and Retrieving the View Delegate

- `setDelegate:` (page 43)
  Sets the composition picker view delegate.
- `delegate` (page 39)
  Retrieves the composition picker view delegate.

Managing the Composition Picker View

- `setCompositionsFromRepositoryWithProtocol:andAttributes:` (page 42)
  Sets the compositions in the composition picker view to those that match the specified criteria.

- `compositions` (page 38)
  Returns the list of compositions that are currently in the composition picker view.
- `setAllowsEmptySelection:` (page 41)
  Sets whether to allow an empty selection in the composition picker view.
- `allowsEmptySelection` (page 37)
  Retrieves the empty-selection state of the composition picker view.
- `setCompositionAspectRatio:` (page 42)
  Sets the aspect ratio used to display compositions in the composition picker view.
- `compositionAspectRatio` (page 38)
  Retrieves the aspect ratio used to display compositions in the composition picker view.
- `setSelectedComposition:` (page 45)
  Sets a composition as selected in the composition picker view.
- `selectedComposition` (page 41)
  Returns the composition that is currently selected in the composition picker view.

Working with Columns and Rows

- `setNumberOfColumns:` (page 44)
  Sets the number of columns in the composition picker view.
- `numberOfColumns` (page 40)
  Retrieves the number of columns in the composition picker view.
- `setNumberOfRows:` (page 45)
  Sets the number of rows in the composition picker view.
- `numberOfRows` (page 40)
  Retrieves the number of rows in the composition picker view.

Instance Methods

allowsEmptySelection

Retrieves the empty-selection state of the composition picker view.

```
- (BOOL) allowsEmptySelection
```

Return Value
`YES` if an empty selection is allowed `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setAllowsEmptySelection:` (page 41)

Declared In
`QCCompositionPickerView.h`

backgroundColor

Returns the background color of the composition picker view.

```
- (NSColor*) backgroundColor;
```

Return Value
The background color.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setBackgroundColor:` (page 42)

Declared In
`QCCompositionPickerView.h`

compositionAspectRatio

Retrieves the aspect ratio used to display compositions in the composition picker view.

```
- (NSSize) compositionAspectRatio
```

Return Value
The aspect ratio.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setCompositionAspectRatio:` (page 42)

Declared In
`QCCompositionPickerView.h`

compositions

Returns the list of compositions that are currently in the composition picker view.

```
- (NSArray*) compositions
```

Return Value
An array of `QCComposition` objects.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setCompositionsFromRepositoryWithProtocol:andAttributes:` (page 42)

Declared In
`QCCompositionPickerView.h`

delegate

Retrieves the composition picker view delegate.

```
- (id) delegate
```

Return Value
The delegate.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setDelegate:` (page 43)

Declared In
`QCCompositionPickerView.h`

drawsBackground

Returns whether the composition picker view draws its background.

```
- (BOOL) drawsBackground;
```

Return Value
`YES` if the composition picker view draws its background; otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setDrawsBackground:` (page 44)

Declared In
`QCCompositionPickerView.h`

isAnimating

Returns whether or not the composition picker view is currently animating its composition.

```
- (BOOL) isAnimating
```

Return Value
`YES` if a composition is animating in the composition picker view; `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `startAnimation:` (page 46)
- `stopAnimation:` (page 46)

Declared In
`QCCompositionPickerView.h`

maxAnimationFrameRate

Retrieves the maximum frame rate for animating compositions.

```
- (float) maxAnimationFrameRate
```

Return Value
The maximum frame rate.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setMaxAnimationFrameRate: (page 44)

Declared In
QCCompositionPickerView.h

numberOfColumns

Retrieves the number of columns in the composition picker view.

```
- (NSUInteger) numberOfColumns;
```

Return Value
The number of columns.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setNumberOfColumns: (page 44)

Declared In
QCCompositionPickerView.h

numberOfRows

Retrieves the number of rows in the composition picker view.

```
- (NSUInteger) numberOfRows;
```

Return Value
The number of columns.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setNumberOfRows: (page 45)

Declared In
QCCompositionPickerView.h

resetDefaultInputValues

Clears all previously set default values for composition input parameters.

```
- (void) resetDefaultInputValues
```

Discussion
This method resets the defaults that were set with the method `setDefaultValue:forInputKey:` (page 43).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionPickerView.h`

selectedComposition

Returns the composition that is currently selected in the composition picker view.

```
- (QCComposition*) selectedComposition
```

Return Value
A `QCComposition` object, or `nil` if a composition is not selected.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setSelectedComposition:` (page 45)

Declared In
`QCCompositionPickerView.h`

setAllowsEmptySelection:

Sets whether to allow an empty selection in the composition picker view.

```
- (void) setAllowsEmptySelection:(BOOL)flag
```

Parameters
flag
`YES` to allow an empty selection. The default value is `NO`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- allowsEmptySelection` (page 37)

Declared In
`QCCompositionPickerView.h`

setBackgroundColor:

Sets the background color for the composition picker view.

```
- (void) setBackgroundColor:(NSColor*)aColor;
```

Parameters

aColor

The color for the background.

Availability
Available in Mac OS X v10.5 and later.

See Also
- backgroundColor (page 38)

Declared In
QCCompositionPickerView.h

setCompositionAspectRatio:

Sets the aspect ratio used to display compositions in the composition picker view.

```
- (void) setCompositionAspectRatio:(NSSize)ratio
```

Parameters

ratio

An aspect ratio.

Availability
Available in Mac OS X v10.5 and later.

See Also
- compositionAspectRatio (page 38)

Declared In
QCCompositionPickerView.h

setCompositionsFromRepositoryWithProtocol:andAttributes:

Sets the compositions in the composition picker view to those that match the specified criteria.

```
- (void) setCompositionsFromRepositoryWithProtocol:(NSString*)protocol
    andAttributes:(NSDictionary*)attributes
```

Parameters

protocol

The protocols that you want compositions shown in the picker view to conform to. You can pass any of these protocols: QCCompositionProtocolAnimation, QCCompositionProtocolImageProducer, QCCompositionProtocolImageFilter, QCCompositionProtocolImageCompositor, QCCompositionProtocolImageTransition, and QCCompositionProtocolScreenSaverRSS.

attributes

A dictionary that contains the attributes, and their associated values, that you want compositions in the picker view to match. For example, you can pass: `QCCompositionAttributeNameKey`, `QCCompositionAttributeDescriptionKey`, `QCCompositionAttributeCopyrightKey`, `QCCompositionAttributeBuiltInKey`, and `QCCompositionAttributeTimeDependentKey`. Pass `nil` if you don't want to filter based on the attributes.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- compositions` (page 38)

Declared In
`QCCompositionPickerView.h`

setDefaultValue:forInputKey:

Sets the default value to use for a composition input parameter.

```
- (void) setDefaultValue:(id)value forInputKey:(NSString*)key
```

Parameters

value

This default value overrides any initial value existing for composition input parameters with this key. Pass `nil` to clear the default value.

key

The input parameter key whose default value you want to set.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- resetDefaultInputValues` (page 41)

Declared In
`QCCompositionPickerView.h`

setDelegate:

Sets the composition picker view delegate.

```
- (void) setDelegate:(id)delegate
```

Parameters

delegate

The delegate to set.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- delegate` (page 39)

Declared In
QCCompositionPickerView.h

setDrawsBackground:

Sets whether the composition picker view draws its background.

```
- (void) setDrawsBackground:(BOOL)flag;
```

Parameters

flag

The background drawing state. Pass YES if the composition picker view draws its background.

Availability
Available in Mac OS X v10.5 and later.

See Also
- drawsBackground (page 39)

Declared In
QCCompositionPickerView.h

setMaxAnimationFrameRate:

Sets the maximum frame rate for animating compositions.

```
- (void) setMaxAnimationFrameRate:(float)maxFPS
```

Parameters

maxFPS

A frame rate in frames per second. Pass 0.0 to specify no limit to the maximum value.

Availability
Available in Mac OS X v10.5 and later.

See Also
- maxAnimationFrameRate (page 40)

Declared In
QCCompositionPickerView.h

setNumberOfColumns:

Sets the number of columns in the composition picker view.

```
- (void) setNumberOfColumns:(NSUInteger)columns;
```

Parameters

columns

The number of columns.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `numberOfColumns` (page 40)

Declared In
`QCCompositionPickerView.h`

setNumberOfRows:

Sets the number of rows in the composition picker view.

```
- (void) setNumberOfRows:(NSUInteger)rows;
```

Parameters

columns

The number of rows.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `numberOfRows` (page 40)

Declared In
`QCCompositionPickerView.h`

setSelectedComposition:

Sets a composition as selected in the composition picker view.

```
- (void) setSelectedComposition:(QCComposition*)composition
```

Parameters

composition

The composition to select. Pass `nil` if you don't want to select a composition. The behavior is undefined if you pass a composition that is not in the list of compositions that are currently in the composition picker view.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `selectedComposition` (page 41)

Declared In
`QCCompositionPickerView.h`

setShowsCompositionNames:

Enables the display of composition names in the composition picker view.

```
- (void) setShowsCompositionNames:(BOOL)flag
```

Parameters

flag

YES specifies to show compositions name. The default value is NO.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCCompositionPickerView.h

showsCompositionNames

Retrieves whether composition names can be shown in the composition picker view.

```
- (BOOL) showsCompositionNames
```

Return Value
YES if the display of names is enabled; otherwise NO.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCCompositionPickerView.h

startAnimation:

Starts animating the composition in the composition picker view.

```
- (void) startAnimation:(id)sender
```

Parameters

sender

The object initiating the animation.

Availability
Available in Mac OS X v10.5 and later.

See Also
- stopAnimation: (page 46)
- isAnimating (page 39)

Declared In
QCCompositionPickerView.h

stopAnimation:

Stops animating the composition that is currently animating in the composition picker view.

```
- (void) stopAnimation:(id)sender
```

Parameters

sender

The object stopping the animation.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `startAnimation:` (page 46)
- `isAnimating` (page 39)

Declared In
`QCCompositionPickerView.h`

Notifications

QCCompositionPickerViewDidSelectCompositionNotification

Posted when the user selects a composition in the picker view.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionPickerView.h`

QCCompositionRepository Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCCompositionRepository.h
Companion guide	Quartz Composer Programming Guide
Related sample code	Quartz Composer SlideShow

Overview

The `QCCompositionRepository` class represents a system-wide centralized repository of built-in and installed Quartz Composer compositions (`/Library/Compositions` and `~/Library/Compositions`). The `QCCompositionRepository` class cannot be subclassed.

Compositions in the repository are represented by the `QCComposition` class. You can use the methods of the `QCCompositionRepository` class to fetch all compositions or only those that meet specific criteria.

Tasks

Getting the Composition Repository

`+ sharedCompositionRepository` (page 50)
Returns the shared instance of the composition repository.

Fetching Compositions

`- compositionWithIdentifier:` (page 51)
Returns the composition that corresponds to the identifier.

`- compositionsWithProtocols:andAttributes:` (page 50)
Returns an array of compositions that match a set of criteria.

- `allCompositions` (page 50)
  Returns an array that contains all compositions currently in the composition repository.

Class Methods

sharedCompositionRepository

Returns the shared instance of the composition repository.

```
+ (QCCompositionRepository*) sharedCompositionRepository
```

Return Value
The shared instance of `QCCompositionRepository`.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
Quartz Composer SlideShow

Declared In
`QCCompositionRepository.h`

Instance Methods

allCompositions

Returns an array that contains all compositions currently in the composition repository.

```
- (NSArray*) allCompositions
```

Return Value
An array of `QCComposition` objects.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `compositionWithIdentifier:` (page 51)
- `compositionsWithProtocols:andAttributes:` (page 50)

Declared In
`QCCompositionRepository.h`

compositionsWithProtocols:andAttributes:

Returns an array of compositions that match a set of criteria.

```
- (NSArray*) compositionsWithProtocols:(NSArray*)protocols
    andAttributes:(NSDictionary*)attributes
```

Parameters

protocols

The protocols that you want compositions to conform to. Pass `nil` if you don't want to filter based on the protocol. You can pass any of these protocols: `QCCompositionProtocolAnimation`, `QCCompositionProtocolImageProducer`, `QCCompositionProtocolImageFilter`, `QCCompositionProtocolImageCompositor`, `QCCompositionProtocolImageTransition`, and `QCCompositionProtocolScreenSaverRSS`.

attributes

A dictionary that contains the attributes, and their associated values, that you want compositions to match. Pass `nil` if you don't want to filter based on the attributes. For example, you can pass any of these attributes: `QCCompositionAttributeNameKey`, `QCCompositionAttributeDescriptionKey`, `QCCompositionAttributeCopyrightKey`, `QCCompositionAttributeBuiltInKey`, and `QCCompositionAttributeTimeDependentKey`.

Return Value
An array of `QCComposition` objects that meet the supplied criteria.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `compositionWithIdentifier:` (page 51)
- `allCompositions` (page 50)

Related Sample Code
Quartz Composer SlideShow

Declared In
`QCCompositionRepository.h`

compositionWithIdentifier:

Returns the composition that corresponds to the identifier.

```
- (QCComposition*) compositionWithIdentifier:(NSString*)identifier
```

Parameters

identifier

A string that uniquely identifies the composition to retrieve.

Return Value
The composition identified by the provided string, or `nil` if there is no composition with that identifier in the composition repository.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `compositionsWithProtocols:andAttributes:` (page 50)
- `allCompositions` (page 50)

Declared In
`QCCompositionRepository.h`

Notifications

QCCompositionRepositoryDidUpdateNotification

Posted whenever the list of compositions in the composition repository is updated.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionRepository.h`

QCPlugIn Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCPlugIn.h
Companion guides	Quartz Composer Custom Patch Programming Guide Quartz Composer Programming Guide
Related sample code	Quartz Composer ImageExporter

Overview

The `QCPlugIn` class provides the base class to subclass for writing custom Quartz Composer patches. You implement a custom patch by subclassing `QCPlugIn`, overriding the appropriate methods, packaging the code as an `NSBundle` object, and installing the bundle in the appropriate location. A bundle can contain more than one subclass of `QCPlugIn`, allowing you to provide a suite of custom patches in one bundle. *Quartz Composer Custom Patch Programming Guide* provides detailed instructions on how to create and package a custom patch. *QCPlugIn Class Reference* supplements the information in the programming guide.

The methods related to the executing the custom patch (called when the Quartz Composer engine is rendering) are passed an opaque object that conforms to the `QCPlugInContext Protocol` protocol. This object represents the execution context of the `QCPlugIn` object. You should not retain the execution context or use it outside of the scope of the execution method that it is passed to.

Tasks

Defining the Characteristics of a Custom Patch

`+ executionMode` (page 57)
Returns the execution mode of the custom patch.

`+ timeMode` (page 59)
Returns the time mode for the custom patch.

Executing a Custom Patch

- `execute:atTime:withArguments:` (page 63)
  Performs the processing or rendering tasks appropriate for the custom patch.

Performing Custom Tasks During Execution

- `startExecution:` (page 66)
  Allows you to perform custom setup tasks before the Quartz Composer engine starts rendering.
- `enableExecution:` (page 63)
  Allows you to perform custom tasks when the execution of the `QCPlugIn` object is resumed.
- `disableExecution:` (page 62)
  Allows you to perform custom tasks when the execution of the `QCPlugIn` object is paused.
- `stopExecution:` (page 67)
  Allows you to perform custom tasks when the `QCPlugIn` object stops executing.

Defining Patch and Property Port Attributes

+ `attributes` (page 55)
  Returns a dictionary that contains strings for the user interface that describe the custom patch.
+ `attributesForPropertyPortWithKey:` (page 56)
  Returns a dictionary that contains strings for the user interface that describe the optional attributes for ports created from properties.

Defining Internal Settings

- `createViewController` (page 61)
  Creates and returns a view controller for the Settings pane of a custom patch.
+ `plugInKeys` (page 58)
  Returns the keys for the internal settings of a custom patch.

Supporting Saving and Retrieving Internal Settings

- `serializedValueForKey:` (page 65)
  Provides custom serialization for patch internal settings that do not comply to the `NSCoding` protocol.
- `setSerializedValue:forKey:` (page 65)
  Provides custom deserialization for patch internal settings that were previously serialized using the method `serializedValueForKey:` (page 65).

Adding Ports Dynamically

- `addInputPortWithType:forKey:withAttributes:` (page 60)
  Adds an input port of the specified type and associates a key and attributes with the port.

- `removeInputPortForKey:` (page 64)
  Removes the input port for a given key.
- `addOutputPortWithType:forKey:withAttributes:` (page 60)
  Adds an output port of the specified type and associates a key and attributes with the port.
- `removeOutputPortForKey:` (page 64)
  Removes the output port for a given key.

Getting and Setting Port Values

- `didValueForInputKeyChange:` (page 62)
  Returns whether the input port value changed since the last execution of the custom patch.
- `valueForInputKey:` (page 67)
  Returns the current value for an input port.
- `setValue:forOutputKey:` (page 66)
  Sets the value of an output port.

Loading Bundle and Custom Patches Manually

+ `loadPlugInAtPath:` (page 57)
  Loads a Quartz Composer plug-in bundle from the specified path.
+ `registerPlugInClass:` (page 59)
  Registers a `QCPlugIn` subclass.

Ordering Property Ports

+ `sortedPropertyPortKeys` (page 59)
  Returns and array of property port keys in the order you want them to appear in the user interface.

Class Methods

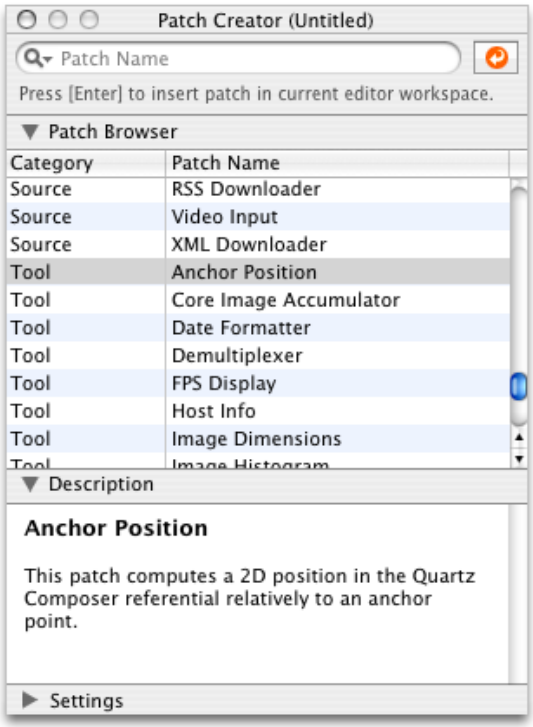
attributes

Returns a dictionary that contains strings for the user interface that describe the custom patch.

```
+ (NSDictionary*) attributes
```

Return Value
The dictionary can contain one or more of these keys along with the appropriate string: `QCPlugInAttributeNameKey` (page 68), `QCPlugInAttributeDescriptionKey` (page 68), and `QQCPlugInAttributeCopyrightKey` (page 68).

Discussion
It's recommended that you implement this method to enhance the experience of those who use your custom patch. The attribute name string that you provide is displayed in the Quartz Composer editor window when the custom patch name is selected in the Patch Creator (see figure). The attribute description key is displayed in the Information pane of the inspector for the custom patch.



Availability
Available in Mac OS X v10.5 and later.

See Also
`+ attributesForPropertyPortWithKey:` (page 56)

Related Sample Code
Quartz Composer ImageExporter

Declared In
`QCPlugIn.h`

attributesForPropertyPortWithKey:

Returns a dictionary that contains strings for the user interface that describe the optional attributes for ports created from properties.

```
+ (NSDictionary*) attributesForPropertyPortWithKey:(NSString*)key
```

Parameters

key

The name of the property.

Return Value
A dictionary that contains key-value pairs for the port's attributes. The keys must be one or more of the constants defined in "Input and Output Port Attributes" (page 68).

Discussion
It's recommended that you implement this method to enhance the experience of those who use your custom patch. The attributes appear in a help tag when the user hovers a pointer over the property port on your custom patch. At a minimum, you should provide a user-readable name for the port. It might also be helpful to provide default, minimum, and maximum values for the port.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ attributes` (page 55)

Declared In
`QCPlugIn.h`

executionMode

Returns the execution mode of the custom patch.

```
+ (QCPlugInExecutionMode) executionMode
```

Return Value
The execution mode of the custom patch. See "`Execution Modes`" (page 72) for the constants you can return.

Discussion
You must implement this method to define whether your custom patch is a provider, a processor, or a consumer.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
Quartz Composer ImageExporter

Declared In
`QCPlugIn.h`

loadPlugInAtPath:

Loads a Quartz Composer plug-in bundle from the specified path.

```
+ (BOOL) loadPlugInAtPath:(NSString*)path
```

Parameters
`path`
The location of the bundle.

Return Value
`YES` if successful.

Discussion
Call this method only if you need to load a plug-in bundle from a nonstandard location. Typically you don't need to call this method because Quartz Composer automatically loads bundles that you install in one of the following locations:

- `/Library/Graphics/Quartz Composer Plug-Ins`
- `~/Library/Graphics/Quartz Composer Plug-Ins`

This method does nothing if the bundle is already loaded. (This method does not load in all environments. Web Kit, for example, cannot load custom patches.)

The bundle can contain more than one `QCPlugIn` subclass. After the bundle is loaded, each `QCPlugIn` subclass appears as a patch in the Quartz Composer patch library.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

plugInKeys

Returns the keys for the internal settings of a custom patch.

```
+ (NSArray*) plugInKeys
```

Return Value
An array of keys used for key-value coding (KVC) of the internal settings.

Discussion
You must override this method if your patch provides a Settings pane. This keys are used for automatic serialization of the internal settings and are also used by the `QCPlugInViewController` instance for the Settings pane. The implementation is straightforward; the keys are strings that represent the instance variables used for the Settings pane. For example, the `plugInKeys` method for these instance variables:

```
@property(ivar, byref) NSColor * systemColor;
@property(ivar, byref) NSConfiguration * systemConfiguration;
```

are:

```
+ (NSArray*) plugInKeys
{
    return [NSArray arrayWithObjects: @"systemColor",
                                      @"systemConfiguration",
                                      nil];
}
```

Availability
Available in Mac OS X v10.5 and later.

See Also
`- createViewController` (page 61)

Declared In
QCPlugIn.h

registerPlugInClass:

Registers a QCPlugIn subclass.

```
+ (void) registerPlugInClass:(Class)aClass
```

Parameters
aClass
The QCPlugIn subclass.

Discussion
You call this method only if the code for your custom patch is mixed with your application code, and you plan only to use the custom patch from within your application.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCPlugIn.h

sortedPropertyPortKeys

Returns and array of property port keys in the order you want them to appear in the user interface.

```
+ (NSArray*) sortedPropertyPortKeys;
```

Return Value
The property port keys in the order you want them to appear in the user interface.

Discussion
Override this method to specify an optional ordering for property based ports in the user interface.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCPlugIn.h

timeMode

Returns the time mode for the custom patch.

```
+ (QCPlugInTimeMode) timeMode
```

Return Value
The time mode of the custom patch. See "Time Modes" (page 73) for the constants you can return.

Discussion
You must implement this method to define whether you custom patch depends on time, doesn't depend on time, or needs time to idle.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
Quartz Composer ImageExporter

Declared In
`QCPlugIn.h`

Instance Methods

addInputPortWithType:forKey:withAttributes:

Adds an input port of the specified type and associates a key and attributes with the port.

```
- (void) addInputPortWithType:(NSString*)type forKey:(NSString*)key
    withAttributes:(NSDictionary*)attributes
```

Parameters

type
The port type. See "Port Input and Output Types" (page 70).

key
The key to associate with the port.

attributes
A dictionary of attributes for the port. See "Input and Output Port Attributes" (page 68). Although the dictionary is optional, it's recommended that provide attributes to enhance the experience of those who use your custom patch. The attributes appear in a help tag when the user hovers a pointer over the property port on your custom patch. (See `attributesForPropertyPortWithKey:` (page 56).) Pass `nil` if you do not want to provide attributes.

Discussion
This method throws an exception if called from within the `execute:atTime:withArguments:` (page 63) method or if there's already an input or output port with that key.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- removeInputPortForKey:` (page 64)

Declared In
`QCPlugIn.h`

addOutputPortWithType:forKey:withAttributes:

Adds an output port of the specified type and associates a key and attributes with the port.

```
- (void) addOutputPortWithType:(NSString*)type forKey:(NSString*)key
    withAttributes:(NSDictionary*)attributes
```

Parameters

type

The port type. See "Port Input and Output Types" (page 70).

key

The key to associate with the port.

attributes

A dictionary of attributes for the port. See "Input and Output Port Attributes" (page 68). Although the dictionary is optional, it's recommended that provide attributes to enhance the experience of those who use your custom patch. The attributes appear in a help tag when the user hovers a pointer over the property port on your custom patch. (See `attributesForPropertyPortWithKey:` (page 56).) Pass `nil` if you do not want to provide attributes.

Discussion

This method throws an exception if called from within the `execute:atTime:withArguments:` (page 63) method or if there is already an output port with that key.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- removeOutputPortForKey:` (page 64)

Declared In

`QCPlugIn.h`

createViewController

Creates and returns a view controller for the Settings pane of a custom patch.

```
- (QCPlugInViewController*) createViewController
```

Return Value

A view controller for the custom patch. Quartz Composer releases the controller when it is no longer needed. If necessary, you can return a subclass of `QCPlugInViewController`, but this it not typically done.

Discussion

This extension to the `QCPlugInViewController` class provides user-interface support for the Settings pane of the inspector for a custom patch. You must override this method if your custom patch provides a Settings pane. The `QCPlugInViewController` object acts as a controller for Cocoa bindings between the custom patch instance (the model) and the `NSView` that contains the controls. It loads the nib file from the bundle.

The implementation is straightforward. You allocate a `QCPlugInViewController` object, initialize it, and provide the name of the nib file that contains the user interface for the Settings pane.

Note that this method follows the Core Foundation "create" rule. See the ownership policy in *Memory Management Programming Guide for Core Foundation*.

For example, if the nib file name that contains the settings pane is `MySettingsPane.nib`, the implementation is:

```
- (QCPlugInViewController *) createViewController
{
    return [[QCPlugInViewController alloc] initWithPlugIn:self
```

```
                                                viewNibName:@"MySettingsPane"];
}
```

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ plugInKeys` (page 58)

Declared In
`QCPlugInViewController.h`

didValueForInputKeyChange:

Returns whether the input port value changed since the last execution of the custom patch.

```
- (BOOL) didValueForInputKeyChange:(NSString*)key
```

Parameters

key

The key for the input port whose value you want to check.

Return Value
`YES` if the value on the input port changed since the last time the `execute:atTime:withArguments:` (page 63) method was called; always returns `NO` if called outside of the `execute:atTime:withArguments:` method.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- valueForInputKey:` (page 67)

Declared In
`QCPlugIn.h`

disableExecution:

Allows you to perform custom tasks when the execution of the `QCPlugIn` object is paused.

```
- (void) disableExecution:(id<QCPlugInContext>)context
```

Parameters

context

An opaque object , conforming to the `QCPlugInContext Protocol` protocol, that represents the execution context of the `QCPlugIn` object. Do not retain this object or use it outside of the scope of this method.

Discussion
The Quartz Composer engine calls this method when results are no longer being pulled from the custom patch. You can optionally override this execution method to perform custom tasks at that time.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `enableExecution:` (page 63)

Declared In
`QCPlugIn.h`

enableExecution:

Allows you to perform custom tasks when the execution of the `QCPlugIn` object is resumed.

```
- (void) enableExecution:(id<QCPlugInContext>)context
```

Parameters

context

An opaque object , conforming to the `QCPlugInContext Protocol` protocol, that represents the execution context of the `QCPlugIn` object. Do not retain this object or use it outside of the scope of this method.

Discussion
The Quartz Composer engine calls this method when results start to be pulled from the custom patch. You can optionally override this execution method to perform custom tasks at that time.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `disableExecution:` (page 62)

Declared In
`QCPlugIn.h`

execute:atTime:withArguments:

Performs the processing or rendering tasks appropriate for the custom patch.

```
- (BOOL) execute:(id<QCPlugInContext>)context atTime:(NSTimeInterval)time
    withArguments:(NSDictionary*)arguments
```

Parameters

context

An opaque object , conforming to the `QCPlugInContext Protocol` protocol, that represents the execution context of the `QCPlugIn` object. Do not retain this object or use it outside of the scope of this method.

time

The execution interval.

arguments

A dictionary of arguments that can be used during execution. See "Execution Arguments" (page 72).

Return Value
`NO` indicates the custom patch was not able to execute successfully. In this case, the Quartz Composer engine stops rendering the current frame.

Discussion
The Quartz Composer engine calls this method each time your custom patch needs to execute. You must implement this method. The method should perform whatever tasks are appropriate for the custom patch, such as:

- reading values from the input ports
- computing output values
- updating the values on the output ports
- rendering to the execution context

For example implementations of this method, see *Quartz Composer Custom Patch Programming Guide*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

removeInputPortForKey:

Removes the input port for a given key.

```
- (void) removeInputPortForKey:(NSString*)key
```

Parameters
key
The key associated with the port that you want to remove.

Discussion
This method throws an exception if from within the `execute:atTime:withArguments:` (page 63) method, if there is not an input port with that key, or if the port is created from a property.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- addInputPortWithType:forKey:withAttributes:` (page 60)

Declared In
`QCPlugIn.h`

removeOutputPortForKey:

Removes the output port for a given key.

```
- (void) removeOutputPortForKey:(NSString*)key
```

Parameters
key
The key associated with the port that you want to remove.

Discussion
This method throws an exception if called from within the `execute:atTime:withArguments:` (page 63) method, if there is not an output port with that key, or if the port is created from a property.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- addOutputPortWithType:forKey:withAttributes:` (page 60)

Declared In
`QCPlugIn.h`

serializedValueForKey:

Provides custom serialization for patch internal settings that do not comply to the `NSCoding` protocol.

```
- (id) serializedValueForKey:(NSString*)key
```

Parameters

key
The key for the value to retrieve.

Return Value
Either `nil` or a value that's compliant with property lists: `NSString`, `NSNumber`, `NSDate`, `NSData`, `NSArray`, or `NSDictionary`.

Discussion
If your patch has internal settings that do not conform to the `NSCoding` protocol, you must implement this method.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setSerializedValue:forKey:` (page 65)

Declared In
`QCPlugIn.h`

setSerializedValue:forKey:

Provides custom deserialization for patch internal settings that were previously serialized using the method `serializedValueForKey:` (page 65).

```
- (void) setSerializedValue:(id)serializedValue forKey:(NSString*)key
```

Parameters

serializedValue
The value to deserialize.

key
The key for the value to deserialize.

Discussion
If your patch has internal settings that do not conform to the `NSCoding` protocol, you must implement this method. After you deserialize the value, you need to call `[self set:value forKey:key]` to set the corresponding internal setting of the custom patch instance to the deserialized value.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

setValue:forOutputKey:

Sets the value of an output port.

```
- (BOOL) setValue:(id)value forOutputKey:(NSString*)key
```

Parameters

key

The key associated with the output port whose value you want to set.

Return Value
`YES` if successful; `NO` if called outside of the `execute:atTime:withArguments:` (page 63) method.

Discussion
You call this method from within your `execute:atTime:withArguments:` (page 63) method to set the output values of your custom patch.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `valueForInputKey:` (page 67)
- `didValueForInputKeyChange:` (page 62)

Declared In
`QCPlugIn.h`

startExecution:

Allows you to perform custom setup tasks before the Quartz Composer engine starts rendering.

```
- (BOOL) startExecution:(id<QCPlugInContext>)context
```

Parameters

context

An opaque object , conforming to the `QCPlugInContext Protocol` protocol, that represents the execution context of the `QCPlugIn` object. Do not retain this object or use it outside of the scope of this method.

Return Value
`NO` indicates a fatal error occurred and prevents the Quartz Composer engine from starting.

Discussion
The Quartz Composer engine calls this method when your custom patch starts to render. You can optionally override this execution method to perform setup tasks.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- stopExecution:` (page 67)

Declared In
`QCPlugIn.h`

stopExecution:

Allows you to perform custom tasks when the `QCPlugIn` object stops executing.

```
- (void) stopExecution:(id<QCPlugInContext>)context
```

Parameters

context

An opaque object , conforming to the `QCPlugInContext Protocol` protocol, that represents the execution context of the `QCPlugIn` object. Do not retain this object or use it outside of the scope of this method.

Discussion
The Quartz Composer engine calls this method when it stops executing. You can optionally override this execution method to perform cleanup tasks.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- startExecution:` (page 66)

Declared In
`QCPlugIn.h`

valueForInputKey:

Returns the current value for an input port.

```
- (id) valueForInputKey:(NSString*)key
```

Parameters

key

The key for the input port you want to check.

Return Value
The value associated with the key or `nil` if called outside of the `execute:atTime:withArguments:` (page 63) method.

Discussion
You call this method from within your `execute:atTime:withArguments:` (page 63) method to retrieve the input values of your custom patch.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setValue:forOutputKey:` (page 66)
- `didValueForInputKeyChange:` (page 62)

Declared In
`QCPlugIn.h`

Constants

Patch Attributes

Attributes for custom patches.

```
extern NSString* const QCPlugInAttributeNameKey;
extern NSString* const QCPlugInAttributeDescriptionKey;
extern NSString* const QCPlugInAttributeCopyrightKey;
```

Constants

`QCPlugInAttributeNameKey`
The key for the custom patch name. The associated value is an `NSString` object.
Available in Mac OS X v10.5 and later.
Declared in `QCPlugIn.h`.

`QCPlugInAttributeDescriptionKey`
The key for the custom patch description. The associated value is an `NSString` object.
Available in Mac OS X v10.5 and later.
Declared in `QCPlugIn.h`.

`QQCPlugInAttributeCopyrightKey`
The key for the custom patch copyright information. The associated value is an `NSString` object.

Declared In
`QCPlugIn.h`

Input and Output Port Attributes

Attributes for input and output ports.

```
extern NSString* const QCPortAttributeTypeKey;
extern NSString* const QCPortAttributeNameKey;
extern NSString* const QCPortAttributeDefaultValueKey;
extern NSString* const QCPortAttributeMinimumValueKey;
extern NSString* const QCPortAttributeMaximumValueKey;
extern NSString* const QCPortAttributeDefaultValueKey;
extern NSString* const QCPortAttributeMenuItemsKey;
```

Constants

`QCPortAttributeTypeKey`

The key for the port type. The associated value can be of any of the following constants: `QCPortTypeBoolean` (page 70), `QCPortTypeIndex` (page 70), `QCPortTypeNumber` (page 70), `QCPortTypeString` (page 71), `QCPortTypeColor` (page 71), `QCPortTypeImage` (page 71), or `QCPortTypeStructure` (page 71).

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortAttributeNameKey`

The key for the port name. The associated value is an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortAttributeMinimumValueKey`

The key for the port minimum value. The associated value is an `NSNumber` object that specifies the minimum numerical value accepted by the port.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortAttributeMaximumValueKey`

The key for the port maximum value. The associated value is an `NSNumber` object that specifies the maximum numerical value accepted by the port.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

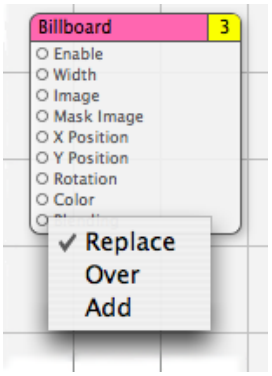
`QCPortAttributeDefaultValueKey`

The key for the port default value. You can use this key only for value ports (Boolean, Index, Number, Color and String).

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`QCPortAttributeMenuItemsKey`
The key for the menu items. The associated value is an array of strings that are displayed in the user interface as a pop-up menu when the user double-clicks a port, as shown for the Blending input port



of the Billboard patch. You can use this key only for an index port.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

Declared In
`QCPlugIn.h`

Port Input and Output Types

Data types for input and output ports.

```
extern NSString* const QCPortTypeBoolean;
extern NSString* const QCPortTypeIndex;
extern NSString* const QCPortTypeNumber;
extern NSString* const QCPortTypeString;
extern NSString* const QCPortTypeColor;
extern NSString* const QCPortTypeImage;
extern NSString* const QCPortTypeStructure;
```

Constants
`QCPortTypeBoolean`
The port type for a Boolean value. The associated value can be an `NSNumber` object or any object that responds to the `-intValue`, `-floatValue`, or `-doubleValue` methods.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortTypeIndex`
The port type for an index value. The associated value can be an `NSNumber` object or any object that responds to the `-intValue`, `-floatValue`, or `-doubleValue` methods.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortTypeNumber`
The port type for a number value. The associated value can be an `NSNumber` object or any object that responds to the `-intValue`, `-floatValue`, or `-doubleValue` methods.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortTypeString`
The port type for a string. The associated value can be an `NSString` object or any object that responds to the `-stringValue` or `-description` methods.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortTypeColor`
The port type for a color value. The associated value must be an `NSColor` object.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortTypeImage`
The port type for an image. The associated value can be an `NSImage` object or a `CIImage` object.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

`QCPortTypeStructure`
The port type for an array, dictionary, or other structure, such as an `NSArray` or `NSDictionary` object.

Available in Mac OS X v10.4 and later.

Declared in `QCPlugIn.h`.

Declared In
`QCPlugIn.h`

Pixel Formats

Supported image pixel formats.

```
extern NSString* const QCPlugInPixelFormatARGB8;
extern NSString* const QCPlugInPixelFormatBGRA8;
extern NSString* const QCPlugInPixelFormatRGBAf;
extern NSString* const QCPlugInPixelFormatI8;
extern NSString* const QCPlugInPixelFormatIf;
```

Constants
`QCPlugInPixelFormatARGB8`
An ARGB8 format. The alpha component is stored in the most significant bits of each pixel. Each pixel component is 8 bits. For best performance, use this format on PowerPC-based Macintosh computers, as it represents of the order of the data in memory.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`QCPlugInPixelFormatBGRA8`
A BGRA8 format. The alpha component is stored in the least significant bits of each pixel. Each pixel component is 8 bits. For best performance, use this format on Intel-PC-based Macintosh computers, as it represents of the order of the data in memory.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`QCPlugInPixelFormatRGBAf`
An RGBAf format. Pixel components are represented as floating-point values.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`QCPlugInPixelFormatI8`
An I8 format. Intensity information is represented as an 8-bit value.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`QCPlugInPixelFormatIf`
An If format. Intensity information is represented as a floating-point value.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

Declared In
`QCPlugIn.h`

Execution Arguments

Arguments to the method `execute:atTime:withArguments:` (page 63).

```
extern NSString* const QCPlugInExecutionArgumentEventKey;
extern NSString* const QCPlugInExecutionArgumentMouseLocationKey;
```

Constants
`QCPlugInExecutionArgumentEventKey`
The current `NSEvent` if available.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`QCPlugInExecutionArgumentMouseLocationKey`
The current location of the mouse (as an `NSPoint` object stored in an `NSValue` object) in normalized coordinates relative to the OpenGL context viewport ([0,1]x[0,1] with the origin (`0,0`) at the lower-left corner).

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

Declared In
`QCPlugIn.h`

Execution Modes

Execution modes for custom patches.

```
typedef enum {
    kQCPlugInExecutionModeProvider = 1,
    kQCPlugInExecutionModeProcessor,
    kQCPlugInExecutionModeConsumer
} QCPlugInExecutionMode;
```

Constants

`kQCPlugInExecutionModeProvider`

A provider execution mode. The custom patch executes on demand—that is, whenever data is requested of it, but at most once per frame.

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`kQCPlugInExecutionModeProcessor`

A processor execution mode. The custom patch executes whenever its inputs change or if the time change (assuming it's time-dependent).

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`kQCPlugInExecutionModeConsumer`

A consumer execution mode. The custom patch always executes assuming the value of its Enable input port is `true`. (The Enable port is automatically added by the system.)

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

Declared In

`QCPlugIn.h`

Time Modes

Time modes for custom patches.

```
typedef enum {
    kQCPlugInTimeModeNone = 0,
    kQCPlugInTimeModeIdle,
    kQCPlugInTimeModeTimeBase
} QCPlugInTimeMode;
```

Constants

`kQCPlugInTimeModeNone`

No time dependency. The custom patch does not depend on time at all. (It does not use the `time` parameter of the `execute:atTime:withArguments:` method.)

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`kQCPlugInTimeModeIdle`

An idle time dependency. The custom patch does not depend on time but needs the system to execute it periodically. For example if the custom patch connects to a piece of hardware, to ensure that it pulls data from the hardware, you would set the custom patch time dependency to idle time mode. This time mode is typically used with providers.]]

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

`kQCPlugInTimeModeTimeBase`

A time base dependency. The custom patch does depend on time explicitly and has a time base defined by the system. (It uses the `time` parameter of the `execute:atTime:withArguments:` method.)

Available in Mac OS X v10.5 and later.

Declared in `QCPlugIn.h`.

Declared In
`QCPlugIn.h`

QCPlugInViewController Class Reference

Inherits from	NSViewController : NSResponder : NSObject
Conforms to	NSCoding (NSViewController) NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	QuartzComposer/QCPlugInViewController.h
Companion guides	Quartz Composer Custom Patch Programming Guide Quartz Composer Programming Guide

Overview

The `QCPlugInViewController` class communicates (through Cocoa bindings) between a custom patch and the view used for the internal settings of the custom patch. Only custom patches that use internal settings exposed to the user need to use the `QCPlugInViewController` class.

You access the internal settings of a custom patch through key-value coding (KVC). All the KVC keys that represent the internal settings of the custom patch must be listed in its `plugInKeys` method.

The view controller for a custom patch expects

- the nib file `File's Owner` class set to the `QCPlugInViewController` class
- the view outlet connected to the view that contains the editing controls

The controls are bound to the `File's Owner` as the target and `plugIn.XXX` as the model key path, where `XXX` is the KVC key for a given internal setting of the custom patch instance.

Tasks

Creating a Controller

`- initWithPlugIn:viewNibName:` (page 76)
Creates and initializes a controller for the specified `QCPlugIn` object and nib file.

Getting the QCPlugIn Object

Instance Methods

initWithPlugIn:viewNibName:

Creates and initializes a controller for the specified QCPlugIn object and nib file.

```
- (id) initWithPlugIn:(QCPlugIn*)plugIn viewNibName:(NSString*)name
```

Parameters

plugIn

A QCPlugIn object that uses internal settings.

name

The name of the nib file that contains the view for the custom patch.

Return Value

A QCPlugInViewController object.

Availability

Available in Mac OS X v10.5 and later.

Declared In

QCPlugInViewController.h

plugIn

Returns the QCPlugIn object associated with the view controller for the custom patch.

```
- (QCPlugIn*) plugIn
```

Return Value

The QCPlugIn object associated with the view controller for the custom patch.

Availability

Available in Mac OS X v10.5 and later.

Declared In

QCPlugInViewController.h

QCRenderer Class Reference

Inherits from	NSObject
Conforms to	QCCompositionRenderer NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCRenderer.h
Availability	Available in Mac OS X v10.4 and later.
Related sample code	Quartz Composer Live DV Quartz Composer Matrix Quartz Composer Offline Rendering Quartz Composer SlideShow Quartz Composer Texture

Overview

A `QCRenderer` class is designed for low-level rendering of Quartz Composer compositions. This is the class to use if you want to be in charge of rendering a composition to a specific OpenGL context—either using the `NSOpenGLContext` class or a `CGLContextObj` object. `QCRenderer` also allows you to load, play, and control a composition.

To render a composition to a specific OpenGL context:

- Create an instance of `QCRenderer` using one of the initialization methods, such as `initWithOpenGLContext:pixelFormat:file:` (page 81).
- Render frames by calling the method `renderAtTime:arguments:` (page 82)
- If you use double buffering in OpenGL, you must swap the OpenGL buffers.
- Release the renderer with you no longer need it.

This code snippet shows how to implement these tasks:

```
NSOpenGLContext*     context = [myNSOpenGLView openGLContext];
NSOpenGLPixelFormat*  format = [myNSOpenGLView pixelFormat];
NSString*               path = @"/Users/MyName/MyComposition.qtz";
QCRenderer* myRenderer;
// Create a Quartz Composer renderer.
myRenderer = [[QCRenderer alloc] initWithOpenGLContext:context
                                           pixelFormat:format
                                                  file:path];
```

```
// Render the first 10 seconds of the composition with steps of 1/25s.
for(double t = 0.0; t <= 10.0; t += 1.0/25.0)
{
  [myRenderer renderAtTime:t arguments:nil];
  [context flushBuffer]; //Required on double-buffered contexts
}
// Clean up
 [renderer release];
```

Tasks

Creating and Initializing a Renderer

- `initWithComposition:colorSpace:` (page 81)
  Creates a renderer object with a composition object and a color space.
- `initWithOpenGLContext:pixelFormat:file:` (page 81)
  Creates a renderer object with an `NSOpenGLContext` object and a composition file.
- `initWithCGLContext:pixelFormat:colorSpace:composition:` (page 80)
  Creates a renderer object with a `CGLContextObj` object, a pixel format, a color space, and a composition object.
- `initOffScreenWithSize:colorSpace:composition:` (page 79)
  Creates an offscreen renderer of a given size with the provided color space and composition object.

Rendering a Composition

- `renderAtTime:arguments:` (page 82)
  Renders a frame of a composition at the specified time.

Getting the Composition Object

- `composition` (page 79)
  Returns the composition object associated with the renderer.

Taking Snapshot Images

- `snapshotImage` (page 83)
  Returns an `NSImage` object of the current image in the OpenGL context associated with the renderer.
- `createSnapshotImageOfType:` (page 79)
  Returns the current image in the OpenGL context associated with the renderer, as an image object of the provided image type.

Instance Methods

composition

Returns the composition object associated with the renderer.

```
- (QCComposition*) composition
```

Return Value
The composition object.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

createSnapshotImageOfType:

Returns the current image in the OpenGL context associated with the renderer, as an image object of the provided image type.

```
- (id) createSnapshotImageOfType:(NSString*)type
```

Parameters

type

A string that specifies any of the following image types: `NSBitmapImageRep`, `NSImage`, `CIImage`, `CGImage`, `CVOpenGLBuffer`, `CVPixelBuffer`.

Return Value
The snapshot image in the provided image type. You are responsible for releasing this object when you no longer need it.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

initOffScreenWithSize:colorSpace:composition:

Creates an offscreen renderer of a given size with the provided color space and composition object.

```
- (id) initOffScreenWithSize:(NSSize)size colorSpace:(CGColorSpaceRef)colorSpace
    composition:(QCComposition*)composition
```

Parameters

size

The size of the offscreen renderer.

colorSpace
A Quartz color space object. This must be an RGB color space. Pass `NULL` to use the default RGB color space. For more information on Quartz color spaces, see *Quartz 2D Programming Guide*.

composition
A `QCComposition` object.

Return Value
The initialized `QCRenderer` object or `nil` if initialization is not successful.

Discussion
This method creates an internal OpenGL context and pixel buffer. Because offscreen rendering is performed on the GPU, the maximum rendering size is limited to the GPU capacity. On typical hardware, the limit is at least 2048 by 2048, but is often 4096 by 4096. The available VRAM affects performance.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

initWithCGLContext:pixelFormat:colorSpace:composition:

Creates a renderer object with a `CGLContextObj` object, a pixel format, a color space, and a composition object.

```
- (id) initWithCGLContext:(CGLContextObj)context
    pixelFormat:(CGLPixelFormatObj)format colorSpace:(CGColorSpaceRef)colorSpace
    composition:(QCComposition*)composition;
```

Parameters

context
A `CGLContextObj` object. The object that you supply must have both a color and a depth buffer.

format
A `CGLPixelFormatObj` object.

colorSpace
A Quartz color space object. This must be an RGB color space. Pass `NULL` to use the default RGB color space. For more information on Quartz color spaces, see *Quartz 2D Programming Guide*.

composition
A `QCComposition` object.

Return Value
The initialized `QCRenderer` object or `nil` if initialization is not successful.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
Quartz Composer SlideShow

Declared In
`QCRenderer.h`

initWithComposition:colorSpace:

Creates a renderer object with a composition object and a color space.

```
- (id) initWithComposition:(QCComposition*)composition
    colorSpace:(CGColorSpaceRef)colorSpace;
```

Parameters

composition

A `QCComposition` object. The composition must not contain any consumer patches. That is, the composition can receive data, process it, and produce output values, but it cannot perform any rendering.

colorSpace

A Quartz color space object. This must be an RGB color space. Pass `NULL` to use the default RGB color space. The color space is used only for the images produced by the output image ports of the composition. For more information on Quartz color spaces, see *Quartz 2D Programming Guide*.

Return Value
The initialized `QCRenderer` object or `nil` if initialization is not successful.

Discussion
Note that `snapshotImage` (page 83) and `createSnapshotImageOfType:` (page 79) always returns `nil` on such `QCRenderer` instances.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

initWithOpenGLContext:pixelFormat:file:

Creates a renderer object with an `NSOpenGLContext` object and a composition file.

```
- (id)initWithOpenGLContext:(NSOpenGLContext *)context
    pixelFormat:(NSOpenGLPixelFormat *)format file:(NSString *)path
```

Parameters

context

An `NSOpenGLContext` object. The object that you supply must have both a color and a depth buffer.

format

An `NSOpenGLPixelFormat` object.

path

A string that specifies the location of a composition(`.qtz`) file.

Return Value
An initialized `QCRenderer` object or `nil` if initialization is not successful.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
Quartz Composer Live DV
Quartz Composer Matrix

Quartz Composer Offline Rendering
Quartz Composer SlideShow
Quartz Composer Texture

Declared In
`QCRenderer.h`

renderAtTime:arguments:

Renders a frame of a composition at the specified time.

```
- (BOOL)renderAtTime:(NSTimeInterval)time arguments:(NSDictionary *)arguments
```

Parameters

time

The time, in seconds, at which to render a composition frame. The time must be a positive value or zero.

arguments

An optional dictionary that can have any of the entries defined in "Rendering Arguments" (page 83).

Return Value
`YES` if successful.

Discussion
You need to call this method each time you want to render a frame of the composition.

All OpenGL states are preserved *except* the following:

- States defined by `GL_CURRENT_BIT`
- Textures on each unit and the environment mode
- Matrix mode

If you are using double buffers, keep in mind that the `renderAtTime:arguments:` method does not swap the front and back buffers of the OpenGL context. You must perform the swap yourself by calling the OpenGL command `flushBuffer` on the context associated with the renderer.

If you are interleaving OpenGL code with rendering of a composition, make sure that the OpenGL context is current. If you are using the `NSOpenGLContext` class, call the `makeCurrentContext` method prior to rendering. If you are using the CGL API, call the function `CGLSetCurrentContext`.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
Quartz Composer Texture

Declared In
`QCRenderer.h`

snapshotImage

Returns an `NSImage` object of the current image in the OpenGL context associated with the renderer.

```
- (NSImage*) snapshotImage
```

Return Value
The snapshot image.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

Constants

Rendering Arguments

Arguments that you can pass to the `renderAtTime:arguments:` (page 82) method.

```
extern NSString* const QCRendererEventKey;
extern NSString* const QCRendererMouseLocationKey;
```

Constants

`QCRendererEventKey`

A key for a renderer event. The associated value is an `NSEvent` object.

Available in Mac OS X v10.4 and later.

Declared in `QCRenderer.h`.

`QCRendererMouseLocationKey`

A key for the mouse location. The associated value is an `NSPoint` object stored in an `NSValue` object. The mouse location is in normalized coordinates relative to the OpenGL context viewport (`[0,1]x[0,1]` with the origin `(0,0)` at the lower-left corner).

Available in Mac OS X v10.4 and later.

Declared in `QCRenderer.h`.

Declared In
`QCRenderer.h`

QCView Class Reference

Inherits from	NSView : NSResponder : NSObject
Conforms to	QCCompositionRenderer NSAnimatablePropertyContainer (NSView) NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCView.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Quartz Composer Programming Guide
Related sample code	iChatTheater QCCocoaComponent Quartz Composer WWDC 2005 TextEdit TrackBall

Overview

The `QCView` class is a custom `NSView` class that loads, plays, and controls Quartz Composer compositions. It is an autonomous view that is driven by an internal timer running on the main thread.

The view can be set to render a composition automatically when it is placed onscreen. The view stops rendering when it is placed offscreen. When not rendering, the view is filled with the current erase color. The rendered composition automatically synchronizes to the vertical retrace of the monitor.

When you archive a `QCView` object, it saves the composition that's loaded at the time the view is archived.

If you want to perform custom operations while a composition is rendering such as setting input parameters or drawing OpenGL content, you need to subclass `QCView` and implement the `renderAtTime:arguments:` (page 93) method.

Tasks

Performing Custom Operations During Rendering

- `renderAtTime:arguments:` (page 93)
  Overrides to perform your custom operations prior to or after rendering a frame of a composition.

Loading a Composition

- `loadCompositionFromFile:` (page 91)
  Loads the composition file located at the specified path.
- `loadComposition:` (page 90)
  Loads a `QCComposition` object into the view.
- `loadedComposition` (page 91)
  Returns the composition loaded in the view.
- `unloadComposition` (page 100)
  Unloads the composition from the view.

Managing the Erase Color

- `erase` (page 88)
  Clears the view using the current erase color.
- `eraseColor` (page 89)
  Retrieves the current color used to erase the view.
- `setEraseColor:` (page 96)
  Sets the color used to erase the view.

Setting and Getting Event Masks

- `eventForwardingMask` (page 89)
  Retrieves the mask used to filter which types of events are forwarded from the view to the composition during rendering.
- `setEventForwardingMask:` (page 96)
  Sets the mask used to filter which types of events are forwarded from the view to the composition during rendering.

Setting and Getting the Maximum Frame Rate

- `maxRenderingFrameRate` (page 91)
  Returns the maximum frame rate for rendering.
- `setMaxRenderingFrameRate:` (page 97)
  Sets the maximum rendering frame rate.

Managing Rendering

- `startRendering` (page 99)
  Starts rendering the composition that is in the view.
- `isRendering` (page 90)
  Checks whether a composition is rendering in the view.
- `autostartsRendering` (page 88)
  Checks whether the view is set to start rendering automatically.
- `setAutostartsRendering:` (page 96)
  Sets whether the composition that is in the view starts rendering automatically when the view is put on the screen.
- `stopRendering` (page 99)
  Stops rendering the composition that is in the view.
- `pauseRendering` (page 93)
  Pauses rendering in the view.
- `isPausedRendering` (page 89)
  Returns whether or not the rendering in the view is paused.
- `resumeRendering` (page 95)
  Resumes rendering a paused composition.

Using Interface Builder

- `play:` (page 93)
  Plays or pauses a composition in a view.
- `start:` (page 98)
  Starts rendering a composition in a view.
- `stop:` (page 99)
  Stops rendering a composition in a view.

Taking Snapshot Images

- `snapshotImage` (page 98)
  Returns an `NSImage` object of the current image in the view.
- `createSnapshotImageOfType:` (page 88)
  Returns the current image in the view as an image object of the provided image type.

Working With OpenGL

- `openGLContext` (page 92)
  Returns the OpenGL context used by the view.
- `openGLPixelFormat` (page 92)
  Returns the OpenGL pixel format used by the view.

Instance Methods

autostartsRendering

Checks whether the view is set to start rendering automatically.

```
- (BOOL)autostartsRendering
```

Return Value
Returns `YES` if the view is set to start rendering automatically when the view is put on screen.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setAutostartsRendering:` (page 96)

Declared In
`QCView.h`

createSnapshotImageOfType:

Returns the current image in the view as an image object of the provided image type.

```
- (id) createSnapshotImageOfType:(NSString*)type
```

Parameters

type

A string that specifies any of the following image types: `NSBitmapImageRep`, `NSImage`, `CIImage`, `CGImage`, `CVOpenGLBuffer`, `CVPixelBuffer`.

Return Value
The snapshot image in the provided image type. You are responsible for releasing this object when you no longer need it.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- snapshotImage` (page 98)

Declared In
`QCView.h`

erase

Clears the view using the current erase color.

```
- (void)erase
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `eraseColor` (page 89)

Declared In
`QCView.h`

eraseColor

Retrieves the current color used to erase the view.

```
- (NSColor *)eraseColor
```

Return Value
The color object previously set using the `setEraseColor:` (page 96) method.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `erase` (page 88)

Declared In
`QCView.h`

eventForwardingMask

Retrieves the mask used to filter which types of events are forwarded from the view to the composition during rendering.

```
- (NSUInteger)eventForwardingMask
```

Return Value
The event filtering mask.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setEventForwardingMask:` (page 96)

Declared In
`QCView.h`

isPausedRendering

Returns whether or not the rendering in the view is paused.

```
- (BOOL) isPausedRendering;
```

Return Value
`YES` if the rendering is paused; otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `pauseRendering` (page 93)
- `resumeRendering` (page 95)

Declared In
`QCView.h`

isRendering

Checks whether a composition is rendering in the view.

```
- (BOOL)isRendering
```

Return Value
Returns `YES` if a composition is rendering in the view; `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCView.h`

loadComposition:

Loads a `QCComposition` object into the view.

```
- (BOOL) loadComposition:(QCComposition*)composition
```

Parameters
composition
The `QCComposition` object to load.

Return Value
`YES` if successful; otherwise `NO`. If unsuccessful, any composition that's already loaded in the view remains loaded.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `loadCompositionFromFile:` (page 91)
- `unloadComposition` (page 100)
- `loadedComposition` (page 91)

Declared In
`QCView.h`

loadCompositionFromFile:

Loads the composition file located at the specified path.

```
- (BOOL)loadCompositionFromFile:(NSString *)path
```

Parameters

path

A string that specifies the location of a Quartz Composer composition file.

Return Value

If unsuccessful, returns `NO`; any composition that's already loaded in the view remains loaded.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `loadComposition:` (page 90)
- `unloadComposition` (page 100)
- `loadedComposition` (page 91)

Declared In

`QCView.h`

loadedComposition

Returns the composition loaded in the view.

```
- (QCComposition*) loadedComposition
```

Return Value

The composition loaded in the view; otherwise `nil`.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `loadCompositionFromFile:` (page 91)
- `loadComposition:` (page 90)
- `unloadComposition` (page 100)

Declared In

`QCView.h`

maxRenderingFrameRate

Returns the maximum frame rate for rendering.

```
- (float)maxRenderingFrameRate
```

Return Value

The maximum frame rate for rendering. A value of `0.0` specifies that there is no limit.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setMaxRenderingFrameRate:` (page 97)

Declared In
`QCView.h`

openGLContext

Returns the OpenGL context used by the view.

```
- (NSOpenGLContext*) openGLContext
```

Return Value
An `NSOpenGLContext` object.

Discussion
This context as a read-only object . Do not attempt to change any of its settings. If you subclass `QCView` so that you can perform custom OpenGL drawing, you'll need to use this method to retrieve the view's OpenGL context.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- renderAtTime:arguments:` (page 93)

Declared In
`QCView.h`

openGLPixelFormat

Returns the OpenGL pixel format used by the view.

```
- (NSOpenGLPixelFormat*) openGLPixelFormat
```

Return Value
An `NSOpenGLPixelFormat` object.

Discussion
This pixel format as a read-only object. Do not attempt to change any of its settings.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCView.h`

pauseRendering

Pauses rendering in the view.

```
- (void) pauseRendering
```

Discussion
You can nest calls to this method.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `resumeRendering` (page 95)
- `isPausedRendering` (page 89)

Declared In
`QCView.h`

play:

Plays or pauses a composition in a view.

```
- (IBAction) play:(id)sender
```

Parameters

sender

The object (such as a button or menu item) sending the message to play the composition. You need to connect the object in the interface to the action.

Return Value
The message sent to the target.

Discussion
This method starts rendering a composition if it is not already rendering, pauses a composition that is rendering, or resumes rendering for a composition whose rendering is paused. The method is invoked when the user clicks a button or issues a command from some other user interface element, such as a menu.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `stop:` (page 99)

Declared In
`QCView.h`

renderAtTime:arguments:

Overrides to perform your custom operations prior to or after rendering a frame of a composition.

```
- (BOOL) renderAtTime:(NSTimeInterval)time arguments:(NSDictionary*)arguments
```

Parameters

time

The rendering time, in seconds, of the composition frame.

arguments

An optional dictionary that can contain `QCRendererEventKey` or `QCRendererMouseLocationKey` and the associated values. (See *QCRenderer Class Reference* or more information.)

Return Value

`NO` if your custom rendering fails, otherwise, `YES`.

Discussion

Do not call this method directly. You override this method only for subclasses of the `QCView` class and only if you want to perform custom operations or OpenGL rendering before and/or after Quartz Composer renders a frame of the composition.

The most common reasons to override this method are to:

- synchronize communication with the composition. For example, you might want to set input parameters of the composition. By overriding this method, you can set parameters only when necessary and only at a specific time.
- underlay or overlay custom OpenGL rendering.

To synchronize communication between a composition and another part of the application, the implementation looks similar to the following:

```
- (BOOL) renderAtTime:(NSTimeInterval)time
            arguments:(NSDictionary*)arguments
{
  // Your code to computer the value of myParameterValue
  [self setValue:myParameterValue forInputKey:@"myInput"];

  BOOL success = [super renderAtTime:time arguments:arguments];

  id result = [self valueForOutputKey:@"myOutput"];
  //Your code to perform some operation on the result

  return success;
}
```

To perform OpenGL drawing in a `QCView` object, follow these guidelines:

- Use the OpenGL context of the `QCView` object to do drawing. You can retrieve the OpenGL context by calling `[self openGLContext]`. Note that this context won't necessarily be set as the current OpenGL context.
- Use CGL macros instead of managing the current OpenGL context yourself.

  OpenGL performs a global context and renderer lookup for each command it executes to ensure that all OpenGL commands are issued to the correct rendering context and renderer. There is significant overhead associated with these lookups that can measurably affect performance. CGL macros let you provide a local context variable and cache the current renderer in that variable. They are simple to use, taking only a few lines of code to set up.

- Save and restore all state changes except the ones that are part of `GL_CURRENT_BIT` (RGBA color, color index, normal vector, texture coordinates, and so forth).
- Check for OpenGL errors with `glGetError`.

Here's an example implementation of this method using OpenGL to draw an overlay:

```
#import <OpenGL/CGLMacro.h>  // Set up using macros

- (BOOL) renderAtTime:(NSTimeInterval)time
            arguments:(NSDictionary*)arguments
{
    BOOL success = [super renderAtTime:time arguments:arguments];

    // Use the OpenGL context of the view for drawing.
    CGLContextObj cgl_ctx = [[self openGLContext] CGLContextObj];

    // Save and set OpenGL states appropriately.
    glGetIntegerv(GL_MATRIX_MODE, &saveMode);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glRotatef(45.0, 0.0, 0.0, 1.0);

    // The code that performs OpenGL drawing goes here.
    //After drawing, restore original OpenGL states.
    glPopMatrix();
    glMatrixMode(saveMode);

    // Check for errors.
    glGetError();
    return success;
}
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCView.h`

resumeRendering

Resumes rendering a paused composition.

```
- (void) resumeRendering
```

Discussion
You can nest calls to this method.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `pauseRendering` (page 93)
- `isPausedRendering` (page 89)

Declared In
QCView.h

setAutostartsRendering:

Sets whether the composition that is in the view starts rendering automatically when the view is put on the screen.

```
- (void)setAutostartsRendering:(BOOL)flag
```

Parameters

flag
Pass YES to enable autostart mode; NO otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- autostartsRendering (page 88)

Declared In
QCView.h

setEraseColor:

Sets the color used to erase the view.

```
- (void)setEraseColor:(NSColor *)color
```

Parameters

color
A color object.

Availability
Available in Mac OS X v10.4 and later.

See Also
- erase (page 88)
- eraseColor (page 89)

Declared In
QCView.h

setEventForwardingMask:

Sets the mask used to filter which types of events are forwarded from the view to the composition during rendering.

```
- (void)setEventForwardingMask:(NSUInteger)mask
```

Parameters

mask

An event filtering mask. The mask can be a combination of any of the mask constants listed in Table 10-1 or the constant `NSAnyEventMask`.

Table 10-1 Events that can be forwarded to a composition

Event	Description
`NSLeftMouseDownMask`	The user pressed the left button.
`NSLeftMouseDraggedMask`	The user moved the mouse with the left button down.
`NSLeftMouseUpMask`	The user released the left button.
`NSRightMouseDownMask`	The user pressed the right button.
`NSRightMouseDraggedMask`	The user moved the mouse with the right button down.
`NSRightMouseUpMask`	The user released the right button.
`NSOtherMouseDownMask`	The user pressed the middle button, or some button other than the left or right button.
`NSOtherMouseDraggedMask`	The user moved the mouse with the middle button down, or some button other than the left or right button.
`NSOtherMouseUpMask`	The user released the middle button, or some button other than the left or right button.
`NSMouseMovedMask`	The user moved the mouse without holding down a mouse button.
`NSScrollWheelMask`	The user moved the mouse scroll wheel.
`NSKeyDownMask`	The user generated a character or characters by pressing a key.
`NSKeyUpMask`	The user released a key.
`NSFlagsChangedMask`	The user pressed or released a modifier key, or toggled the Caps Lock key.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `eventForwardingMask` (page 89)

Declared In
`QCView.h`

setMaxRenderingFrameRate:

Sets the maximum rendering frame rate.

```
- (void)setMaxRenderingFrameRate:(float)maxFPS
```

Parameters

maxFPS

The frame rate to set. Pass `0.0` to specify that there is no limit.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `maxRenderingFrameRate` (page 91)

Declared In
`QCView.h`

snapshotImage

Returns an `NSImage` object of the current image in the view.

```
- (NSImage*) snapshotImage
```

Return Value
The snapshot image.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `createSnapshotImageOfType:` (page 88)

Declared In
`QCView.h`

start:

Starts rendering a composition in a view.

```
- (IBAction)start:(id)sender
```

Parameters

sender

The object (such as a button or menu item) sending the message to start rendering. You need to connect the object in the interface to the action.

Return Value
The message sent to the target.

Discussion
The method is invoked when the user clicks a button or issues a command from some other user interface element, such as a menu. It is equivalent to the `startRendering` (page 99) method.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `stop:` (page 99)

Declared In
QCView.h

startRendering

Starts rendering the composition that is in the view.

```
- (BOOL)startRendering
```

Return Value
Returns NO if the composition fails to start rendering; YES otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- stopRendering (page 99)

Declared In
QCView.h

stop:

Stops rendering a composition in a view.

```
- (IBAction)stop:(id)sender
```

Parameters

sender

The object (such as a button or menu item) sending the message to stop rendering. You need to connect the object in the interface to the action.

Return Value
The message sent to the target.

Discussion
The method is invoked when the user clicks a button or issues a command from some other user interface element, such as a menu. It is equivalent to the stopRendering (page 99) method.

Availability
Available in Mac OS X v10.4 and later.

See Also
- start: (page 98)

Declared In
QCView.h

stopRendering

Stops rendering the composition that is in the view.

```
- (void)stopRendering
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `startRendering` (page 99)

Declared In
`QCView.h`

unloadComposition

Unloads the composition from the view.

```
- (void) unloadComposition;
```

Discussion
If necessary, this method calls `stopRendering` (page 99) prior to unloading the composition.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `loadCompositionFromFile:` (page 91)
- `loadComposition:` (page 90)
- `loadedComposition` (page 91)

Declared In
`QCView.h`

Notifications

QCViewDidStartRenderingNotification

Posted when the view starts rendering.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`QCView.h`

QCViewDidStopRenderingNotification

Posted when the view stops rendering.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`QCView.h`

Protocols

QCCompositionParameterViewDelegate Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCCompositionParameterView.h
Companion guide	Quartz Composer Programming Guide

Overview

The `QCCompositionParameterViewDelegate` informal protocol allows your application it define which parameters should be visible in a `QCCompositionParameterView` object.

Tasks

Responding to Composition Selections

`- compositionParameterView:shouldDisplayParameterWithKey:attributes:` (page 103)
Allows you to define which composition parameters are visible in the user interface when the composition parameter view refreshes.

Instance Methods

compositionParameterView:shouldDisplayParameterWithKey:attributes:

Allows you to define which composition parameters are visible in the user interface when the composition parameter view refreshes.

```
- (BOOL) compositionParameterView:(QCCompositionParameterView *)parameterView
    shouldDisplayParameterWithKey:(NSString *)portKey attributes:(NSDictionary
    *)portAttributes;
```

Parameters

parameterView
The composition parameter view in which the selection changed.

portKey
A key for one of the composition parameters, which is provided to you by the Quartz Composer engine.

portAttributes
A dictionary of the attributes that you want to display in the user interface.

Return Value
`YES` if port attributes should be displayed; `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionParameterView.h`

QCCompositionPickerViewDelegate Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCCompositionPickerView.h
Companion guide	Quartz Composer Programming Guide

Overview

The `QCCompositionPickerViewDelegate` informal protocol defines methods that allow your application to respond to changes in a composition picker view (a `QCCompositionPickerView` object).

Tasks

Responding to Composition Selections

`- compositionPickerView:didSelectComposition:` (page 105)
Performs custom tasks when the selected composition in the composition picker view changes.

Responding to Animation State Changes

`- compositionPickerViewDidStartAnimating:` (page 106)
Performs custom tasks when the composition picker view starts animating a composition.

`- compositionPickerViewWillStopAnimating:` (page 106)
Performs custom tasks when the composition picker view stops animating a composition.

Instance Methods

compositionPickerView:didSelectComposition:

Performs custom tasks when the selected composition in the composition picker view changes.

```
- (void) compositionPickerView:(QCCompositionPickerView*)pickerView
    didSelectComposition:(QCComposition*)composition
```

Parameters

pickerView

The composition picker view in which the selection changed.

composition

The selected composition or `nil` if the previously selected composition is no longer selected.

Discussion

Quartz Composer invokes this method when the selected composition in the composition picker view changes. Implement this method if you want to perform custom tasks at that time.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`QCCompositionPickerView.h`

compositionPickerViewDidStartAnimating:

Performs custom tasks when the composition picker view starts animating a composition.

```
- (void) compositionPickerViewDidStartAnimating:(QCCompositionPickerView*)pickerView
```

Parameters

pickerView

The composition picker view in which the composition started animating.

Discussion

Quartz Composer invokes this method when the composition picker view starts animating a composition. Implement this method if you want to perform custom tasks at that time.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`QCCompositionPickerView.h`

compositionPickerViewWillStopAnimating:

Performs custom tasks when the composition picker view stops animating a composition.

```
 (void) compositionPickerViewWillStopAnimating:(QCCompositionPickerView*)pickerView
```

Parameters

pickerView

The composition picker view in which the composition stopped animating.

Discussion

Quartz Composer invokes this method whenever the composition picker view stops animating a composition. Implement this method if you want to perform custom tasks at that time.

Availability

Available in Mac OS X v10.5 and later.

Declared In
`QCCompositionPickerView.h`

QCCompositionRenderer Protocol Reference

Adopted by	QCRenderer QCView QCCompositionLayer
Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCRenderer.h
Availability	Available in Mac OS X v10.5 and later.

Overview

The `QCRenderer` protocol defines the methods used to pass data to the input ports or retrieve data from the output ports of the root patch of a Quartz Composer composition. This protocol is adopted by the `QCRenderer`, `QCView`, and `QCCompositionLayer` classes.

Tasks

Passing and Retrieving Values From a Composition

- `setValue:forInputKey:` (page 112) *required method*
  Sets the value for an input port of a composition. (required)
- `valueForInputKey:` (page 113) *required method*
  Returns the value for an input port of a composition. (required)
- `valueForOutputKey:` (page 114) *required method*
  Returns the value for an output port of a composition. (required)
- `valueForOutputKey:ofType:` (page 114) *required method*
  Returns the current value on an output port (identified by its key) of the root patch of the composition. (required)

Getting Input and Output Keys

- `inputKeys` (page 111) *required method*
  Returns an array that contains the keys that identify the input ports of the root patch of the composition. (required)

- `outputKeys` (page 111) *required method*
  Returns an array that contains the keys that identify the output ports of the root patch of the composition. (required)

Getting Attributes

- `attributes` (page 110) *required method*
  Returns the attributes of the composition associated with the renderer. (required)

Storing Arbitrary Information

- `userInfo` (page 113) *required method*
  Returns a mutable dictionary for storing arbitrary information. (required)

Saving and Restoring Input Values

- `propertyListFromInputValues` (page 111) *required method*
  Returns a property list object that represents the current values for all the input keys of the composition. (required)
- `setInputValuesWithPropertyList:` (page 112) *required method*
  Sets the values for the input keys of the composition from a previously saved property list. (required)

Instance Methods

attributes

Returns the attributes of the composition associated with the renderer. (required)

```
- (NSDictionary *)attributes
```

Return Value
A dictionary that contains the attributes that describe the composition, including the input and output ports of the root patch.

Discussion
The dictionary can define any of the attributes that are specified by the composition attribute keys. See `QCCompositionAttributeNameKey`, `QCCompositionAttributeDescriptionKey`, and `QCCompositionAttributeCopyrightKey`.

The dictionary can also contain dictionaries that correspond to the keys that identify the input and output ports of the root patch of the composition. See `QCPortAttributeTypeKey`, `QCPortAttributeNameKey`, `QCPortAttributeMinimumValueKey`, `QCPortAttributeMaximumValueKey`, and `QCPortAttributeMenuItemsKey` (page 70).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `inputKeys` (page 111)
- `outputKeys` (page 111)

Declared In
`QCRenderer.h`

inputKeys

Returns an array that contains the keys that identify the input ports of the root patch of the composition. (required)

```
- (NSArray *)inputKeys
```

Return Value
An array of keys associated with input ports.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `outputKeys` (page 111)

Declared In
`QCRenderer.h`

outputKeys

Returns an array that contains the keys that identify the output ports of the root patch of the composition. (required)

```
- (NSArray *)outputKeys
```

Return Value
An array of keys associated with input ports.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `inputKeys` (page 111)

Declared In
`QCRenderer.h`

propertyListFromInputValues

Returns a property list object that represents the current values for all the input keys of the composition. (required)

```
- (id) propertyListFromInputValues
```

Return Value
A property list object.

Discussion
This is a convenience method that allows you to easily save the set of input values on a composition. Typically, you store the set of values in application preferences.

Availability
Available in Mac OS X v10.5 and later.

See Also
`setInputValuesWithPropertyList:` (page 112)

Declared In
`QCRenderer.h`

setInputValuesWithPropertyList:

Sets the values for the input keys of the composition from a previously saved property list. (required)

```
- (void) setInputValuesWithPropertyList:(id)plist
```

Discussion
This is a convenience method that allows you to restore the set of input values that you obtained previously by calling the method `propertyListFromInputValues` (page 111). If the property list object does not define a value for an input key, or if the value is not of the proper type, Quartz Composer does not set a value for the corresponding input port.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

setValue:forInputKey:

Sets the value for an input port of a composition. (required)

```
- (BOOL)setValue:(id)value forInputKey:(NSString *)key
```

Parameters

value

The value to set for the input port. The input port must be at the root patch of the composition. The data type of the `value` argument must match the input port. See `QCPortAttributeTypeKey` (page 69) for the data types accepted by a particular port type.

key

The key associated with the input port of the composition. This method throws an exception if `key` is invalid.

Return Value
Returns `NO` if it cannot set the value.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `valueForInputKey:` (page 113)
- `valueForOutputKey:` (page 114)

Related Sample Code
TrackBall

Declared In
`QCRenderer.h`

userInfo

Returns a mutable dictionary for storing arbitrary information. (required)

```
- (NSMutableDictionary*) userInfo
```

Return Value
A mutable dictionary.

Discussion
The `userInfo` dictionary is shared—there is one per Quartz Composer context. In fact, it is the same dictionary as the one available for the plug-in execution context for instances of the `QCPlugIn` class.

When you add information to the dictionary, make sure that you use unique keys, such as `"com.myCompany.foo"`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCRenderer.h`

valueForInputKey:

Returns the value for an input port of a composition. (required)

```
- (id)valueForInputKey:(NSString *)key
```

Parameters

key

The key associated with an input port for the root patch of a composition. This method throws an exception if `key` is invalid.

Return Value
The value. The data type of returned value depends on the type of the input port. See `QCPortAttributeTypeKey` (page 69) for more information.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setValue:forInputKey:` (page 112)
- `valueForOutputKey:` (page 114)

Declared In
`QCRenderer.h`

valueForOutputKey:

Returns the value for an output port of a composition. (required)

```
- (id)valueForOutputKey:(NSString *)key
```

Parameters

key

The key associated with an output port for the root patch of a composition. This method throws an exception if `key` is invalid.

Return Value
The value. The data type of returned value depends on the type of the output port. See `QCPortAttributeTypeKey` (page 69) for more information.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setValue:forInputKey:` (page 112)
- `valueForInputKey:` (page 113)

Declared In
`QCRenderer.h`

valueForOutputKey:ofType:

Returns the current value on an output port (identified by its key) of the root patch of the composition. (required)

```
- (id) valueForOutputKey:(NSString*)key ofType:(NSString*)type
```

Parameters

key

The key associated with an output port for the root patch of a composition. This method throws an exception if `key` is invalid.

type

A string that specifies the class.

Return Value
The value.

Discussion
The value type depends on the type of the port type, as shown in the following table

Port type	Value type
Boolean, Index, or Number	`NSNumber` or any object that responds to the methods `integerValue`, `floatValue`, or `doubleValue`
String	`NSString` or any object that responds to the methods`stringValue` or `description`
Color	`NSColor`, `CIColor`, or `CGColor` object
Image	`NSImage`, `NSBitmapImageRep`, `CGImage` object, `CIImage`, `CVPixelBuffer` object, `CVOpenGLBuffer` object, `CVOpenGLTexture` object, or an opaque `QCImage` (that is, an optimized abstract image object only to be used with `setValue: forInputKey:` of another `<QCCompositionRenderer>`)
Structure	`NSArray` or `NSDictionary`

Availability
Available in Mac OS X v10.5 and later.

See Also
- `- setValue:forInputKey:` (page 112)
- `- valueForInputKey:` (page 113)

Declared In
`QCRenderer.h`

QCPlugInContext Protocol Reference

Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCPlugIn.h
Availability	Available in Mac OS X v10.5 and later.

Overview

The `QCPlugInContext` protocol defines methods that you use only from within the execution method (`execute:atTime:withArguments:` (page 63)) of a `QCPlugIn` object.

Tasks

Getting the OpenGL Context

- `CGLContextObj` (page 118) *required method*
  Returns the destination CGL context to use for OpenGL rendering from within the execution method. (required)

Logging Messages

- `logMessage:` (page 119) *required method*
  Writes a message to the Quartz Composer log. (required)

Getting Execution Context Information

- `userInfo` (page 122) *required method*
  Returns a mutable dictionary that contains information that can be shared between all instances of the `QCPlugIn` subclass, running in the same Quartz Composer context. (required)
- `bounds` (page 118) *required method*
  Returns the bounds of the rendering context. (required)
- `colorSpace` (page 119) *required method*
  Returns the color space used by the rendering context. (required)

Getting an Image Provider

- outputImageProviderFromBufferWithPixelFormat:pixelsWide:pixelsHigh:baseAddress:bytesPerRow:releaseCallback:releaseContext:colorSpace:shouldColorMatch: (page 120) *required method*

  Returns an image provider from a single memory buffer. (required)

- outputImageProviderFromTextureWithPixelFormat:pixelsWide:pixelsHigh:name:flipped:releaseCallback:releaseContext:colorSpace:shouldColorMatch: (page 121) *required method*

  Returns an image provider from an OpenGL texture. (required)

Instance Methods

bounds

Returns the bounds of the rendering context. (required)

```
- (NSRect) bounds
```

Return Value
The bounds of the rendering context expressed in Quartz Composer units.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

CGLContextObj

Returns the destination CGL context to use for OpenGL rendering from within the execution method. (required)

```
- (CGLContextObj) CGLContextObj
```

Return Value
The destination CGL context.

Discussion
To send commands to the OpenGL context:

- Use CGL macros instead of changing the current OpenGL context.
- Save and restore all OpenGL states except those defines by `GL_CURRENT_BIT` (vertex position, color, texture, and so on)

The following code shows how you'd use the method `CGLContextObj`:

```
// Set up using CGL macros.
#import <OpenGL/CGLMacro.h>

- (BOOL) execute:(id<QCPlugInContext>)context
              atTime:(NSTimeInterval)time
           withArguments:(NSDictionary *)arguments
```

```
{
    // Set the CGL context to a local variable.
    CGLContextObj cgl_ctx = [context CGLContextObj];
    if(cgl_ctx == NULL)
    return NO;

    // Save and set OpenGL states.
    // Put your OpenGL code here.
    // Restore the OpenGL states.
    return YES;
}
```

You can retrieve the corresponding OpenGL pixel format by calling the function `CGLGetPixelFormat`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

colorSpace

Returns the color space used by the rendering context. (required)

```
- (CGColorSpaceRef) colorSpace
```

Return Value
An RGB color space; `NULL` if the custom patch execution mode is not consumer.

Discussion
If the method returns a color space, it must be an RGB color space.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

logMessage:

Writes a message to the Quartz Composer log. (required)

```
- (void) logMessage:(NSString*)format, ...
```

Parameters

format
The string to write to the log. The default location for the log is the standard output.

Discussion
This method is an alternative to using the functions `NSLog` or `printf`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

outputImageProviderFromBufferWithPixelFormat:pixelsWide:pixelsHigh:baseAddress:bytesPerRow:releaseCallback:releaseContext:colorSpace:shouldColorMatch:

Returns an image provider from a single memory buffer. (required)

```
- (id) outputImageProviderFromBufferWithPixelFormat:(NSString*)format
    pixelsWide:(NSUInteger)width pixelsHigh:(NSUInteger)height baseAddress:(const
     void*)baseAddress bytesPerRow:(NSUInteger)rowBytes
    releaseCallback:(QCPlugInBufferReleaseCallback)callback
    releaseContext:(void*)context colorSpace:(CGColorSpaceRef)colorSpace
    shouldColorMatch:(BOOL)colorMatch
```

Parameters

format
The pixel format of the memory buffer. This must be compatible with the color space.

width
The width, in bytes, of the memory buffer.

height
The height, in bytes, of the memory buffer.

baseAddress
The base address of the memory buffer, which must be multiple of 16.

rowBytes
The number of bytes per row of the memory buffer, which must be multiple of 16.

callback
The release callback. Your callback must use this type definition:

```
typedef void (*QCPlugInBufferReleaseCallback)(const void* address, void* context);
```

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

```
void MyQCPlugInBufferReleaseCallback (const void address,
              void * context);
```

Quartz Composer invokes your callback when the memory buffer is no longer needed. The callback can be called from any thread at any time

context
The context to pass to the release callback.

colorSpace
The color space of the memory buffer. This must be compatible with the pixel format.

colorMatch
A Boolean that specifies whether Quartz Composer should color match the image. Pass `NO` if the image is a mask or gradient or should not be color matched for some other reason. Otherwise, pass `YES`.

Return Value
An image provider.

Discussion
You must not modify the image until the release callback is invoked.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

outputImageProviderFromTextureWithPixelFormat:pixelsWide:pixelsHigh:name:flipped:releaseCallback:releaseContext:colorSpace:shouldColorMatch:

Returns an image provider from an OpenGL texture. (required)

```
- (id) outputImageProviderFromTextureWithPixelFormat:(NSString*)format
    pixelsWide:(NSUInteger)width pixelsHigh:(NSUInteger)height name:(GLuint)name
    flipped:(BOOL)flipped releaseCallback:(QCPlugInTextureReleaseCallback)callback
     releaseContext:(void*)context colorSpace:(CGColorSpaceRef)colorSpace
    shouldColorMatch:(BOOL)colorMatch;
```

Parameters

format
The pixel format of the texture. This must be compatible with the color space.

width
The width, in bytes, of the texture.

height
The height, in bytes, of the texture.

name
An OpenGL texture of type `GL_TEXTURE_RECTANGLE_EXT` that is valid on the Quartz Composer OpenGL context. Note that textures do not have a retain and release mechanism. This means that your application must make sure that the texture exists for the life cycle of the image provider.

flipped
`YES` to have Quartz Composer flip the contents of the texture vertically.

callback
The release callback. Your callback must use this type definition:

```
typedef void (*QCPlugInTextureReleaseCallback)(CGLContextObj cgl_ctx, GLuint
name, void* context);
```

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

```
void MyQCPlugInTextureReleaseCallback (CGLContextObj cgl_ctx,
              GLuint name,
              void* context);
```

Quartz Composer invokes your callback when the memory buffer is no longer needed. The callback can be called from any thread at any time

context
The context to pass to the release callback.

colorSpace
The color space of the texture. This must be compatible with the pixel format.

colorMatch

A Boolean that specifies whether Quartz Composer should color match the texture. Pass `NO` if the texture is a mask or gradient or should not be color matched for some other reason. Otherwise, pass `YES`.

Return Value
An image provider.

Discussion
You must not modify the texture until the release callback is invoked.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

userInfo

Returns a mutable dictionary that contains information that can be shared between all instances of the `QCPlugIn` subclass, running in the same Quartz Composer context. (required)

```
- (NSMutableDictionary*) userInfo
```

Return Value
A mutable dictionary.

Discussion
When you add information to the dictionary, make sure that you use unique keys, such as `com.myCompany.foo`. You can use this dictionary to cache data that you want to share.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

QCPlugInInputImageSource Protocol Reference

Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCPlugIn.h
Availability	Available in Mac OS X v10.5 and later.
Related sample code	Quartz Composer ImageExporter

Overview

The `QCPlugInInputImageSource` protocol eliminates the need to use explicit image types for the image input ports on your custom patch. Not only does using the protocol avoid restrictions of a specific image type, but it avoids impedance mismatches, and provides better performance by deferring pixel computation until it is needed. When you need to access the pixels in an image, you simply convert the image to a representation (texture or buffer) using one of the methods defined by the `QCPlugInInputImageSource` protocol. Use a texture representation when you want to use input images on the GPU. Use a buffer representation when you want to use input images on the CPU.

Input images are opaque source objects that comply to this protocol. To create an image input port as an Objective-C 2.0 property, declare it as follows:

```
@property(dynamic) id<QCPlugInInputImageSource> inputImage;
```

To create an image input port dynamically. use the type `QCPortTypeImage`:

```
[self addInputPortWithType:QCPortTypeImage
                    forKey:@"inputImage"
            withAttributes:nil];
```

Tasks

Converting an Image to a Representation

- `lockTextureRepresentationWithColorSpace:forBounds:` (page 129) *required method*
  Creates an OpenGL texture representation from a subregion of the image source using the provided color space. (required)
- `unlockTextureRepresentation` (page 133) *required method*
  Releases the OpenGL texture representation of the image source. (required)

- `lockBufferRepresentationWithPixelFormat:colorSpace:forBounds:` (page 128) *required method*
  Creates a memory buffer representation from a subregion of the image source using the provided pixel format and color space. (required)
- `bindTextureRepresentationToCGLContext:textureUnit:normalizeCoordinates:` (page 125) *required method*
  Binds the texture to a given texture unit and optionally scales or flips the texture. (required)
- `unbindTextureRepresentationFromCGLContext:textureUnit:` (page 132) *required method*
  Unbinds the texture from a texture unit. (required)
- `unlockBufferRepresentation` (page 132) *required method*
  Releases the memory buffer representation of the image source. (required)

Getting Color Space Information

- `imageColorSpace` (page 128) *required method*
  Returns the color space of the image source. (required)
- `shouldColorMatch` (page 129) *required method*
  Returns whether or not the image source should be color matched. (required)

Getting Texture Information

- `texturePixelsWide` (page 131) *required method*
  Returns the width of the texture representation. (required)
- `texturePixelsHigh` (page 131) *required method*
  Returns the height of the texture representation. (required)
- `textureTarget` (page 132) *required method*
  Returns the texture target. (required)
- `textureName` (page 131) *required method*
  Returns the texture name. (required)
- `textureColorSpace` (page 129) *required method*
  Returns the color space of the texture representation. (required)
- `textureFlipped` (page 130) *required method*
  Returns whether or not the contents of the texture are flipped vertically. (required)
- `textureMatrix` (page 130) *required method*
  Returns a texture matrix. (required)

Getting Image Buffer Information

- `imageBounds` (page 127) *required method*
  Returns the actual bounds of the image source expressed in pixels and aligned to integer boundaries. (required)
- `bufferPixelsWide` (page 127) *required method*
  Returns the width of the image buffer representation. (required)

- `bufferPixelsHigh` (page 127) *required method*
  Returns the height of the image buffer representation. (required)
- `bufferPixelFormat` (page 126) *required method*
  Returns the pixel format of the image buffer representation. (required)
- `bufferColorSpace` (page 126) *required method*
  Returns the color space of the image buffer representation. (required)
- `bufferBaseAddress` (page 125) *required method*
  Returns the base address of the image buffer. (required)
- `bufferBytesPerRow` (page 126) *required method*
  Returns the bytes per row of the buffer representation. (required)

Instance Methods

bindTextureRepresentationToCGLContext:textureUnit:normalizeCoordinates:

Binds the texture to a given texture unit and optionally scales or flips the texture. (required)

```
- (void) bindTextureRepresentationToCGLContext:(CGLContextObj)cgl_ctx
    textureUnit:(GLenum)unit normalizeCoordinates:(BOOL)flag
```

Parameters

cgl_ctx
The CGL context to render to.)

unit
The texture unit to bind to (such as, `GL_TEXTURE0`)

flag
To apply a texture matrix to scale coordinates (from `[0, pixels]` to `[0,1]`) and flip them vertically (if necessary), pass `YES`.

Discussion
When you no longer need the texture, call `unbindTextureRepresentationFromCGLContext:textureUnit:` (page 132).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

bufferBaseAddress

Returns the base address of the image buffer. (required)

```
- (const void*) bufferBaseAddress
```

Return Value
The base address of the buffer.

Discussion
The base address is guaranteed to be aligned on a 16-byte boundary.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

bufferBytesPerRow

Returns the bytes per row of the buffer representation. (required)

```
- (NSUInteger) bufferBytesPerRow
```

Return Value
The number of bytes per row of the buffer.

Discussion
The number of bytes per row is guaranteed to be a multiple of 16.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

bufferColorSpace

Returns the color space of the image buffer representation. (required)

```
- (CGColorSpaceRef) bufferColorSpace
```

Return Value
The color space of the image buffer.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

bufferPixelFormat

Returns the pixel format of the image buffer representation. (required)

```
- (NSString*) bufferPixelFormat
```

Return Value
A string that specifies the pixel format. The supported formats are ARGB8 (8-bit alpha, red, green, blue), BGRA8 (8-bit blue, green, red, and alpha), RGBAf (floating-point, red, green, blue, alpha), I8 (8-bit intensity), and If (floating-point intensity).

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCPlugIn.h

bufferPixelsHigh

Returns the height of the image buffer representation. (required)

```
- (NSUInteger) bufferPixelsHigh
```

Return Value
The height, expressed in pixels.

Availability
Available in Mac OS X v10.5 and later.

See Also
- bufferPixelsHigh (page 127)

Declared In
QCPlugIn.h

bufferPixelsWide

Returns the width of the image buffer representation. (required)

```
- (NSUInteger) bufferPixelsWide
```

Return Value
The width, expressed in pixels.

Availability
Available in Mac OS X v10.5 and later.

See Also
- bufferPixelsHigh (page 127)

Declared In
QCPlugIn.h

imageBounds

Returns the actual bounds of the image source expressed in pixels and aligned to integer boundaries. (required)

```
- (NSRect) imageBounds;
```

Return Value
The bounds of the image source.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

imageColorSpace

Returns the color space of the image source. (required)

```
- (CGColorSpaceRef) imageColorSpace
```

Return Value
The color space of the image source, typically RGB or Gray type.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

lockBufferRepresentationWithPixelFormat:colorSpace:forBounds:

Creates a memory buffer representation from a subregion of the image source using the provided pixel format and color space. (required)

```
- (BOOL) lockBufferRepresentationWithPixelFormat:(NSString*)format
    colorSpace:(CGColorSpaceRef)colorSpace forBounds:(NSRect)bounds
```

Parameters

format
A pixel format that is compatible with the color space.

colorSpace
A Quartz color space that is compatible with the pixel format.

bounds
The bounds of the subregion, expressed as pixels, and aligned to integer boundaries.

Return Value
`YES` if successful; otherwise `NO`.

Discussion
The content of the buffer is read-only. You should not attempt to modify it.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- unlockBufferRepresentation` (page 132)

Declared In
`QCPlugIn.h`

lockTextureRepresentationWithColorSpace:forBounds:

Creates an OpenGL texture representation from a subregion of the image source using the provided color space. (required)

```
- (BOOL) lockTextureRepresentationWithColorSpace:(CGColorSpaceRef)colorSpace
    forBounds:(NSRect)bounds
```

Parameters

colorSpace

A Quartz color space.

bounds

The bounds of the subregion, expressed in pixels. They must be aligned to integer boundaries.

Return Value

`YES` is successful; `NO` if texture can't be created.

Discussion

Neither the content of the texture nor its states (for example, the wrap mode) must be modified; you can only draw with it. The texture is valid only in the plug-in context.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- unlockTextureRepresentation` (page 133)

Declared In

`QCPlugIn.h`

shouldColorMatch

Returns whether or not the image source should be color matched. (required)

```
- (BOOL) shouldColorMatch
```

Return Value

`NO` if the source is a mask or gradient; `YES` otherwise.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`QCPlugIn.h`

textureColorSpace

Returns the color space of the texture representation. (required)

```
- (CGColorSpaceRef) textureColorSpace
```

Return Value

The color space of the texture.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

textureFlipped

Returns whether or not the contents of the texture are flipped vertically. (required)

```
- (BOOL) textureFlipped
```

Return Value
`YES` if the contents of the texture are flipped (upside-down); `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

textureMatrix

Returns a texture matrix. (required)

```
- (const GLfloat*) textureMatrix
```

Return Value
A 4x4 texture matrix created by scaling (from [`0`, pixels] to [`0`,`1`]) and vertically flipping the texture coordinates; `NULL` if coordinate transformation is not required.

Discussion
This method is provided as a convenience for 2D textures to take care of two issues:

- Coordinates for rectangular textures are expressed in pixels rather than the normalized units used for power-of-two textures. The coordinates need to be normalized before you can process the texture.
- Texture coordinates are typically flipped by OpenGL for processing on the GPU and need to be flipped to the original coordinates.

You can take care of these two issues simply by loading a the matrix returned by this method onto the OpenGL stack. If you are not sure that your texture needs either of these operations, you can load the matrix on the OpenGL stack anyway, as it acts as an identity matrix if it's not needed.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

textureName

Returns the texture name. (required)

```
- (GLuint) textureName
```

Return Value
The texture name.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCPlugIn.h

texturePixelsHigh

Returns the height of the texture representation. (required)

```
- (NSUInteger) texturePixelsHigh
```

Return Value
The height of the texture, expressed in pixels.

Availability
Available in Mac OS X v10.5 and later.

See Also
- texturePixelsWide (page 131)

Declared In
QCPlugIn.h

texturePixelsWide

Returns the width of the texture representation. (required)

```
- (NSUInteger) texturePixelsWide
```

Return Value
The width of the texture, expressed in pixels.

Availability
Available in Mac OS X v10.5 and later.

See Also
- texturePixelsHigh (page 131)

Declared In
QCPlugIn.h

textureTarget

Returns the texture target. (required)

```
- (GLenum) textureTarget
```

Return Value
The texture target, either `GL_TEXTURE_2D` or `GL_TEXTURE_RECTANGLE_EXT`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

unbindTextureRepresentationFromCGLContext:textureUnit:

Unbinds the texture from a texture unit. (required)

```
- (void) unbindTextureRepresentationFromCGLContext:(CGLContextObj)cgl_ctx
    textureUnit:(GLenum)unit
```

Parameters

cgl_ctx
A CGL context.)

unit
The texture unit to unbind from (such as, `GL_TEXTURE0`)

Availability
Available in Mac OS X v10.5 and later.

See Also
`- bindTextureRepresentationToTextureUnit:normalizeCoordinates:` (page 125)

Declared In
`QCPlugIn.h`

unlockBufferRepresentation

Releases the memory buffer representation of the image source. (required)

```
- (void) unlockBufferRepresentation
```

Availability
Available in Mac OS X v10.5 and later.

See Also
`- lockBufferRepresentationWithPixelFormat:colorSpace:` (page 128)

Declared In
`QCPlugIn.h`

unlockTextureRepresentation

Releases the OpenGL texture representation of the image source. (required)

```
- (void) unlockTextureRepresentation
```

Availability
Available in Mac OS X v10.5 and later.

See Also
`- lockTextureRepresentationWithTarget:colorSpace:forBounds:` (page 129)

Declared In
`QCPlugIn.h`

QCPlugInOutputImageProvider Protocol Reference

Framework	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
Declared in	QuartzComposer/QCPlugIn.h
Availability	Available in Mac OS X v10.5 and later.

Overview

The `QCPlugInOuputImageProvider` protocol eliminates the need to use explicit image types for the image output ports on a custom patch. The methods in this protocol are called by the Quartz Composer engine when the output image is needed. If your custom patch has an image output port, you need to implement the appropriate methods for rendering image data and to supply information about the rendering destination and the image bounds.

Output images are opaque provider objects that comply to this protocol. To create an image output port as an Objective-C 2.0 property, declare it as follows:

```
@property(dynamic) id<QCPlugInOutputImageProvider> outputImage;
```

To create an image input port dynamically use the type `QCPortTypeImage`:

```
[self addOutputPortWithType:QCPortTypeImage
                     forKey:@"outputImage"
             withAttributes:nil];
```

To write images to that port, you need to implement the methods in this protocol and create an internal class that represents the images produced by the custom patch. For example, a simple interface for an image provider is:

```
@interface MyOutputImage : NSObject <QCPlugInOutputImageProvider>
{
    NSUInteger _width;
    NSUInteger _height;
}
```

Tasks

Rendering an Image to a Destination

- `renderToBuffer:withBytesPerRow:pixelFormat:forBounds:` (page 139) *required method*
  Renders a subregion of the image into the supplied memory buffer using the specified pixel format. (required)
- `copyRenderedTextureForCGLContext:pixelFormat:bounds:isFlipped:` (page 137) *required method*
  Returns the name of an OpenGL texture of type `GL_TEXTURE_RECTANGLE_EXT` that contains a subregion of the image in a given pixel format. (required)
- `renderWithCGLContext:forBounds:` (page 139) *required method*
  Renders a subregion of the image to the provided CGL context. (required)
- `releaseRenderedTexture:forCGLContext:` (page 138) *required method*
  Releases the previously copied texture. (required)

Providing Information About the Image

- `imageBounds` (page 138) *required method*
  Returns the bounds of the image expressed in pixels and aligned to integer boundaries. (required)
- `imageColorSpace` (page 138) *required method*
  Returns the color space of the image or `NULL` if the image should not be color matched. (required)
- `shouldColorMatch` (page 140) *required method*
  Returns whether the image should be color matched. (required)

Providing Information About the Rendering Destination

- `supportedBufferPixelFormats` (page 140) *required method*
  Returns a list of pixel formats that are supported for rendering to a memory buffer. (required)
- `supportedRenderedTexturePixelFormats` (page 141) *required method*
  Returns a list of pixel formats that are supported for rendering to an onscreen OpenGL context. (required)
- `canRenderWithCGLContext:` (page 136) *required method*
  Returns whether the image data can be rendered into the provided CGL context. (required)

Instance Methods

canRenderWithCGLContext:

Returns whether the image data can be rendered into the provided CGL context. (required)

```
- (BOOL) canRenderWithCGLContext:(CGLContextObj)cgl_ctx
```

Parameters

ctx

The CGL context that your image will be rendered to.

Return Value

YES if the image can be rendered into this CGL context; otherwise NO, in which case renderToBuffer:withBytesPerRow:pixelFormat:forBounds: (page 139) is called.

Discussion

If your image can render using any OpenGL context, simply return YES. If your code requires special extensions, you'll need to check for them and then provide the appropriate return value. For more information on checking for OpenGL capabilities supported by the hardware, see *OpenGL Programming Guide for Mac OS X*.

Availability

Available in Mac OS X v10.5 and later.

Declared In

QCPlugIn.h

copyRenderedTextureForCGLContext:pixelFormat:bounds:isFlipped:

Returns the name of an OpenGL texture of type GL_TEXTURE_RECTANGLE_EXT that contains a subregion of the image in a given pixel format. (required)

```
- (GLuint) copyRenderedTextureForCGLContext:(CGLContextObj)cgl_ctx
    pixelFormat:(NSString*)format bounds:(NSRect)bounds isFlipped:(BOOL*)flipped
```

Parameters

cgl_ctx

The CGL context to render to.

format

A string that represents the pixel format of the texture.

bounds

The bounds of the subregion of the image.

isFlipped

Set to YES on output if the contents of the returned texture are vertically flipped.

Return Value

The name of an OpenGL texture of type GL_TEXTURE_RECTANGLE_EXT that contains a subregion of the image in a given pixel format or 0 if the texture can't be provided.

Discussion

Implement this method if you want to create the texture yourself or use framebuffer objects (FBO). Use <OpenGL/CGLMacro.h> to send commands to the OpenGL context. Make sure to preserve all the OpenGL states except the ones defined by GL_CURRENT_BIT.

Availability

Available in Mac OS X v10.5 and later.

Declared In

QCPlugIn.h

imageBounds

Returns the bounds of the image expressed in pixels and aligned to integer boundaries. (required)

```
- (NSRect) imageBounds;
```

Return Value
The bounds of the image. Note that the `QCPlugIn` class does not support images that have infinite bounds.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

imageColorSpace

Returns the color space of the image or `NULL` if the image should not be color matched. (required)

```
- (CGColorSpaceRef) imageColorSpace
```

Return Value
The color space of the image or `NULL`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

releaseRenderedTexture:forCGLContext:

Releases the previously copied texture. (required)

```
- (void) releaseRenderedTexture:(GLuint)name forCGLContext:(CGLContextObj)cgl_ctx;
```

Parameters

name
The name of the previously bound texture.

cgl_ctx
The CGL context.

Discussion
Your OpenGL code should save and restore all states *except* for those that are part of `GL_CURRENT_BIT` (vertex position, color, texture, and so on). Also use CGL macros instead of changing the current context, by including this statement:

```
#import <OpenGL/CGLMacro.h>
```

For more details, see *Quartz Composer Custom Patch Programming Guide*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
QCPlugIn.h

renderToBuffer:withBytesPerRow:pixelFormat:forBounds:

Renders a subregion of the image into the supplied memory buffer using the specified pixel format. (required)

```
- (BOOL) renderToBuffer:(void*)baseAddress withBytesPerRow:(NSUInteger)rowBytes
    pixelFormat:(NSString*)format forBounds:(NSRect)bounds
```

Parameters

baseAddress
The base address of the memory buffer. The Quartz Composer engine passes you an address that is aligned on a 16-byte boundary.

rowBytes
The number of bytes per row of the image data. The Quartz Composer engine guarantees this value is a multiple of 16.

format
The pixel format of the image data.

bounds
The bounds of the subregion.

Return Value
YES if the image is rendered successfully into the buffer; NO on failure or if the image provider doesn't support CPU rendering.

Discussion
The Quartz Composer engine calls this method when it needs pixels. It gives you the base address, the number of row bytes, and the format. Then, you write pixels to the buffer.

Availability
Available in Mac OS X v10.5 and later.

See Also
- renderWithCGLContext:forBounds: (page 139)

Declared In
QCPlugIn.h

renderWithCGLContext:forBounds:

Renders a subregion of the image to the provided CGL context. (required)

```
- (BOOL) renderWithCGLContext:(CGLContextObj)cgl_ctx forBounds:(NSRect)bounds
```

Parameters

cgl_ctx
The CGL context to render to.

bounds
The bounds of the subregion.

Return Value
`YES` if successful; `NO` on failure or if the image provider doesn't support GPU rendering.

Discussion
The view port is set for you. The model view and projection matrixes are set to the identity.

Your OpenGL code should save and restore all states *except* for those that are part of `GL_CURRENT_BIT` (vertex position, color, texture, and so on). Also use CGL macros instead of changing the current context, by including this statement:

```
#import <OpenGL/CGLMacro.h>
```

For more details, see *Quartz Composer Custom Patch Programming Guide*.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- renderToBuffer:withBytesPerRow:pixelFormat:forBounds:` (page 139)

Declared In
`QCPlugIn.h`

shouldColorMatch

Returns whether the image should be color matched. (required)

```
- (BOOL) shouldColorMatch
```

Return Value
`NO` if the image is a mask or gradient; otherwise `YES`, which is the default.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QCPlugIn.h`

supportedBufferPixelFormats

Returns a list of pixel formats that are supported for rendering to a memory buffer. (required)

```
- (NSArray*) supportedBufferPixelFormats
```

Return Value
A list of pixel formats, in order of preference, that the image can be rendered to in memory, or `nil` if the image provider does not support rendering to the CPU.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- supportedRenderedTexturePixelFormats` (page 141)

Declared In
QCPlugIn.h

supportedRenderedTexturePixelFormats

Returns a list of pixel formats that are supported for rendering to an onscreen OpenGL context. (required)

```
- (NSArray*) supportedRenderedTexturePixelFormats
```

Return Value
Returns the list of texture pixel formats supported by `copyRenderedTextureForCGLContext:pixelFormat:bounds:isFlipped:` (page 137) or `nil` if not supported.

Discussion
If this method returns nil, then Quartz Composer calls `canRenderWithCGLContext:` (page 136) /`renderWithCGLContext:forBounds:` (page 139).

Availability
Available in Mac OS X v10.5 and later.

See Also
`- supportedBufferPixelFormats` (page 140)

Declared In
QCPlugIn.h

Document Revision History

This table describes the changes to *Quartz Composer Reference Collection*.

Date	Notes
2007-01-25	Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a collection of separate documents.

Document Revision History

Index

H

I

K

L

M

N

O

P

Q

R

S

T

U

V