UIView Class Reference

User Experience: Windows & Views

2010-06-04

Contents

UIView Class Reference

Inherits from	UIResponder : NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITextField.h UIView.h
Related sample code	iAdSuite iPhoneCoreDataRecipes ScrollViewSuite Touches UICatalog

Overview

The `UIView` class implements the basic behavior used to facilitate drawing in your applications. You can use this class as-is to act as a simple container for other view objects. You can also subclass it and override its methods to draw custom content. Because it is also a responder object, you can also respond to interactions with that content.

`UIView` objects are arranged within an `UIWindow` object, in a nested hierarchy of subviews. Parent objects in the view hierarchy are called **superviews**, and children are called **subviews**. A view object claims a rectangular region of its enclosing superview, is responsible for all drawing within that region, and is eligible to receive events occurring in it as well. Sibling views are able to overlap without any issues, allowing complex view placement.

The `UIView` class provides common methods you use to create all types of views and access their properties. For example, unless a subclass has its own designated initializer, you use the `initWithFrame:` (page 49) method to create a view. The `frame` (page 20) property specifies the origin and size of a view in superview coordinates. The origin of the coordinate system for all views is in the upper-left corner.

You can also use the `center` (page 17) and `bounds` (page 16) properties to set the position and size of a view. The `center` property specifies the view's center point in superview's coordinates. The bounds property specifies the origin in the view's coordinates and its size (the view's content may be larger than the bounds size). The `frame` property is actually computed based on the `center` and bounds property values. Therefore, you can set any of these three properties and they affect the values of the others.

It's important to set the autoresizing properties of views so that when they are displayed or the orientation changes, the views are displayed correctly within the superview's bounds. Use the `autoresizesSubviews` (page 14) property, especially if you subclass `UIView`, to specify whether the view should automatically resize its subviews. Use the `autoresizingMask` (page 15) property with the constants described in `UIViewAutoresizing` (page 62) to specify how a view should automatically resize.

The `UIView` class provides a number of methods for managing the view hierarchy. Use the `superview` (page 24) property to get the parent view and the `subviews` (page 23) property to get the child views in the hierarchy. There are also a number of methods, listed in "Managing the View Hierarchy" (page 9), for adding, inserting, and removing subviews as well as arranging subviews in front of or in back of siblings.

When you subclass `UIView` to create a custom class that draws itself, implement the `drawRect:` (page 47) method to draw the view within the specified region. This method is invoked the first time a view displays or when an event occurs that invalidates a part of the view's frame requiring it to redraw its content.

Normal geometry changes do not require redrawing the view. Therefore, if you alter the appearance of a view and want to force it to redraw, send `setNeedsDisplay` (page 54) or `setNeedsDisplayInRect:` (page 55) to the view. You can also set the `contentMode` (page 18) to `UIViewContentModeRedraw` (page 60) to invoke the `drawRect:` (page 47) method when the bounds change; otherwise, the view is scaled and clipped without redrawing the content.

Subclasses can also be containers for other views. In this case, just override the designated initializer, `initWithFrame:` (page 49), to create a view hierarchy. If you want to programmatically force the layout of subviews before drawing, send `setNeedsLayout` (page 55) to the view. Then when `layoutIfNeeded` (page 52) is invoked, the `layoutSubviews` (page 52) method is invoked just before displaying. Subclasses should override `layoutSubviews` (page 52) to perform any custom arrangement of subviews.

Some of the property changes to view objects can be animated—for example, setting the `frame` (page 20), `bounds` (page 16), `center` (page 17), and `transform` (page 25) properties. If you change these properties in an animation block, the changes from the current state to the new state are animated. Invoke the `beginAnimations:context:` (page 29) class method to begin an animation block, set the properties you want animated, and then invoke the `commitAnimations` (page 29) class method to end an animation block. The animations are run in a separate thread and begin when the application returns to the run loop. Other animation class methods allow you to control the start time, duration, delay, and curve of the animations within the block.

Use the `hitTest:withEvent:` (page 48) and `pointInside:withEvent:` (page 53) methods if you are processing events and want to know where they occur. The `UIView` class inherits other event processing methods from `UIResponder`. For more information on how views handle events, read *UIResponder Class Reference*.

To associate a gesture recognizer with a view so that object can interpret gestures made on the view, you must call the `addGestureRecognizer:` (page 41) method. (Gesture recognizers are instances of a concrete subclass of `UIGestureRecognizer`.) You remove a gesture recognizer with the `removeGestureRecognizer:` (page 54) method and find out which gesture recognizers are associated with a view using the `gestureRecognizers` (page 21) property. Gesture recognition is a feature that was introduced in iOS 3.2.

Read Window and Views in *iOS Application Programming Guide* to learn how to use this class.

Note: Prior to iOS 3.0, `UIView` instances may have a maximum height and width of 1024 x 1024. In iOS 3.0 and later, views are no longer restricted to this maximum size but are still limited by the amount of memory they consume. Therefore, it is in your best interests to keep view sizes as small as possible. Regardless of which version of iOS is running, you should consider using a `CATiledLayer` object if you need to create views larger than 1024 x 1024 in size.

Tasks

Creating Instances

`– initWithFrame:` (page 49)
    Initializes and returns a newly allocated view object with the specified frame rectangle.

Setting and Getting Attributes

`userInteractionEnabled` (page 25) *property*
    A Boolean value that determines whether user events are ignored and removed from the event queue.

Modifying the Bounds and Frame Rectangles

`frame` (page 20) *property*
    The receiver's frame rectangle.

`bounds` (page 16) *property*
    The receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

`center` (page 17) *property*
    The center of the frame.

`transform` (page 25) *property*
    Specifies the transform applied to the receiver, relative to the center of its bounds.

Managing the View Hierarchy

`superview` (page 24) *property*
    The receiver's superview, or `nil` if it has none. (read-only)

`subviews` (page 23) *property*
    The receiver's immediate subviews. (read-only)

`window` (page 26) *property*
    The receiver's window object, or `nil` if it has none. (read-only)

`– addSubview:` (page 42)
    Adds a view to the receiver's subviews so it's displayed above its siblings.

`– bringSubviewToFront:` (page 43)
    Moves the specified subview to the front of its siblings.

– `sendSubviewToBack:` (page 54)
  Moves the specified subview to the back of its siblings.

– `removeFromSuperview` (page 53)
  Unlinks the receiver from its superview and its window, and removes it from the responder chain.

– `insertSubview:atIndex:` (page 50)
  Inserts a subview at the specified index.

– `insertSubview:aboveSubview:` (page 50)
  Inserts a view above another view in the view hierarchy.

– `insertSubview:belowSubview:` (page 51)
  Inserts a view below another view in the view hierarchy.

– `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 48)
  Exchanges the subviews in the receiver at the given indices.

– `isDescendantOfView:` (page 51)
  Returns a Boolean value indicating whether the receiver is a subview of a given view or whether it is identical to that view.

Converting Coordinates

– `convertPoint:toView:` (page 44)
  Converts a point from the receiver's coordinate system to that of a given view.

– `convertPoint:fromView:` (page 43)
  Converts a point from the coordinate system of a given view to that of the receiver.

– `convertRect:toView:` (page 45)
  Converts a rectangle from the receiver's coordinate system to that of another view.

– `convertRect:fromView:` (page 44)
  Converts a rectangle from the coordinate system of another view to that of the receiver.

`contentScaleFactor` (page 18) *property*
  The scale factor applied to the view.

Resizing Subviews

`autoresizesSubviews` (page 14) *property*
  A Boolean value that determines whether the receiver automatically resizes its subviews when its frame size changes.

`autoresizingMask` (page 15) *property*
  An integer bit mask that determines how the receiver resizes itself when its bounds change.

– `sizeThatFits:` (page 56)
  Asks the view to calculate and return the size that best fits its subviews.

– `sizeToFit` (page 57)
  Resizes and moves the receiver view so it just encloses its subviews.

`contentMode` (page 18) *property*
  A flag used to determine how a view lays out its content when its bounds rectangle changes.

`contentStretch` (page 19) *property*
  The rectangle that defines the stretchable and nonstretchable regions of a view.

Searching for Views

`tag` (page 24) *property*
    The receiver's tag, an integer that you can use to identify view objects in your application.

`- viewWithTag:` (page 57)
    Returns the view with the specified tag.

Laying out Views

`- setNeedsLayout` (page 55)
    Sets whether subviews need to be rearranged before displaying.

`- layoutIfNeeded` (page 52)
    Lays out the subviews if needed.

`- layoutSubviews` (page 52)
    Lays out subviews.

Displaying

`clipsToBounds` (page 18) *property*
    A Boolean value that determines whether subviews can be drawn outside the bounds of the receiver.

`backgroundColor` (page 15) *property*
    The receiver's background color.

`alpha` (page 14) *property*
    The receiver's alpha value.

`opaque` (page 23) *property*
    A Boolean value that determines whether the receiver is opaque.

`clearsContextBeforeDrawing` (page 17) *property*
    A Boolean value that determines whether the receiver's bounds should be automatically cleared before drawing.

`- drawRect:` (page 47)
    Draws the receiver's image within the passed-in rectangle.

`- setNeedsDisplay` (page 54)
    Controls whether the receiver's entire bounds rectangle is marked as needing display.

`- setNeedsDisplayInRect:` (page 55)
    Marks the region of the receiver within the specified rectangle as needing display, increasing the receiver's existing invalid region to include it.

`+ layerClass` (page 30)
    Returns the class used to create the layer for instances of this class.

`layer` (page 22) *property*
    The view's Core Animation layer used for rendering. (read-only)

`hidden` (page 21) *property*
    A Boolean value that determines whether the receiver is hidden.

Animating Views with Blocks

+ `animateWithDuration:delay:options:animations:completion:` (page 27)
  Animate changes to one or more views using the specified duration, delay, options, and completion handler.

+ `animateWithDuration:animations:completion:` (page 27)
  Animate changes to one or more views using the specified duration and completion handler.

+ `animateWithDuration:animations:` (page 26)
  Animate changes to one or more views using the specified duration.

+ `transitionWithView:duration:options:animations:completion:` (page 40)
  Creates a transition animation for the specified container view.

+ `transitionFromView:toView:duration:options:completion:` (page 39)
  Creates a transition animation between the specified views using the given parameters.

Animating Views

+ `beginAnimations:context:` (page 29)
  Begins an animation block.

+ `commitAnimations` (page 29)
  Ends an animation block and starts animations when this is the outer animation block.

+ `setAnimationStartDate:` (page 37)
  Sets the start time of animating property changes within an animation block.

+ `setAnimationsEnabled:` (page 37)
  Sets whether animations are enabled.

+ `setAnimationDelegate:` (page 33)
  Sets the delegate for animation messages.

+ `setAnimationWillStartSelector:` (page 39)
  Sets the message to send to the animation delegate when animation starts.

+ `setAnimationDidStopSelector:` (page 33)
  Sets the message to send to the animation delegate when animation stops.

+ `setAnimationDuration:` (page 34)
  Sets the duration (in seconds) of animating property changes within an animation block.

+ `setAnimationDelay:` (page 32)
  Sets the delay (in seconds) of animating property changes within an animation block.

+ `setAnimationCurve:` (page 31)
  Sets the curve of animating property changes within an animation block.

+ `setAnimationRepeatCount:` (page 36)
  Sets the number of times animations within an animation block repeat.

+ `setAnimationRepeatAutoreverses:` (page 35)
  Sets whether the animation of property changes within an animation block automatically reverses repeatedly.

+ `setAnimationBeginsFromCurrentState:` (page 31)
  Sets whether the animation should begin playing from the current state.

`+ setAnimationTransition:forView:cache:` (page 38)
  Sets a transition to apply to a view during an animation block.

`+ areAnimationsEnabled` (page 28)
  Returns a Boolean value indicating whether animations are enabled.

Handling Events

`- hitTest:withEvent:` (page 48)
  Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains a specified point.

`- pointInside:withEvent:` (page 53)
  Returns a Boolean value indicating whether the receiver contains the specified point.

`multipleTouchEnabled` (page 22) *property*
  A Boolean value indicating whether the receiver handles multi-touch events.

`exclusiveTouch` (page 20) *property*
  A Boolean value indicating whether the receiver handles touch events exclusively.

`- endEditing:` (page 47)
  Causes the view (or one of its embedded text fields) to resign the first responder status.

Managing Gesture Recognizers

`- addGestureRecognizer:` (page 41)
  Attaches a gesture recognizer to the receiving view.

`- removeGestureRecognizer:` (page 54)
  Detaches a gesture recognizer from the receiving view.

`gestureRecognizers` (page 21) *property*
  The gesture-recognizer objects currently attached to the view.

Observing Changes

`- didAddSubview:` (page 45)
  Tells the view when subviews are added.

`- didMoveToSuperview` (page 46)
  Informs the receiver that its superview has changed (possibly to `nil`).

`- didMoveToWindow` (page 46)
  Informs the receiver that it has been added to a window.

`- willMoveToSuperview:` (page 58)
  Informs the receiver that its superview is about to change to the specified superview (which may be `nil`).

`- willMoveToWindow:` (page 58)
  Informs the receiver that it's being added to the view hierarchy of the specified window object (which may be `nil`).

`– willRemoveSubview:` (page 58)
: Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

alpha

The receiver’s alpha value.

```
@property(nonatomic) CGFloat alpha
```

Discussion
Changes to this property can be animated. Use the `beginAnimations:context:` (page 29) class method to begin and the `commitAnimations` (page 29) class method to end an animation block.

Availability
Available in iOS 2.0 and later.

See Also
`@property backgroundColor` (page 15)
`@property opaque` (page 23)

Related Sample Code
iPhoneCoreDataRecipes
LocateMe
Reflection
SimpleGestureRecognizers
SimpleNetworkStreams

Declared In
`UIView.h`

autoresizesSubviews

A Boolean value that determines whether the receiver automatically resizes its subviews when its frame size changes.

```
@property(nonatomic) BOOL autoresizesSubviews
```

Discussion
If `YES`, the receiver adjusts the size of its subviews when the bounds change. The default value is `YES`.

Availability
Available in iOS 2.0 and later.

See Also
`@property autoresizingMask` (page 15)

Declared In
`UIView.h`

autoresizingMask

An integer bit mask that determines how the receiver resizes itself when its bounds change.

```
@property(nonatomic) UIViewAutoresizing autoresizingMask
```

Discussion
This mask can be specified by combining, using the C bitwise OR operator, any of the options described in `UIViewAutoresizing` (page 62).

Where more than one option along an axis is set, the default behavior is to distribute the size difference as evenly as possible among the flexible portions. For example, if `frame` (page 20) and `autoresizingMask` (page 15) are set and the superview's width has increased by 10.0 units, the receiver's frame and right margin are each widened by 5.0 units. Subclasses of `UIView` can override the `layoutSubviews` (page 52) method to explicitly adjust the position of subviews.

If the autoresizing mask is equal to `UIViewAutoresizingNone` (page 62), then the receiver doesn't resize at all when its bounds changes. The default value is `UIViewAutoresizingNone`.

Availability
Available in iOS 2.0 and later.

See Also
  `@property autoresizesSubviews` (page 14)

Related Sample Code
AdvancedTableViewCells
iAdSuite
LocateMe
SeismicXML
UICatalog

Declared In
`UIView.h`

backgroundColor

The receiver's background color.

```
@property(nonatomic, copy) UIColor *backgroundColor
```

Discussion
Changes to this property can be animated. The default is `nil`.

Availability
Available in iOS 2.0 and later.

See Also
  `@property alpha` (page 14)

`@property opaque` (page 23)

Related Sample Code
AdvancedTableViewCells

BubbleLevel

iPhoneCoreDataRecipes

Reflection

UICatalog

Declared In
`UIView.h`

bounds

The receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

```
@property(nonatomic) CGRect bounds
```

Discussion
The bounds rectangle determines the origin and scale in the view's coordinate system within its frame rectangle and is measured in points. Setting this property changes the value of the `frame` (page 20) property accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 47) method. If you want the `drawRect:` (page 47) method invoked when the frame rectangle changes, set the `contentMode` (page 18) property to `UIViewContentModeRedraw` (page 60).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 29) class method to begin and the `commitAnimations` (page 29) class method to end an animation block.

The default bounds origin is `(0,0)` and the size is the same as the frame rectangle's size.

Availability
Available in iOS 2.0 and later.

See Also
`@property frame` (page 20)
`@property center` (page 17)
`@property transform` (page 25)

Related Sample Code
iAdSuite

QuartzDemo

Teslameter

TheElements

UICatalog

Declared In
`UIView.h`

center

The center of the frame.

```
@property(nonatomic) CGPoint center
```

Discussion

The center is specified within the coordinate system of its superview and is measured in points. Setting this property changes the values of the `frame` (page 20) properties accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 47) method. If you want the `drawRect:` (page 47) method invoked when the frame rectangle changes, set the `contentMode` (page 18) property to `UIViewContentModeRedraw` (page 60).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 29) class method to begin and the `commitAnimations` (page 29) class method to end an animation block.

Availability

Available in iOS 2.0 and later.

See Also

`@property frame` (page 20)
`@property bounds` (page 16)
`@property transform` (page 25)

Related Sample Code

BubbleLevel
Metronome
MoveMe
QuartzDemo
Touches

Declared In

`UIView.h`

clearsContextBeforeDrawing

A Boolean value that determines whether the receiver's bounds should be automatically cleared before drawing.

```
@property(nonatomic) BOOL clearsContextBeforeDrawing
```

Discussion

The default value of this property is `YES`. When set to `YES`, the current graphics context buffer in the `drawRect:` (page 47) method is automatically cleared to transparent black before `drawRect:` (page 47) is invoked. If the view's `opaque` (page 23) property is also set to `YES`, the `backgroundColor` (page 15) property of the view must not be `nil` or drawing errors may occur.

If the value of this property is `NO`, it is the view's responsibility to completely fill its content. Drawing performance can be improved if this property is `NO`—for example, when scrolling.

Availability

Available in iOS 2.0 and later.

Declared In
`UIView.h`

clipsToBounds

A Boolean value that determines whether subviews can be drawn outside the bounds of the receiver.

```
@property(nonatomic) BOOL clipsToBounds
```

Discussion
`YES` if subviews should be clipped to the bounds of the receiver; otherwise, `NO`. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AdvancedTableViewCells

Scrolling

Declared In
`UIView.h`

contentMode

A flag used to determine how a view lays out its content when its bounds rectangle changes.

```
@property(nonatomic) UIViewContentMode contentMode
```

Discussion
Set to a value described in `UIViewContentMode` (page 60). The default value is `UIViewContentModeScaleToFill` (page 60).

Availability
Available in iOS 2.0 and later.

Related Sample Code
AdvancedTableViewCells

iPhoneCoreDataRecipes

SimpleGestureRecognizers

ZoomingPDFViewer

Declared In
`UIView.h`

contentScaleFactor

The scale factor applied to the view.

```
@property(nonatomic) CGFloat contentScaleFactor
```

Discussion
The scale factor determines how content in the view is mapped from the logical coordinate space (measured in points) to the device coordinate space (measured in pixels). This value is typically either `1.0` or `2.0`. Higher scale factors indicate that each point in the view is represented by more than one pixel in the underlying layer. For example, if the scale factor is `2.0` and the view frame size is 50 x 50 points, the size of the bitmap used to present that content is 100 x 100 pixels.

For views that implement a custom `drawRect:` (page 47) method and are associated with a window, the default value for this property is the scale factor associated with the screen currently displaying the view. For system views and views that are backed by a `CAEAGLLayer` object for, the value of this property may be `1.0` even on high resolution screens.

In general, you should not need to modify the value in this property. However, if your application draws using OpenGL ES, you may want to change the scale factor to support higher-resolution drawing on screens that support it. For more information on how to adjust your OpenGL ES rendering environment, see "Supporting High-Resolution Screens" in *iOS Application Programming Guide*.

Availability
Available in iOS 4.0 and later.

Related Sample Code
PhotoScroller

ZoomingPDFViewer

Declared In
`UIView.h`

contentStretch

The rectangle that defines the stretchable and nonstretchable regions of a view.

```
@property(nonatomic) CGRect contentStretch
```

Discussion
You use this property to control how a view's content is stretched to fill its bounds when the view is resized. Content stretching is often used to animate the resizing of a view. For example, buttons and other controls use stretching to maintain crisp borders while allowing the middle portions of the control to stretch and fill the available space. This technique applies the stretching to the view's underlying layer and alleviates the need to use stretchable `UIImage` objects inside image views.

The values you specify for this rectangle must be normalized to the range 0.0 to 1.0. These values are then scaled to the bounds of the view to obtain the appropriate pixel values. The rectangle's origin point represents the point at which to begin stretching the content. The rectangle's size values indicate the width and height of the stretchable portion. The default value for this rectangle has an origin of (0.0, 0.0) and a size of (1.0, 1.0). This reflects a rectangle whose stretchable portion encompasses the entire view. In other words, the stretchable portion starts at the top-left corner of the view and ends at the bottom-right corner. Specifying a size value of 0.0 stretches the single pixel at the current origin point. For example, to stretch a view's middle pixel only, you could specify an origin of (0.5, 0.5) and a size of (0.0, 0.0).

You can change this property from the default to define a different stretchable area for your content. For example, suppose you have an image view that is 21 pixels wide by 16 pixels high. To make the view stretch horizontally about the middle pixel of its image, you would set the rectangle's origin point to (10/21, 0.0) and its size to (1/21, 1.0).

Availability
Available in iOS 3.0 and later.

Related Sample Code
AdvancedTableViewCells

Declared In
`UIView.h`

exclusiveTouch

A Boolean value indicating whether the receiver handles touch events exclusively.

```
@property(nonatomic, getter=isExclusiveTouch) BOOL exclusiveTouch
```

Discussion
If `YES`, the receiver blocks other views in the same window from receiving touch events; otherwise, it does not. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`@property multipleTouchEnabled` (page 22)

Declared In
`UIView.h`

frame

The receiver's frame rectangle.

```
@property(nonatomic) CGRect frame
```

Discussion
This rectangle is measured in points. Setting the frame rectangle repositions and resizes the receiver within the coordinate system of its superview. The origin of the frame is in superview coordinates. Setting this property changes the values of the `center` (page 17) and `bounds` (page 16) properties accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 47) method. If you want the `drawRect:` (page 47) method invoked when the frame rectangle changes, set the `contentMode` (page 18) property to `UIViewContentModeRedraw` (page 60).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 29) class method to begin and the `commitAnimations` (page 29) class method to end an animation block. If the `transform` (page 25) property is also set, use the `bounds` (page 16) and `center` (page 17) properties instead; otherwise, animating changes to the `frame` property does not correctly reflect the actual location of the view.

Warning: If the `transform` (page 25) property is not the identity transform, the value of this property is undefined and therefore should be ignored.

Availability
Available in iOS 2.0 and later.

See Also
- `@property bounds` (page 16)
- `@property center` (page 17)
- `@property transform` (page 25)

Related Sample Code
GKTank
PVRTextureLoader
ScrollViewSuite
TouchCells
UICatalog

Declared In
`UIView.h`

gestureRecognizers

The gesture-recognizer objects currently attached to the view.

```
@property(nonatomic,copy) NSArray *gestureRecognizers
```

Discussion
Each of these objects is an instance of a subclass of the abstract base class `UIGestureRecognizer`. If there are no gesture recognizers attached, the value of this property is an empty array.

Availability
Available in iOS 3.2 and later.

Declared In
`UIView.h`

hidden

A Boolean value that determines whether the receiver is hidden.

```
@property(nonatomic, getter=isHidden) BOOL hidden
```

Discussion
`YES` if the receiver should be hidden; otherwise, `NO`. The default value is `NO`.

A hidden view disappears from its window and does not receive input events. It remains in its superview's list of subviews, however, and participates in autoresizing as usual. Hiding a view with subviews has the effect of hiding those subviews and any view descendants they might have. This effect is implicit and does not alter the hidden state of the receiver's descendants.

Hiding the view that is the window's current first responder causes the view's next valid key view to become the new first responder.

The value of this property reflects the state of the receiver only and does not account for the state of the receiver's ancestors in the view hierarchy. Thus this property can be `NO` if the receiver is hidden because an ancestor is hidden.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BubbleLevel
CryptoExercise
GKTank
MessageComposer
UICatalog

Declared In
`UIView.h`

layer

The view's Core Animation layer used for rendering. (read-only)

```
@property(nonatomic, readonly, retain) CALayer *layer
```

Discussion
This property is never `nil`. The view is the layer's delegate.

Warning: Since the view is the layer's delegate, you should never set the view as a delegate of another `CALayer` object. Additionally, you should never change the delegate of this layer.

Availability
Available in iOS 2.0 and later.

See Also
`+ layerClass` (page 30)

Related Sample Code
BubbleLevel
Metronome
PhotoScroller
ScrollViewSuite
SpeakHere

Declared In
`UIView.h`

multipleTouchEnabled

A Boolean value indicating whether the receiver handles multi-touch events.

```
@property(nonatomic, getter=isMultipleTouchEnabled) BOOL multipleTouchEnabled
```

Discussion
If `YES`, the receiver handles multi-touch events; otherwise, it does not. If `NO`, the receiver is sent only the first touch event in a multi-touch sequence. Other views in the same window can still receive touch events when this property is `NO`. Set this property and the `exclusiveTouch` (page 20) property to `YES` if this view should handle multi-touch events exclusively—for example, when tracking a sequence of multi-touch events. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`@property exclusiveTouch` (page 20)

Related Sample Code
aurioTouch

Declared In
`UIView.h`

opaque

A Boolean value that determines whether the receiver is opaque.

```
@property(nonatomic, getter=isOpaque) BOOL opaque
```

Discussion
`YES` if it is opaque; otherwise, `NO`. If opaque, the drawing operation assumes that the view fills its bounds and can draw more efficiently. The results are unpredictable if opaque and the view doesn't fill its bounds. Set this property to `NO` if the view is fully or partially transparent. The default value is `YES`.

Availability
Available in iOS 2.0 and later.

See Also
`@property backgroundColor` (page 15)
`@property alpha` (page 14)

Related Sample Code
AdvancedTableViewCells
BubbleLevel
NavBar
QuartzDemo
UICatalog

Declared In
`UIView.h`

subviews

The receiver's immediate subviews. (read-only)

```
@property(nonatomic, readonly, copy) NSArray *subviews
```

Availability
Available in iOS 2.0 and later.

See Also
  `@property superview` (page 24)
`- removeFromSuperview` (page 53)

Related Sample Code
ScrollViewSuite
SimpleNetworkStreams

Declared In
`UIView.h`

superview

The receiver's superview, or `nil` if it has none. (read-only)

```
@property(nonatomic, readonly) UIView *superview
```

Availability
Available in iOS 2.0 and later.

See Also
  `@property subviews` (page 23)
`- removeFromSuperview` (page 53)

Related Sample Code
DateCell
PageControl
Touches
WiTap

Declared In
`UIView.h`

tag

The receiver's tag, an integer that you can use to identify view objects in your application.

```
@property(nonatomic) NSInteger tag
```

Discussion
The default value is `0`. Subclasses can set this to individual tags.

Availability
Available in iOS 2.0 and later.

See Also
`- viewWithTag:` (page 57)

Related Sample Code
iPhoneCoreDataRecipes
QuartzDemo
SeismicXML
TaggedLocations
UICatalog

Declared In
`UIView.h`

transform

Specifies the transform applied to the receiver, relative to the center of its bounds.

```
@property(nonatomic) CGAffineTransform transform
```

Discussion
The origin of the transform is the value of the `center` (page 17) property, or the layer's `anchorPoint` property if it was changed. (Use the `layer` (page 22) property to get the underlying Core Animation layer object.) The default value is `CGAffineTransformIdentity`.

Changes to this property can be animated. Use the `beginAnimations:context:` (page 29) class method to begin and the `commitAnimations` (page 29) class method to end an animation block. The default is whatever the center value is (or anchor point if changed)

Warning: If this property is not the identity transform, the value of the `frame` (page 20) property is undefined and therefore should be ignored.

Availability
Available in iOS 2.0 and later.

See Also
- `@property frame` (page 20)
- `@property bounds` (page 16)
- `@property center` (page 17)

Related Sample Code
GKTank
MoveMe
MoviePlayer
SimpleGestureRecognizers
Touches

Declared In
`UIView.h`

userInteractionEnabled

A Boolean value that determines whether user events are ignored and removed from the event queue.

```
@property(nonatomic, getter=isUserInteractionEnabled) BOOL userInteractionEnabled
```

Discussion
If `NO`, user events—such as touch and keyboard—are ignored and removed from the event queue. The default value is `YES`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
QuartzDemo
TheElements

Declared In
`UIView.h`

window

The receiver's window object, or `nil` if it has none. (read-only)

```
@property(nonatomic, readonly) UIWindow *window
```

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

Class Methods

animateWithDuration:animations:

Animate changes to one or more views using the specified duration.

```
+ (void)animateWithDuration:(NSTimeInterval)duration animations:(void
    (^)(void))animations
```

Parameters

duration
The total duration of the animations, measured in seconds. If you specify a negative value or `0`, the changes are made without animating them.

animations
A block object containing the changes to commit to the views. This is where you programmatically change any animatable properties of the views in your view hierarchy. This block takes no parameters and has no return value. This parameter must not be `NULL`.

Discussion
This method performs the specified animations immediately using the default animation options. The default options are `UIViewAnimationOptionCurveEaseInOut` (page 65) and `UIViewAnimationOptionTransitionNone` (page 65).

Availability
Available in iOS 4.0 and later.

Related Sample Code
iAdSuite

Declared In
`UIView.h`

animateWithDuration:animations:completion:

Animate changes to one or more views using the specified duration and completion handler.

```
+ (void)animateWithDuration:(NSTimeInterval)duration animations:(void
    (^)(void))animations completion:(void (^)(BOOL finished))completion
```

Parameters

duration
The total duration of the animations, measured in seconds. If you specify a negative value or `0`, the changes are made without animating them.

animations
A block object containing the changes to commit to the views. This is where you programmatically change any animatable properties of the views in your view hierarchy. This block takes no parameters and has no return value. This parameter must not be `NULL`.

completion
A block object to be executed when the animation sequence ends. This block has no return value and takes a single Boolean argument that indicates whether or not the animations actually finished before the completion handler was called. If the duration of the animation is 0, this block is performed at the beginning of the next run loop cycle. This parameter may be `NULL`.

Discussion
This method performs the specified animations immediately using the default animation options. The default options are `UIViewAnimationOptionCurveEaseInOut` (page 65) and `UIViewAnimationOptionTransitionNone` (page 65).

For example, if you want to fade a view until it is totally transparent and then remove it from your view hierarchy, you could use code similar to the following:

```
[UIView animateWithDuration:0.2
          animations:^{ view.alpha = 0.0; }
          completion:^(BOOL finished){ [view removeFromSuperview]; }]
```

Availability
Available in iOS 4.0 and later.

Declared In
`UIView.h`

animateWithDuration:delay:options:animations:completion:

Animate changes to one or more views using the specified duration, delay, options, and completion handler.

```
+ (void)animateWithDuration:(NSTimeInterval)duration delay:(NSTimeInterval)delay
    options:(UIViewAnimationOptions)options animations:(void (^)(void))animations
     completion:(void (^)(BOOL finished))completion
```

Parameters

duration
: The total duration of the animations, measured in seconds. If you specify a negative value or `0`, the changes are made without animating them.

delay
: The amount of time (measured in seconds) to wait before beginning the animations. Specify a value of `0` to begin the animations immediately.

options
: A mask of options indicating how you want to perform the animations. For a list of valid constants, see `UIViewAnimationOptions` (page 63).

animations
: A block object containing the changes to commit to the views. This is where you programmatically change any animatable properties of the views in your view hierarchy. This block takes no parameters and has no return value. This parameter must not be `NULL`.

completion
: A block object to be executed when the animation sequence ends. This block has no return value and takes a single Boolean argument that indicates whether or not the animations actually finished before the completion handler was called. If the duration of the animation is 0, this block is performed at the beginning of the next run loop cycle. This parameter may be `NULL`.

Discussion
This method initiates a set of animations to perform on the view. The block object in the animations parameter contains the code for animating the properties of one or more views.

Availability
Available in iOS 4.0 and later.

Declared In
`UIView.h`

areAnimationsEnabled

Returns a Boolean value indicating whether animations are enabled.

```
+ (BOOL)areAnimationsEnabled
```

Return Value
`YES` if animations are enabled; otherwise, `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`+ setAnimationsEnabled:` (page 37)

Related Sample Code
Table View Animations and Gestures

Declared In
`UIView.h`

beginAnimations:context:

Begins an animation block.

```
+ (void)beginAnimations:(NSString *)animationID context:(void *)context
```

Parameters

`animationID`

Application-supplied identifier for the animations within a block that is passed to the animation delegate messages—the selectors set using the `setAnimationWillStartSelector:` (page 39) and `setAnimationDidStopSelector:` (page 33) methods.

`context`

Additional application-supplied information that is passed to the animation delegate messages—the selectors set using the `setAnimationWillStartSelector:` (page 39) and `setAnimationDidStopSelector:` (page 33) methods.

Discussion
The visual changes caused by setting some property values can be animated in an animation block. Animation blocks can be nested. The `setAnimation...` class methods do nothing if they are not invoked in an animation block. Use the `beginAnimations:context:` (page 29) to begin and the `commitAnimations` (page 29) class method to end an animation block.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ commitAnimations` (page 29)
`+ setAnimationWillStartSelector:` (page 39)
`+ setAnimationDidStopSelector:` (page 33)
`+ setAnimationDelegate:` (page 33)

Related Sample Code
MoveMe
ScrollViewSuite
SimpleGestureRecognizers
Touches
UICatalog

Declared In
`UIView.h`

commitAnimations

Ends an animation block and starts animations when this is the outer animation block.

```
+ (void)commitAnimations
```

Discussion
If the current animation block is the outer animation block, starts animations when the application returns to the run loop. Animations are run in a separate thread so the application is not blocked. In this way, multiple animations can be piled on top of one another. See `setAnimationBeginsFromCurrentState:` (page 31) for how to start animations while others are in progress.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)

Related Sample Code
MoveMe
ScrollViewSuite
SimpleGestureRecognizers
Touches
UICatalog

Declared In
`UIView.h`

layerClass

Returns the class used to create the layer for instances of this class.

```
+ (Class)layerClass
```

Return Value
The class used to create the view's layer.

Discussion
Overridden by subclasses to specify a custom class used for rendering. Invoked when creating the underlying layer for a view. The default value is the `CALayer` class object.

Availability
Available in iOS 2.0 and later.

See Also
`@property layer` (page 22)

Related Sample Code
aurioTouch
avTouch
GLImageProcessing
SpeakHere
ZoomingPDFViewer

Declared In
`UIView.h`

setAnimationBeginsFromCurrentState:

Sets whether the animation should begin playing from the current state.

```
+ (void)setAnimationBeginsFromCurrentState:(BOOL)fromCurrentState
```

Parameters

fromCurrentState
`YES` if animations should begin from their currently visible state; otherwise, `NO`.

Discussion
If set to `YES` when an animation is in flight, the current view position of the in-flight animation is used as the starting state for the new animation. If set to `NO`, the in-flight animation ends before the new animation begins using the last view position as the starting state. This method does nothing if an animation is not in flight or invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value is `NO`.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationStartDate:` (page 37)
`+ setAnimationDuration:` (page 34)
`+ setAnimationDelay:` (page 32)
`+ setAnimationCurve:` (page 31)
`+ setAnimationRepeatCount:` (page 36)
`+ setAnimationRepeatAutoreverses:` (page 35)

Declared In
`UIView.h`

setAnimationCurve:

Sets the curve of animating property changes within an animation block.

```
+ (void)setAnimationCurve:(UIViewAnimationCurve)curve
```

Discussion
The animation curve is the relative speed of the animation over its course. This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value of the animation curve is `UIViewAnimationCurveEaseInOut` (page 59).

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationStartDate:` (page 37)
`+ setAnimationDuration:` (page 34)
`+ setAnimationDelay:` (page 32)
`+ setAnimationRepeatCount:` (page 36)
`+ setAnimationRepeatAutoreverses:` (page 35)
`+ setAnimationBeginsFromCurrentState:` (page 31)

Declared In
`UIView.h`

setAnimationDelay:

Sets the delay (in seconds) of animating property changes within an animation block.

```
+ (void)setAnimationDelay:(NSTimeInterval)delay
```

Discussion
This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value of the animation delay is `0.0`.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationStartDate:` (page 37)
`+ setAnimationDuration:` (page 34)
`+ setAnimationCurve:` (page 31)
`+ setAnimationRepeatCount:` (page 36)
`+ setAnimationRepeatAutoreverses:` (page 35)
`+ setAnimationBeginsFromCurrentState:` (page 31)

Declared In
`UIView.h`

setAnimationDelegate:

Sets the delegate for animation messages.

```
+ (void)setAnimationDelegate:(id)delegate
```

Parameters

delegate

The object that receives the delegate messages set using the `setAnimationWillStartSelector:` (page 39) and `setAnimationDidStopSelector:` (page 33) methods.

Discussion

This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value is `nil`.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability

Available in iOS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 29)

`+ commitAnimations` (page 29)

`+ setAnimationWillStartSelector:` (page 39)

`+ setAnimationDidStopSelector:` (page 33)

Related Sample Code

AQOfflineRenderTest

DateCell

iPhoneMixerEQGraphTest

MoveMe

TheElements

Declared In

`UIView.h`

setAnimationDidStopSelector:

Sets the message to send to the animation delegate when animation stops.

```
+ (void)setAnimationDidStopSelector:(SEL)selector
```

Parameters

selector

The message sent to the animation delegate after animations end. The default value is `NULL`. The selector should be of the form: `- (void)animationDidStop:(NSString *)animationID finished:(NSNumber *)finished context:(void *)context`. Your method must take the following arguments:

`animationID`
An `NSString` containing an optional application-supplied identifier. This is the identifier that is passed to the `beginAnimations:context:` (page 29) method. This argument can be `nil`.

`finished`
An `NSNumber` object containing a Boolean value. The value is `YES` if the animation ran to completion before it stopped or `NO` if it did not.

`context`
An optional application-supplied context. This is the context data passed to the `beginAnimations:context:` (page 29) method. This argument can be `nil`.

Discussion

This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value is `NULL`.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability

Available in iOS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationDelegate:` (page 33)
`+ setAnimationWillStartSelector:` (page 39)

Related Sample Code

AQOfflineRenderTest
Breadcrumb
iPhoneMixerEQGraphTest
MoveMe
TheElements

Declared In

`UIView.h`

setAnimationDuration:

Sets the duration (in seconds) of animating property changes within an animation block.

```
+ (void)setAnimationDuration:(NSTimeInterval)duration
```

Parameters

duration

The period over which the animation occurs.

Discussion

This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value is `0.2`.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability

Available in iOS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationStartDate:` (page 37)
`+ setAnimationDelay:` (page 32)
`+ setAnimationCurve:` (page 31)
`+ setAnimationRepeatCount:` (page 36)
`+ setAnimationRepeatAutoreverses:` (page 35)
`+ setAnimationBeginsFromCurrentState:` (page 31)

Related Sample Code

MoveMe
ScrollViewSuite
SimpleGestureRecognizers
Touches
UICatalog

Declared In

`UIView.h`

setAnimationRepeatAutoreverses:

Sets whether the animation of property changes within an animation block automatically reverses repeatedly.

```
+ (void)setAnimationRepeatAutoreverses:(BOOL)repeatAutoreverses
```

Parameters

repeatAutoreverses

If `YES` if the animation automatically reverses repeatedly; if `NO`, it does not.

Discussion

Autoreverses is when the animation plays backward after playing forward and similarly plays forward after playing backward. Use the `setAnimationRepeatCount:` (page 36) class method to specify the number of times the animation autoreverses. This method does nothing if the repeat count is zero or this method is invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. The default value is `NO`.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationStartDate:` (page 37)
`+ setAnimationDuration:` (page 34)
`+ setAnimationDelay:` (page 32)
`+ setAnimationCurve:` (page 31)
`+ setAnimationRepeatCount:` (page 36)
`+ setAnimationBeginsFromCurrentState:` (page 31)

Declared In
`UIView.h`

setAnimationRepeatCount:

Sets the number of times animations within an animation block repeat.

```
+ (void)setAnimationRepeatCount:(float)repeatCount
```

Parameters

`repeatCount`
The number of times animations repeat. This value can be a fraction.

Discussion
This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block. By default, animations don't repeat.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationStartDate:` (page 37)
`+ setAnimationDuration:` (page 34)
`+ setAnimationDelay:` (page 32)
`+ setAnimationCurve:` (page 31)
`+ setAnimationRepeatAutoreverses:` (page 35)
`+ setAnimationBeginsFromCurrentState:` (page 31)

Declared In
`UIView.h`

setAnimationsEnabled:

Sets whether animations are enabled.

```
+ (void)setAnimationsEnabled:(BOOL)enabled
```

Parameters

`enabled`
If `YES`, animations are enabled; if `NO`, they are not.

Discussion
Animation attribute changes are ignored when animations are disabled. By default, animations are enabled.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ areAnimationsEnabled` (page 28)

Related Sample Code
AdvancedTableViewCells
Table View Animations and Gestures

Declared In
`UIView.h`

setAnimationStartDate:

Sets the start time of animating property changes within an animation block.

```
+ (void)setAnimationStartDate:(NSDate *)startTime
```

Parameters

`startTime`
The time to begin the animations.

Discussion
Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block.

The default start time is the value returned by the `CFAbsoluteTimeGetCurrent` function.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
+ `beginAnimations:context:` (page 29)
+ `commitAnimations` (page 29)
+ `setAnimationDuration:` (page 34)
+ `setAnimationDelay:` (page 32)
+ `setAnimationCurve:` (page 31)
+ `setAnimationRepeatCount:` (page 36)
+ `setAnimationRepeatAutoreverses:` (page 35)
+ `setAnimationBeginsFromCurrentState:` (page 31)

Declared In
`UIView.h`

setAnimationTransition:forView:cache:

Sets a transition to apply to a view during an animation block.

```
+ (void)setAnimationTransition:(UIViewAnimationTransition)transition forView:(UIView
    *)view cache:(BOOL)cache
```

Parameters

transition
A transition to apply to *view*. Possible values are described in UIViewAnimationTransition (page 63).

view
The view to apply the transition to.

cache
If `YES`, the before and after images of *view* are rendered once and used to create the frames in the animation. Caching can improve performance but if you set this parameter to `YES`, you must not update the view or its subviews during the transition. Updating the view and its subviews may interfere with the caching behaviors and cause the view contents to be rendered incorrectly (or in the wrong location) during the animation. You must wait until the transition ends to update the view.

If `NO`, the view and its contents must be updated for each frame of the transition animation, which may noticeably affect the frame rate.

Discussion
If you want to change the appearance of a view during a transition—for example, flip from one view to another—then use a container view, an instance of `UIView`, as follows:

1. Begin an animation block.
2. Set the transition on the container view.
3. Remove the subview from the container view.
4. Add the new subview to the container view.
5. Commit the animation block.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AQOfflineRenderTest
BubbleLevel
iPhoneMixerEQGraphTest
TheElements
UICatalog

Declared In
`UIView.h`

setAnimationWillStartSelector:

Sets the message to send to the animation delegate when animation starts.

```
+ (void)setAnimationWillStartSelector:(SEL)selector
```

Parameters

`selector`
The message sent to the animation delegate before animations start. The default value is `NULL`. The selector should have the same arguments as the `beginAnimations:context:` (page 29) method, an optional application-supplied identifier and context. Both of these arguments can be `nil`.

Discussion
This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 29) class method to start and the `commitAnimations` (page 29) class method to end an animation block.

Use of this method is discouraged in iOS 4.0 and later. You should use the block-based animation methods instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 29)
`+ commitAnimations` (page 29)
`+ setAnimationDelegate:` (page 33)
`+ setAnimationDidStopSelector:` (page 33)

Declared In
`UIView.h`

transitionFromView:toView:duration:options:completion:

Creates a transition animation between the specified views using the given parameters.

```
+ (void)transitionFromView:(UIView *)fromView toView:(UIView *)toView
    duration:(NSTimeInterval)duration options:(UIViewAnimationOptions)options
    completion:(void (^)(BOOL finished))completion
```

Parameters

fromView
: The starting view for the transition. By default, this view is removed from its parent view as part of the transition.

toView
: The ending view for the transition. By default, this view is added to the parent of *fromView* as part of the transition.

duration
: The duration of the transition animation, measured in seconds. If you specify a negative value or `0`, the transition is made without animations.

options
: A mask of options indicating how you want to perform the animations. For a list of valid constants, see `UIViewAnimationOptions` (page 63).

completion
: A block object to be executed when the animation sequence ends. This block has no return value and takes a single Boolean argument that indicates whether or not the animations actually finished before the completion handler was called. If the duration of the animation is 0, this block is performed at the beginning of the next run loop cycle. This parameter may be `NULL`.

Discussion

This method provides a simple way to transition from the view in the *fromView* parameter to the view in the *toView* parameter. By default, the view in *fromView* is replaced in the view hierarchy by the view in *toView*. If both views are already part of your view hierarchy, you can include the `UIViewAnimationOptionShowHideTransitionViews` (page 65) option in the *options* parameter to simply hide or show them.

The view transition starts immediately unless another animation is already in-flight, in which case it starts immediately after the current animation finishes.

Availability

Available in iOS 4.0 and later.

Declared In

`UIView.h`

transitionWithView:duration:options:animations:completion:

Creates a transition animation for the specified container view.

```
+ (void)transitionWithView:(UIView *)view duration:(NSTimeInterval)duration
    options:(UIViewAnimationOptions)options animations:(void (^)(void))animations
     completion:(void (^)(BOOL finished))completion
```

Parameters

view
: The container view that contains the views involved in the transition.

duration
: The duration of the transition animation, measured in seconds. If you specify a negative value or `0`, the transition is made without animations.

options
: A mask of options indicating how you want to perform the animations. For a list of valid constants, see `UIViewAnimationOptions` (page 63).

animations
: A block object that adds or removes the views involved in the transition. This block takes no parameters and has no return value. This parameter must not be `NULL`.

completion
: A block object to be executed when the animation sequence ends. This block has no return value and takes a single Boolean argument that indicates whether or not the animations actually finished before the completion handler was called. If the duration of the animation is 0, this block is performed at the beginning of the next run loop cycle. This parameter may be `NULL`.

Discussion

You can use this method to create your own view transition animations. The block you specify in the *animations* parameter should add or remove the relevant views from your view hierarchy. (Alternatively, if you do not want to add and remove views, you can simply hide or show them.) Because you specify a custom block, you can add or remove any number of views as part of the transitions. Of course, all views in the animation block share the animation parameters passed to this method.

For example, to implement a flip transition between two views in the same container view, you could use code similar to the following:

```
[UIView transitionWithView:containerView
           duration:0.2
           options:UIViewAnimationOptionTransitionFlipFromLeft
           animations:^{ [fromView removeFromSuperview]; [containerView
addSubview:toView] }
           completion:NULL];
```

Availability

Available in iOS 4.0 and later.

Declared In

`UIView.h`

Instance Methods

addGestureRecognizer:

Attaches a gesture recognizer to the receiving view.

```
- (void)addGestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
```

Parameters

gestureRecognizer
: An instance of a subclass of `UIGestureRecognizer`. This parameter must not be `nil`.

Discussion

Attaching a gesture recognizer to a view defines the scope of the represented gesture, causing it to receive touches hit-tested to that view and all of its subviews. The view retains the gesture recognizer.

Availability
Available in iOS 3.2 and later.

See Also
- `removeGestureRecognizer:` (page 54)
  `@property gestureRecognizers` (page 21)

Related Sample Code
ScrollViewSuite
Table View Animations and Gestures
Touches

Declared In
`UIView.h`

addSubview:

Adds a view to the receiver's subviews so it's displayed above its siblings.

```
- (void)addSubview:(UIView *)view
```

Discussion
This method also sets the receiver as the next responder of *`view`*. The receiver retains *`view`*. If you use `removeFromSuperview` (page 53) to remove *`view`* from the view hierarchy, *`view`* is released. If you want to keep using *`view`* after removing it from the view hierarchy (if, for example, you are swapping through a number of views), you must retain it before invoking `removeFromSuperview`.

Views can have only one superview. If the superview of *`view`* is not `nil` and is not the same as the current view, this method removes it from the previous superview before making it a subview of the current view.

Availability
Available in iOS 2.0 and later.

See Also
- `insertSubview:atIndex:` (page 50)
- `insertSubview:aboveSubview:` (page 50)
- `insertSubview:belowSubview:` (page 51)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 48)

Related Sample Code
iPhoneCoreDataRecipes
PhotoScroller
SeismicXML
TableViewSuite
WiTap

Declared In
`UIView.h`

bringSubviewToFront:

Moves the specified subview to the front of its siblings.

```
- (void)bringSubviewToFront:(UIView *)view
```

Parameters

view
: The subview to move to the front.

Availability

Available in iOS 2.0 and later.

See Also

- `sendSubviewToBack:` (page 54)

Related Sample Code

ScrollViewSuite

Declared In

`UIView.h`

convertPoint:fromView:

Converts a point from the coordinate system of a given view to that of the receiver.

```
- (CGPoint)convertPoint:(CGPoint)point fromView:(UIView *)view
```

Parameters

point
: A point specifying a location in the coordinate system of *view*.

view
: The view with *point* in its coordinate system. If *view* is `nil`, this method instead converts from window base coordinates. Otherwise, both *view* and the receiver must belong to the same `UIWindow` object.

Return Value

The point converted to the coordinate system of the receiver.

Availability

Available in iOS 2.0 and later.

See Also

- `convertPoint:toView:` (page 44)
- `convertRect:toView:` (page 45)
- `convertRect:fromView:` (page 44)

Related Sample Code

PhotoScroller

Declared In

`UIView.h`

convertPoint:toView:

Converts a point from the receiver's coordinate system to that of a given view.

```
- (CGPoint)convertPoint:(CGPoint)point toView:(UIView *)view
```

Parameters

point

A point specifying a location in the coordinate system of the receiver.

view

The view into whose coordinate system *point* is to be converted. If *view* is `nil`, this method instead converts to window base coordinates. Otherwise, both *view* and the receiver must belong to the same `UIWindow` object.

Return Value

The point converted to the coordinate system of *view*.

Availability

Available in iOS 2.0 and later.

See Also

- `convertPoint:fromView:` (page 43)
- `convertRect:toView:` (page 45)
- `convertRect:fromView:` (page 44)

Related Sample Code

PhotoScroller

ScrollViewSuite

Declared In

`UIView.h`

convertRect:fromView:

Converts a rectangle from the coordinate system of another view to that of the receiver.

```
- (CGRect)convertRect:(CGRect)rect fromView:(UIView *)view
```

Parameters

rect

The rectangle in `view`'s coordinate system.

view

The view with *rect* in its coordinate system. If *view* is `nil`, this method instead converts from window base coordinates. Otherwise, both *view* and the receiver must belong to the same `UIWindow` object.

Return Value

The converted rectangle.

Availability

Available in iOS 2.0 and later.

See Also

- `convertPoint:toView:` (page 44)

- `convertPoint:fromView:` (page 43)
- `convertRect:toView:` (page 45)

Declared In
`UIView.h`

convertRect:toView:

Converts a rectangle from the receiver's coordinate system to that of another view.

```
- (CGRect)convertRect:(CGRect)rect toView:(UIView *)view
```

Parameters

`rect`
A rectangle in the receiver's coordinate system.

`view`
The view that is the target of the conversion operation. If *`view`* is `nil`, this method instead converts to window base coordinates. Otherwise, both *`view`* and the receiver must belong to the same `UIWindow` object.

Return Value
The converted rectangle.

Availability
Available in iOS 2.0 and later.

See Also
- `convertPoint:toView:` (page 44)
- `convertPoint:fromView:` (page 43)
- `convertRect:fromView:` (page 44)

Declared In
`UIView.h`

didAddSubview:

Tells the view when subviews are added.

```
- (void)didAddSubview:(UIView *)subview
```

Parameters

`subview`
The view that was added as a subview.

Discussion
Overridden by subclasses to perform additional actions when subviews are added to the receiver. This method is invoked by `addSubview:` (page 42).

Availability
Available in iOS 2.0 and later.

See Also

- `willRemoveSubview:` (page 58)
- `addSubview:` (page 42)

Declared In

`UIView.h`

didMoveToSuperview

Informs the receiver that its superview has changed (possibly to `nil`).

```
- (void)didMoveToSuperview
```

Discussion

The default implementation does nothing; subclasses can override this method to perform whatever actions are necessary.

Availability

Available in iOS 2.0 and later.

See Also

- `willMoveToSuperview:` (page 58)

Declared In

`UIView.h`

didMoveToWindow

Informs the receiver that it has been added to a window.

```
- (void)didMoveToWindow
```

Discussion

The default implementation does nothing; subclasses can override this method to perform whatever actions are necessary.

The `window` (page 26) property may be `nil` when this method is invoked, indicating that the receiver does not currently reside in any window. This occurs when the receiver has just been removed from its superview or when the receiver has just been added to a superview that is not attached to a window. Overrides of this method may choose to ignore such cases if they are not of interest.

Availability

Available in iOS 2.0 and later.

See Also

- `willMoveToWindow:` (page 58)

Declared In

`UIView.h`

drawRect:

Draws the receiver's image within the passed-in rectangle.

```
- (void)drawRect:(CGRect)rect
```

Parameters

`rect`
: A rectangle defining the area to restrict drawing to.

Discussion

Subclasses override this method if they actually draw their views. Subclasses need not override this method if the subclass is a container for other views. The default implementation does nothing. If your custom view is a direct `UIView` subclass, you do not need to call the implementation of `super`. Note that it is the responsibility of each subclass to totally fill *`rect`* if its superclass's implementation actually draws and `opaque` (page 23) is `YES`.

When this method is invoked, the receiver can assume the coordinate transformations of its frame and bounds rectangles have been applied; all it needs to do is invoke rendering client functions. Use the `UIGraphicsGetCurrentContext` function to get the current graphics context for drawing that also has the coordinate origin in the upper-left corner. Do not retain the graphics context since it can change between calls to the `drawRect:` method.

Availability

Available in iOS 2.0 and later.

See Also

`- setNeedsDisplay` (page 54)
`- setNeedsDisplayInRect:` (page 55)
`@property contentMode` (page 18)

Declared In

`UIView.h`

endEditing:

Causes the view (or one of its embedded text fields) to resign the first responder status.

```
- (BOOL)endEditing:(BOOL)force
```

Parameters

`force`
: If `YES`, force the first responder to resign, regardless of whether it wants to do so.

Return Value

`YES` if the view resigned the first responder status or `NO` if it did not.

Discussion

This method looks at the view and its subview hierarchy for a text field that is currently the first responder. If it finds one, it asks that text field to resign as first responder. If the *`force`* parameter is set to `YES`, the text field is never even asked; it is forced to resign.

Availability

Available in iOS 2.0 and later.

Related Sample Code
Formulaic

TaggedLocations

Declared In
`UITextField.h`

exchangeSubviewAtIndex:withSubviewAtIndex:

Exchanges the subviews in the receiver at the given indices.

```
- (void)exchangeSubviewAtIndex:(NSInteger)index1
    withSubviewAtIndex:(NSInteger)index2
```

Parameters

index1
The index of the subview with which to replace the subview at index *index2*.

index2
The index of the subview with which to replace the subview at index *index1*.

Availability
Available in iOS 2.0 and later.

See Also
- `addSubview:` (page 42)
- `insertSubview:atIndex:` (page 50)
- `insertSubview:aboveSubview:` (page 50)
- `insertSubview:belowSubview:` (page 51)

Declared In
`UIView.h`

hitTest:withEvent:

Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains a specified point.

```
- (UIView *)hitTest:(CGPoint)point withEvent:(UIEvent *)event
```

Parameters

point
A point that is in the receiver's coordinate system.

event
The event that triggered this method or `nil` if this method is invoked programmatically.

Return Value
A view object that is the farthest descendent of *point*. Returns `nil` if the point lies completely outside the receiver.

Discussion
This method traverses the view hierarchy by sending the `pointInside:withEvent:` (page 53) message to each subview to determine which subview should receive a touch event. If `pointInside:withEvent:` (page 53) returns `YES`, then the subview's hierarchy is traversed; otherwise, its branch of the view hierarchy is ignored. You rarely need to invoke this method, but you might override it to hide touch events from subviews.

This method ignores view objects that are hidden, that have disabled user interaction, or have an alpha level less than 0.01. This method does not take the view's content into account when determining a hit. Thus, a view can still be returned even if the specified point is in a transparent portion of that view's content.

Availability
Available in iOS 2.0 and later.

See Also
`– pointInside:withEvent:` (page 53)

Declared In
`UIView.h`

initWithFrame:

Initializes and returns a newly allocated view object with the specified frame rectangle.

```
- (id)initWithFrame:(CGRect)aRect
```

Parameters

`aRect`
The frame rectangle for the view, measured in points. The origin of the frame is relative to the superview in which you plan to add it. This method uses the frame rectangle to set the `center` (page 17) and `bounds` (page 16) properties accordingly.

Return Value
An initialized view object or `nil` if the object couldn't be created.

Discussion
The new view object must be inserted into the view hierarchy of a window before it can be used. If you create a view object programmatically, this method is the designated initializer for the `UIView` class.

If you use Interface Builder to design your interface, this method is not called when your view objects are subsequently loaded from the nib file. Objects in a nib file are reconstituted and then initialized using their `initWithCoder:` method, which modifies the attributes of the view to match the attributes stored in the nib file. For detailed information about how views are loaded from a nib file, see *Resource Programming Guide*.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AccelerometerGraph
BubbleLevel
Table View Animations and Gestures

Declared In
`UIView.h`

insertSubview:aboveSubview:

Inserts a view above another view in the view hierarchy.

```
- (void)insertSubview:(UIView *)view  aboveSubview:(UIView *)siblingSubview
```

Parameters

view

The view to insert above another view. It's removed from its superview if it's not a sibling of `siblingSubview`.

siblingSubview

The sibling view that will be behind the inserted view.

Discussion

Views can have only one superview. If the superview of *view* is not `nil` and is not the same as the current view, this method removes it from the previous superview before making it a subview of the current view.

Availability

Available in iOS 2.0 and later.

See Also

- `addSubview:` (page 42)
- `insertSubview:atIndex:` (page 50)
- `insertSubview:belowSubview:` (page 51)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 48)

Declared In

`UIView.h`

insertSubview:atIndex:

Inserts a subview at the specified index.

```
- (void)insertSubview:(UIView *)view  atIndex:(NSInteger)index
```

Parameters

view

The view to insert. This value cannot be `nil`.

index

Subview indices start at `0` and cannot be greater than the number of subviews.

Discussion

Views can have only one superview. If the superview of *view* is not `nil` and is not the same as the current view, this method removes it from the previous superview before making it a subview of the current view.

Availability

Available in iOS 2.0 and later.

See Also

- `addSubview:` (page 42)
- `insertSubview:aboveSubview:` (page 50)
- `insertSubview:belowSubview:` (page 51)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 48)

Declared In
`UIView.h`

insertSubview:belowSubview:

Inserts a view below another view in the view hierarchy.

```
- (void)insertSubview:(UIView *)view  belowSubview:(UIView *)siblingSubview
```

Parameters

`view`
: The view to insert below another view. It's removed from its superview if it's not a sibling of *`siblingSubview`*.

`siblingSubview`
: The sibling view that will be above the inserted view.

Discussion
Views can have only one superview. If the superview of *`view`* is not `nil` and is not the same as the current view, this method removes it from the previous superview before making it a subview of the current view.

Availability
Available in iOS 2.0 and later.

See Also
`– addSubview:` (page 42)
`– insertSubview:atIndex:` (page 50)
`– insertSubview:aboveSubview:` (page 50)
`– exchangeSubviewAtIndex:withSubviewAtIndex:` (page 48)

Declared In
`UIView.h`

isDescendantOfView:

Returns a Boolean value indicating whether the receiver is a subview of a given view or whether it is identical to that view.

```
- (BOOL)isDescendantOfView:(UIView *)view
```

Parameters

`view`
: The view to test for subview relationship within the view hierarchy.

Return Value
`YES` if the receiver is an immediate or distant subview of *`view`*, or if *`view`* is the receiver; otherwise `NO`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

layoutIfNeeded

Lays out the subviews if needed.

```
- (void)layoutIfNeeded
```

Discussion
Use this method to force the layout of subviews before drawing. Starting with the receiver, this method traverses upward through the view hierarchy as long as superviews require layout. Then it lays out the entire tree beneath that ancestor. Therefore, calling this method can potentially force the layout of your entire view hierarchy. The `UIView` implementation of this calls the equivalent `CALayer` method and so has the same behavior as `CALayer`.

Availability
Available in iOS 2.0 and later.

See Also
`- setNeedsLayout` (page 55)
`- layoutSubviews` (page 52)

Declared In
`UIView.h`

layoutSubviews

Lays out subviews.

```
- (void)layoutSubviews
```

Discussion
Overridden by subclasses to layout subviews when `layoutIfNeeded` (page 52) is invoked. The default implementation of this method does nothing.

Availability
Available in iOS 2.0 and later.

See Also
`- setNeedsLayout` (page 55)
`- layoutIfNeeded` (page 52)

Related Sample Code
aurioTouch
GLImageProcessing
SpeakHere
TouchCells
ZoomingPDFViewer

Declared In
`UIView.h`

pointInside:withEvent:

Returns a Boolean value indicating whether the receiver contains the specified point.

```
- (BOOL)pointInside:(CGPoint)point withEvent:(UIEvent *)event
```

Parameters

`point`
A point that is in the receiver's coordinate system.

`event`
The event that triggered this method or `nil` if this method is invoked programmatically.

Return Value

`YES` if *`point`* is inside the receiver's bounds; otherwise, `NO`.

Availability

Available in iOS 2.0 and later.

See Also

`– hitTest:withEvent:` (page 48)

Declared In

`UIView.h`

removeFromSuperview

Unlinks the receiver from its superview and its window, and removes it from the responder chain.

```
- (void)removeFromSuperview
```

Discussion

If the receiver's superview is not `nil`, this method releases the receiver. If you plan to reuse the view, be sure to retain it before calling this method and be sure to release it as appropriate when you are done with it or after adding it to another view hierarchy.

Never invoke this method while displaying.

Availability

Available in iOS 2.0 and later.

See Also

`@property superview` (page 24)
`@property subviews` (page 23)

Related Sample Code

iPhoneCoreDataRecipes
PhotoScroller
SpeakHere
TheElements
UICatalog

Declared In

`UIView.h`

removeGestureRecognizer:

Detaches a gesture recognizer from the receiving view.

```
- (void)removeGestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
```

Parameters

gestureRecognizer

An instance of a subclass of the abstract base class `UIGestureRecognizer`.

Discussion

When you remove a gesture recognizer from the view its retain count is decremented.

Availability

Available in iOS 3.2 and later.

See Also

`- addGestureRecognizer:` (page 41)
`@property gestureRecognizers` (page 21)

Declared In

`UIView.h`

sendSubviewToBack:

Moves the specified subview to the back of its siblings.

```
- (void)sendSubviewToBack:(UIView *)view
```

Parameters

view

The subview to move to the back.

Availability

Available in iOS 2.0 and later.

See Also

`- bringSubviewToFront:` (page 43)

Declared In

`UIView.h`

setNeedsDisplay

Controls whether the receiver's entire bounds rectangle is marked as needing display.

```
- (void)setNeedsDisplay
```

Discussion

By default, geometry changes to a view automatically redisplays the view without needing to invoke the `drawRect:` (page 47) method. Therefore, you need to request that a view redraw only when the data or state used for drawing a view changes. In this case, send the view the `setNeedsDisplay` (page 54) message. Any `UIView` objects marked as needing display are automatically redisplayed when the application returns to the run loop.

Availability
Available in iOS 2.0 and later.

See Also
- `drawRect:` (page 47)
- `setNeedsDisplayInRect:` (page 55)

`@property contentMode` (page 18)

Related Sample Code
avTouch

MoveMe

SpeakHere

TableViewSuite

Teslameter

Declared In
`UIView.h`

setNeedsDisplayInRect:

Marks the region of the receiver within the specified rectangle as needing display, increasing the receiver's existing invalid region to include it.

```
- (void)setNeedsDisplayInRect:(CGRect)invalidRect
```

Parameters

`invalidRect`
The rectangular region of the receiver to mark as invalid; it should be specified in the coordinate system of the receiver.

Discussion
By default, geometry changes to a view automatically redisplays the view without needing to invoke the `drawRect:` (page 47) method. Therefore, you need to request that a view or a region of a view redraw only when the data or state used for drawing a view changes. Use this method or the `setNeedsDisplay` (page 54) method to mark a view as needing display.

Availability
Available in iOS 2.0 and later.

See Also
- `drawRect:` (page 47)
- `setNeedsDisplay` (page 54)

`@property contentMode` (page 18)

Declared In
`UIView.h`

setNeedsLayout

Sets whether subviews need to be rearranged before displaying.

```
- (void)setNeedsLayout
```

Discussion
If you invoke this method before the next display operation, then `layoutIfNeeded` (page 52) lays out the subviews; otherwise, it does not.

Availability
Available in iOS 2.0 and later.

See Also
- `layoutIfNeeded` (page 52)
- `layoutSubviews` (page 52)

Related Sample Code
ScrollViewSuite

Declared In
`UIView.h`

sizeThatFits:

Asks the view to calculate and return the size that best fits its subviews.

```
- (CGSize)sizeThatFits:(CGSize)size
```

Parameters

`size`
The current size of the receiver.

Return Value
A new size that fits the receiver’s subviews.

Discussion
The default implementation of this method simply returns the value in the *`size`* parameter. However, subclasses can override this method to return a custom value based on the desired layout of any subviews. For example, a `UISwitch` object returns a fixed size value that represents the standard size of a switch view, and a `UIImageView` object returns the size of the image it is currently displaying.

This method does not resize the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `sizeToFit` (page 57)
  `@property frame` (page 20)
  `@property bounds` (page 16)

Declared In
`UIView.h`

sizeToFit

Resizes and moves the receiver view so it just encloses its subviews.

```
- (void)sizeToFit
```

Discussion
Call this method when you want to resize the current view so that it uses the most appropriate amount of space. Specific UIKit views size themselves according to their own internal needs. In some cases, if a view does not have a superview, it may size itself to the screen bounds. Thus, if you want a given view to size itself to its parent view, you should add it to the parent view before calling this method.

You should not override this method. If you want to change the default sizing information for your view, override the `sizeThatFits:` instead. That method performs any needed calculations and returns them to this method, which then makes the change.

Availability
Available in iOS 2.0 and later.

See Also
`- sizeThatFits:` (page 56)

Related Sample Code
BonjourWeb
WiTap

Declared In
`UIView.h`

viewWithTag:

Returns the view with the specified tag.

```
- (UIView *)viewWithTag:(NSInteger)tag
```

Parameters
`tag`
The tag used to search for the view.

Return Value
The view in the receiver's hierarchy that matches *`tag`*. The receiver is included in the search.

Availability
Available in iOS 2.0 and later.

See Also
`@property tag` (page 24)

Related Sample Code
GenericKeychain
SeismicXML
UICatalog

Declared In
`UIView.h`

willMoveToSuperview:

Informs the receiver that its superview is about to change to the specified superview (which may be `nil`).

```
- (void)willMoveToSuperview:(UIView *)newSuperview
```

Parameters

`newSuperview`
A view object that will be the new superview of the receiver.

Discussion
Subclasses can override this method to perform whatever actions are necessary.

Availability
Available in iOS 2.0 and later.

See Also
`- didMoveToSuperview` (page 46)

Declared In
`UIView.h`

willMoveToWindow:

Informs the receiver that it's being added to the view hierarchy of the specified window object (which may be `nil`).

```
- (void)willMoveToWindow:(UIWindow *)newWindow
```

Parameters

`newWindow`
A window object that will be at the root of the receiver's new view hierarchy.

Discussion
Subclasses can override this method to perform whatever actions are necessary.

Availability
Available in iOS 2.0 and later.

See Also
`- didMoveToWindow` (page 46)

Declared In
`UIView.h`

willRemoveSubview:

Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

```
- (void)willRemoveSubview:(UIView *)subview
```

Parameters

`subview`

The subview that will be removed.

Discussion

This method is invoked when *`subview`* receives a `removeFromSuperview` (page 53) message or *`subview`* is removed from the receiver because it is being added to another view.

Availability

Available in iOS 2.0 and later.

See Also

`– didAddSubview:` (page 45)

`– addSubview:` (page 42)

Declared In

`UIView.h`

Constants

UIViewAnimationCurve

Specifies the animation curve. For example, specifies whether animation changes speed at the beginning or end.

```
typedef enum {
    UIViewAnimationCurveEaseInOut,
    UIViewAnimationCurveEaseIn,
    UIViewAnimationCurveEaseOut,
    UIViewAnimationCurveLinear
} UIViewAnimationCurve;
```

Constants

`UIViewAnimationCurveEaseInOut`

An ease-in ease-out curve causes the animation to begin slowly, accelerate through the middle of its duration, and then slow again before completing.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationCurveEaseIn`

An ease-in curve causes the animation to begin slowly, and then speed up as it progresses.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationCurveEaseOut`

An ease-out curve causes the animation to begin quickly, and then slow as it completes.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationCurveLinear`
: A linear animation curve causes an animation to occur evenly over its duration.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

UIViewContentMode

Specifies how a view resizes its subviews when its size changes.

```
typedef enum {
    UIViewContentModeScaleToFill,
    UIViewContentModeScaleAspectFit,
    UIViewContentModeScaleAspectFill,
    UIViewContentModeRedraw,
    UIViewContentModeCenter,
    UIViewContentModeTop,
    UIViewContentModeBottom,
    UIViewContentModeLeft,
    UIViewContentModeRight,
    UIViewContentModeTopLeft,
    UIViewContentModeTopRight,
    UIViewContentModeBottomLeft,
    UIViewContentModeBottomRight,
} UIViewContentMode;
```

Constants

`UIViewContentModeScaleToFill`
: Scales the content to fit the size of itself by changing the aspect ratio of the content if necessary.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeScaleAspectFit`
: Scales the content to fit the size of the view by maintaining the aspect ratio. Any remaining area of the view's bounds is transparent.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeScaleAspectFill`
: Scales the content to fill the size of the view. Some portion of the content may be clipped to fill the view's bounds.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeRedraw`
: Redisplays the view when the bounds change by invoking the `setNeedsDisplay` (page 54) method.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeCenter`
: Centers the content in the view's bounds, keeping the proportions the same.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeTop`
: Centers the content aligned at the top in the view's bounds.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeBottom`
: Centers the content aligned at the bottom in the view's bounds.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeLeft`
: Aligns the content on the left of the view.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeRight`
: Aligns the content on the right of the view.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeTopLeft`
: Aligns the content in the top-left corner of the view.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeTopRight`
: Aligns the content in the top-right corner of the view.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeBottomLeft`
: Aligns the content in the bottom-left corner of the view.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

`UIViewContentModeBottomRight`
: Aligns the content in the bottom-right corner of the view.

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

UIViewAutoresizing

Specifies how a view is automatically resized.

```
enum {
    UIViewAutoresizingNone                 = 0,
    UIViewAutoresizingFlexibleLeftMargin   = 1 << 0,
    UIViewAutoresizingFlexibleWidth        = 1 << 1,
    UIViewAutoresizingFlexibleRightMargin  = 1 << 2,
    UIViewAutoresizingFlexibleTopMargin    = 1 << 3,
    UIViewAutoresizingFlexibleHeight       = 1 << 4,
    UIViewAutoresizingFlexibleBottomMargin = 1 << 5
};
typedef NSUInteger UIViewAutoresizing;
```

Constants

`UIViewAutoresizingNone`

The view does not resize.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleLeftMargin`

The view resizes by expanding or shrinking in the direction of the left margin.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleWidth`

The view resizes by expanding or shrinking its width.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleRightMargin`

The view resizes by expanding or shrinking in the direction of the right margin.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleTopMargin`

The view resizes by expanding or shrinking in the direction of the top margin.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleHeight`

The view resizes by expanding or shrinking its height.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleBottomMargin`

The view resizes by expanding or shrinking in the direction of the bottom margin.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

Availability

Available in iOS 2.0 and later.

Declared In
`UIView.h`

UIViewAnimationTransition

Specifies a transition to apply to a view in an animation block.

```
typedef enum {
    UIViewAnimationTransitionNone,
    UIViewAnimationTransitionFlipFromLeft,
    UIViewAnimationTransitionFlipFromRight
    UIViewAnimationTransitionCurlUp,
    UIViewAnimationTransitionCurlDown,
} UIViewAnimationTransition;
```

Constants

`UIViewAnimationTransitionNone`

No transition specified.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionFlipFromLeft`

A transition that flips a view around a vertical axis from left to right. The left side of the view moves towards the front and right side towards the back.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionFlipFromRight`

A transition that flips a view around a vertical axis from right to left. The right side of the view moves towards the front and left side towards the back.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionCurlUp`

A transition that curls a view up from the bottom.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionCurlDown`

A transition that curls a view down from the top.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

UIViewAnimationOptions

Specifies options for animating views with blocks.

```
enum {
    UIViewAnimationOptionLayoutSubviews            = 1 <<  0,
    UIViewAnimationOptionAllowUserInteraction      = 1 <<  1,
    UIViewAnimationOptionBeginFromCurrentState     = 1 <<  2,
    UIViewAnimationOptionRepeat                    = 1 <<  3,
    UIViewAnimationOptionAutoreverse               = 1 <<  4,
    UIViewAnimationOptionOverrideInheritedDuration = 1 <<  5,
    UIViewAnimationOptionOverrideInheritedCurve    = 1 <<  6,
    UIViewAnimationOptionAllowAnimatedContent      = 1 <<  7,
    UIViewAnimationOptionShowHideTransitionViews   = 1 <<  8,

    UIViewAnimationOptionCurveEaseInOut            = 0 << 16,
    UIViewAnimationOptionCurveEaseIn               = 1 << 16,
    UIViewAnimationOptionCurveEaseOut              = 2 << 16,
    UIViewAnimationOptionCurveLinear               = 3 << 16,

    UIViewAnimationOptionTransitionNone            = 0 << 20,
    UIViewAnimationOptionTransitionFlipFromLeft    = 1 << 20,
    UIViewAnimationOptionTransitionFlipFromRight   = 2 << 20,
    UIViewAnimationOptionTransitionCurlUp          = 3 << 20,
    UIViewAnimationOptionTransitionCurlDown        = 4 << 20,
};
typedef NSUInteger UIViewAnimationOptions;
```

Constants

`UIViewAnimationOptionLayoutSubviews`

Lay out subviews at commit time so that they are animated along with their parent.

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionAllowUserInteraction`

Allow the user to interact with views while they are being animated.

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionBeginFromCurrentState`

Start the animation from the current setting associated with an already in-flight animation. If this key is not present, any in-flight animations are allowed to finish before the new animation is started. If another animation is not in flight, this key has no effect.

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionRepeat`

Repeat the animation indefinitely.

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionAutoreverse`

Run the animation backwards and forwards. Must be combined with the `UIViewAnimationOptionRepeat` option.

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionOverrideInheritedDuration`
: Force the animation to use the original duration value specified when the animation was submitted. If this key is not present, the animation inherits the remaining duration of the in-flight animation, if any.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionOverrideInheritedCurve`
: Force the animation to use the original curve value specified when the animation was submitted. If this key is not present, the animation inherits the curve of the in-flight animation, if any.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionAllowAnimatedContent`
: Animate the views by changing the property values dynamically and redrawing the view. If this key is not present, the views are animated using a snapshot image.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionShowHideTransitionViews`
: When present, this key causes views to be hidden or shown (instead of removed or added) when performing a view transition. Both views must already be present in the parent view's hierarchy when using this key. If this key is not present, the to-view in a transition is added to, and the from-view is removed from, the parent view's list of subviews.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionCurveEaseInOut`
: An ease-in ease-out curve causes the animation to begin slowly, accelerate through the middle of its duration, and then slow again before completing.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionCurveEaseIn`
: An ease-in curve causes the animation to begin slowly, and then speed up as it progresses.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionCurveEaseOut`
: An ease-out curve causes the animation to begin quickly, and then slow as it completes.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionCurveLinear`
: A linear animation curve causes an animation to occur evenly over its duration.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionTransitionNone`
: No transition is specified.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionTransitionFlipFromLeft`
: A transition that flips a view around a vertical axis from left to right. The left side of the view moves towards the front and right side towards the back.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionTransitionFlipFromRight`
: A transition that flips a view around a vertical axis from right to left. The right side of the view moves towards the front and left side towards the back.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionTransitionCurlUp`
: A transition that curls a view up from the bottom.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionTransitionCurlDown`
: A transition that curls a view down from the top.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

Availability
Available in iOS 4.0 and later.

Declared In
`UIView.h`

Document Revision History

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

Date	Notes
2010-06-04	Updated to include symbols introduced in iOS 4.0.
2010-02-25	Added descriptions of property and methods related to gesture recognizers.
2009-11-17	Corrected description of the convertRect:toView: method.
2009-07-23	Updated the information about view size restrictions to account for changes in iOS 3.0.
2009-06-15	Updated the description of caching behaviors for the setAnimationTransition:forView:cache: method. Also updated the description of the behavior of the layoutIfNeeded method.
2009-03-05	Updated for iOS 3.0.
2008-11-13	Updated animationDidEndSelector: documentation.
2008-09-09	Corrected description of background property.
2008-07-05	New document that describes the superclass providing concrete subclasses with a structure for drawing and handling events in views.