UIKit Framework Reference

Cocoa Touch Layer: UIKit

2008-05-18

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

Simultaneously published in the United States and Canada.

Contents

Figures and Tables

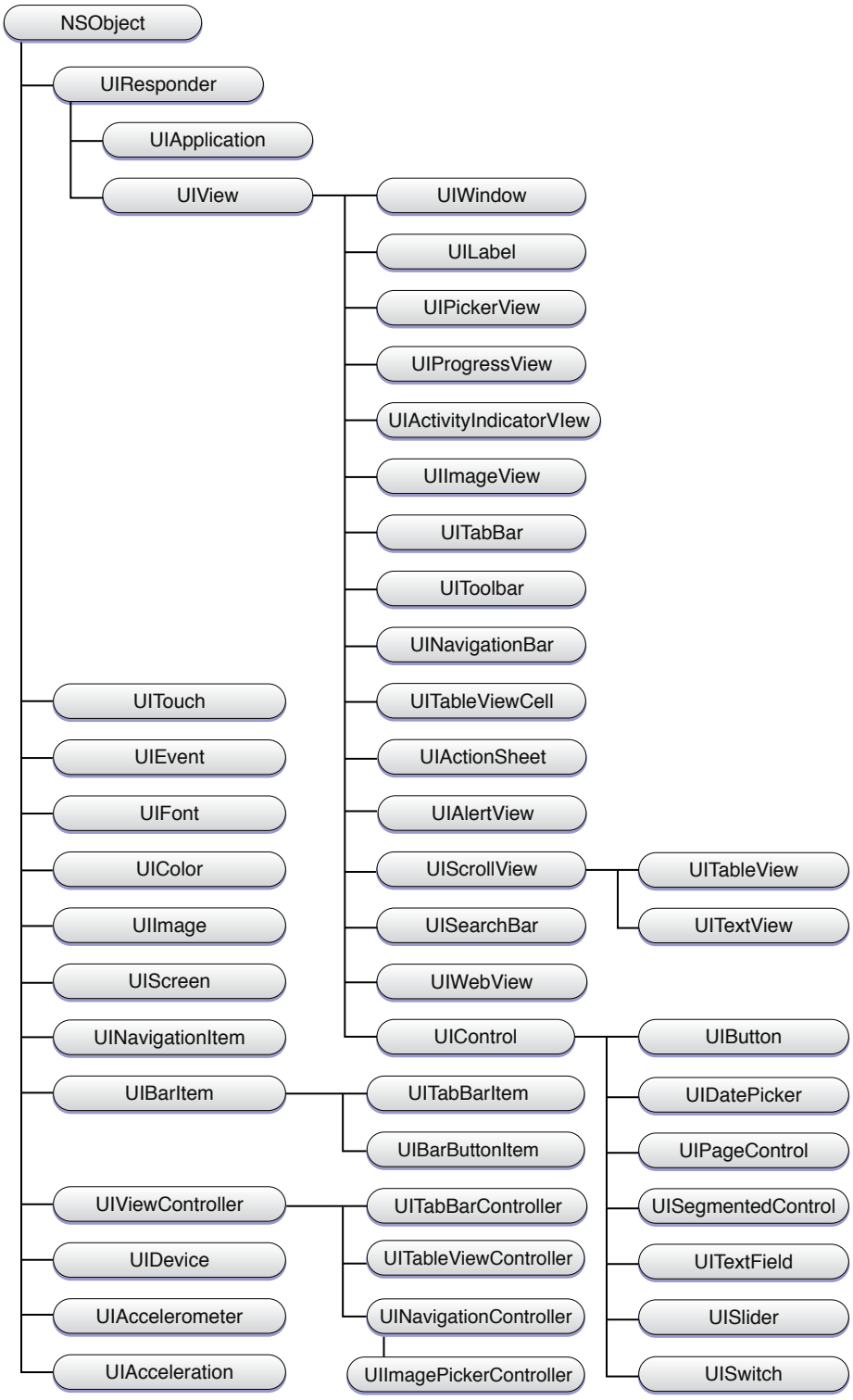
Introduction

Framework	/System/Library/Frameworks/UIKit.framework
Header file directories	/System/Library/Frameworks/UIKit.framework/Headers
Declared in:	UIAccelerometer.h UIActivityIndicatorView.h UIAlert.h UIApplication.h UIBarButtonItem.h UIBarItem.h UIButton.h UIColor.h UIControl.h UIDatePicker.h UIDevice.h UIEvent.h UIFont.h UIGeometry.h UIGraphics.h UIImage.h UIImagePickerController.h UIImageView.h UIInterface.h UILabel.h UINavigationBar.h UINavigationController.h UINibLoading.h UIPageControl.h UIPickerView.h UIProgressView.h UIResponder.h UIScreen.h UIScrollView.h UISearchBar.h UISegmentedControl.h UISlider.h UIStringDrawing.h UISwitch.h UITabBar.h UITabBarController.h UITabBarItem.h

UITableView.h
UITableViewCell.h
UITableViewController.h
UITextField.h
UITextInputTraits.h
UITextView.h
UIToolbar.h
UITouch.h
UIView.h
UIViewController.h
UIWebView.h
UIWindow.h

The UIKit framework provides the classes needed to construct and manage an application's user interface for iPhone and iPod touch. It provides an application object, event handling, drawing model, windows, views, and controls specifically designed for a touch screen interface. Figure I-1 (page 19) illustrates the classes in this framework.

Figure I-1 UIKit class hierarchy



Classes

NSBundle UIKit Additions Reference

Inherits from:	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	NSNibLoading.h

Overview

This category adds methods to the Foundation framework's `NSBundle` class. The method in this category provides support for loading nib files into your application.

Tasks

Loading Nib Files

`- loadNibNamed:owner:options:` (page 23)
Unarchives the contents of a nib file located in the receiver's bundle.

Instance Methods

loadNibNamed:owner:options:

Unarchives the contents of a nib file located in the receiver's bundle.

```
- (NSArray *)loadNibNamed:(NSString *)name owner:(id)owner options:(NSDictionary
    *)options
```

Parameters

name

The name of the nib file, which need not include the `.nib` extension.

owner

The object to assign as the nib's File's Owner object.

options

A dictionary containing the options to use when opening the nib file. For a list of available keys for this dictionary, see "Nib File Loading Options" (page 24).

Return Value

An array containing the top-level objects in the nib file. The array does not contain references to the File's Owner or any proxy objects; it contains only those objects that were instantiated when the nib file was unarchived. You should retain either the returned array or the objects it contains manually to prevent the nib file objects from being released prematurely.

Discussion

You can use this method to load user interfaces and make the objects available to your code. During the loading process, this method unarchives each object, initializes it, sets its properties to their configured values, and reestablishes any connections to other objects. (To establish outlet connections, this method uses the `setValue:forKey:` method, which may cause the object in the outlet to be retained automatically.) For detailed information about the nib-loading process, see *Resource Programming Guide*.

If the nib file contains any proxy objects beyond just the File's Owner proxy object, you can specify the runtime replacement objects for those proxies using the options dictionary. In that dictionary, add the `UINibProxiedObjectsKey` key and set its value to a dictionary containing the names of any proxy objects (the keys) and the real objects to use in their place. The proxy object's name is the string you assign to it in the Name field of the Interface Builder inspector window.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UINibLoading.h`

Constants

Nib File Loading Options

The options that can be specified during nib loading.

```
extern NSString * const UINibProxiedObjectsKey;
```

Constants

`UINibProxiedObjectsKey`

The value for this key is a dictionary that contains the runtime replacement objects for any proxy objects used in the nib file. In this dictionary, the keys are the names associated with the proxy objects and the values are the actual objects from your code that should be used in their place.

Available in iPhone OS 2.0 and later.

Declared in `UINibLoading.h`

NSCoder UIKit Additions Reference

Inherits from:	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIGeometry.h

Overview

This category adds methods to the Foundation framework's `NSCoder` class. The methods in this category let you encode and decode geometry-based data used by the UIKit framework.

Tasks

Encoding Data

- `encodeCGPoint:forKey:` (page 31)
  Encodes a point and associates it with the specified key in the receiver's archive.
- `encodeCGRect:forKey:` (page 31)
  Encodes a rectangle and associates it with the specified key in the receiver's archive.
- `encodeCGSize:forKey:` (page 32)
  Encodes size information and associates it with the specified key in the receiver's archive.
- `encodeCGAffineTransform:forKey:` (page 30)
  Encodes an affine transform and associates it with the specified key in the receiver's archive.
- `encodeUIEdgeInsets:forKey:` (page 32)
  Encodes edge inset data and associates it with the specified key in the receiver's archive.

Decoding Data

- `- decodeCGPointForKey:` (page 29)
  Decodes and returns the `CGPoint` structure associated with the specified key in the receiver's archive.
- `- decodeCGRectForKey:` (page 29)
  Decodes and returns the `CGRect` structure associated with the specified key in the receiver's archive.
- `- decodeCGSizeForKey:` (page 29)
  Decodes and returns the `CGSize` structure associated with the specified key in the receiver's archive.
- `- decodeCGAffineTransformForKey:` (page 28)
  Decodes and returns the `CGAffineTransform` structure associated with the specified key in the receiver's archive.
- `- decodeUIEdgeInsetsForKey:` (page 30)
  Decodes and returns the `UIEdgeInsets` structure associated with the specified key in the receiver's archive.

Instance Methods

decodeCGAffineTransformForKey:

Decodes and returns the `CGAffineTransform` structure associated with the specified key in the receiver's archive.

```
- (CGAffineTransform)decodeCGAffineTransformForKey:(NSString *)key
```

Parameters

key
The key that identifies the affine transform.

Return Value

The affine transform.

Discussion

Use this method to decode size information that was previously encoded using the `encodeCGAffineTransform:forKey:` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- encodeCGAffineTransform:forKey:` (page 30)

Declared In

`UIGeometry.h`

decodeCGPointForKey:

Decodes and returns the `CGPoint` structure associated with the specified key in the receiver's archive.

```
- (CGPoint)decodeCGPointForKey:(NSString *)key
```

Parameters

key

The key that identifies the point.

Return Value

The `CGPoint` structure.

Discussion

Use this method to decode a point that was previously encoded using the `encodeCGPoint:forKey:` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeCGPoint:forKey:` (page 31)

Declared In

`UIGeometry.h`

decodeCGRectForKey:

Decodes and returns the `CGRect` structure associated with the specified key in the receiver's archive.

```
- (CGRect)decodeCGRectForKey:(NSString *)key
```

Parameters

key

The key that identifies the rectangle.

Return Value

The `CGRect` structure.

Discussion

Use this method to decode a rectangle that was previously encoded using the `encodeCGRect:forKey:` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeCGRect:forKey:` (page 31)

Declared In

`UIGeometry.h`

decodeCGSizeForKey:

Decodes and returns the `CGSize` structure associated with the specified key in the receiver's archive.

```
- (CGSize)decodeCGSizeForKey:(NSString *)key
```

Parameters

key

The key that identifies the size information.

Return Value

The `CGSize` structure.

Discussion

Use this method to decode size information that was previously encoded using the `encodeCGSize:forKey:` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeCGSize:forKey:` (page 32)

Declared In

`UIGeometry.h`

decodeUIEdgeInsetsForKey:

Decodes and returns the `UIEdgeInsets` structure associated with the specified key in the receiver's archive.

```
- (UIEdgeInsets)decodeUIEdgeInsetsForKey:(NSString *)key
```

Parameters

key

The key that identifies the edge insets.

Return Value

The edge insets data.

Discussion

Use this method to decode size information that was previously encoded using the `encodeUIEdgeInsets:forKey:` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeUIEdgeInsets:forKey:` (page 32)

Declared In

`UIGeometry.h`

encodeCGAffineTransform:forKey:

Encodes an affine transform and associates it with the specified key in the receiver's archive.

```
- (void)encodeCGAffineTransform:(CGAffineTransform)transform forKey:(NSString *)key
```

Parameters

transform

The transform information to encode.

key

The key identifying the data.

Discussion

When decoding the data from the archive, you pass the value in the *key* parameter to the corresponding `decodeCGAffineTransformForKey:` method to retrieve the data.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- decodeCGAffineTransformForKey:` (page 28)

Declared In

`UIGeometry.h`

encodeCGPoint:forKey:

Encodes a point and associates it with the specified key in the receiver's archive.

```
- (void)encodeCGPoint:(CGPoint)point forKey:(NSString *)key
```

Parameters

point

The point to encode.

key

The key identifying the data.

Discussion

When decoding the data from the archive, you pass the value in the *key* parameter to the corresponding `decodeCGPointForKey:` method to retrieve the data.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- decodeCGPointForKey:` (page 29)

Declared In

`UIGeometry.h`

encodeCGRect:forKey:

Encodes a rectangle and associates it with the specified key in the receiver's archive.

```
- (void)encodeCGRect:(CGRect)rect forKey:(NSString *)key
```

Parameters

rect

The rectangle to encode.

key
The key identifying the data.

Discussion
When decoding the data from the archive, you pass the value in the *key* parameter to the corresponding `decodeCGRectForKey:` method to retrieve the data.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- decodeCGRectForKey:` (page 29)

Declared In
`UIGeometry.h`

encodeCGSize:forKey:

Encodes size information and associates it with the specified key in the receiver's archive.

```
- (void)encodeCGSize:(CGSize)size forKey:(NSString *)key
```

Parameters

size
The size information to encode.

key
The key identifying the data.

Discussion
When decoding the data from the archive, you pass the value in the *key* parameter to the corresponding `decodeCGSizeForKey:` method to retrieve the data.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- decodeCGSizeForKey:` (page 29)

Declared In
`UIGeometry.h`

encodeUIEdgeInsets:forKey:

Encodes edge inset data and associates it with the specified key in the receiver's archive.

```
- (void)encodeUIEdgeInsets:(UIEdgeInsets)insets forKey:(NSString *)key
```

Parameters

insets
The edge insets data to encode.

key
The key identifying the data.

Discussion

When decoding the data from the archive, you pass the value in the *key* parameter to the corresponding `decodeUIEdgeInsetsForKey:` method to retrieve the data.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- decodeUIEdgeInsetsForKey:` (page 30)

Declared In

`UIGeometry.h`

NSIndexPath UIKit Additions

Inherits from:	NSObject
Conforms to:	NSObject (NSObject) NSCoding NSCopying
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITableView.h

Overview

The UIKit framework adds programming interfaces to the `NSIndexPath` class of the Foundation framework to facilitate the identification of rows and sections in `UITableView` objects.

The API consists of a class factory method and two properties. The `indexPathForRow:inSection:` (page 36) method creates an `NSIndexPath` object from row and section index numbers. The properties return the row index number and the section index number from such objects.

Tasks

Creating an Index Path Object

`+ indexPathForRow:inSection:` (page 36)
Returns an index-path object initialized with the indexes of a specific row and section in a table view.

Getting the Row and Section Indexes

`row` (page 36) *property*
An index number identifying a row in a section of a table view. (read-only)

`section` (page 36) *property*
An index number identifying a section in a table view. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

row

An index number identifying a row in a section of a table view. (read-only)

```
@property(readonly) NSUInteger row
```

Discussion
The section the row is in is identified by the value of `section` (page 36).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

section

An index number identifying a section in a table view. (read-only)

```
@property(readonly) NSUInteger section
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

Class Methods

indexPathForRow:inSection:

Returns an index-path object initialized with the indexes of a specific row and section in a table view.

```
+ (NSIndexPath *)indexPathForRow:(NSUInteger)row inSection:(NSUInteger)section
```

Parameters

row

An index number identifying a row in a `UITableView` object in a section identified by *section*.

section

An index number identifying a section in a `UITableView` object.

Return Value

An `NSIndexPath` object or `nil` if the object could not be created. The returned object is autoreleased.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

NSObject UIKit Additions Reference

Inherits from:	none
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	NSNibLoading.h

Overview

This category adds methods to the Foundation framework's `NSObject` class. The method in this category provides support for loading nib files into your application.

Tasks

Responding to Being Loaded from a Nib File

- `awakeFromNib` (page 39)
  Prepares the receiver for service after it has been loaded from an Interface Builder archive, or nib file.

Instance Methods

awakeFromNib

Prepares the receiver for service after it has been loaded from an Interface Builder archive, or nib file.

```
- (void)awakeFromNib
```

Discussion

The nib-loading infrastructure sends an `awakeFromNib` message to each object recreated from a nib archive, but only after all the objects in the archive have been loaded and initialized. When an object receives an `awakeFromNib` message, it is guaranteed to have all its outlet and action connections already established.

You must call the `super` implementation of `awakeFromNib` to give parent classes the opportunity to perform any additional initialization they require. Although the default implementation of this method does nothing, many UIKit classes provide non-empty implementations. You may call the `super` implementation at any point during your own `awakeFromNib` method.

Note: During Interface Builder's test mode, this message is also sent to objects instantiated from loaded Interface Builder plug-ins. Because plug-ins link against the framework containing the object definition code, Interface Builder is able to call their `awakeFromNib` method when present. The same is not true for custom objects that you create for your Xcode projects. Interface Builder knows only about the defined outlets and actions of those objects; it does not have access to the actual code for them.

During the instantiation process, each object in the archive is unarchived and then initialized with the method befitting its type. Objects that conform to the `NSCoding` protocol (including all subclasses of `UIView` and `UIViewController`) are initialized using their `initWithCoder:` method. All objects that do not conform to the `NSCoding` protocol are initialized using their `init` method. After all objects have been instantiated and initialized, the nib-loading code reestablishes the outlet and action connections for all of those objects. It then calls the `awakeFromNib` method of the objects. For more detailed information about the steps followed during the nib-loading process, see Nib Files and Cocoa in *Resource Programming Guide*.

Important: Because the order in which objects are instantiated from an archive is not guaranteed, your initialization methods should not send messages to other objects in the hierarchy. Messages to other objects can be sent safely from within an `awakeFromNib` method.

Typically, you implement `awakeFromNib` for objects that require additional set up that cannot be done at design time. For example, you might use this method to customize the default configuration of any controls to match user preferences or the values in other controls. You might also use it to restore individual controls to some previous state of your application.

Availability

Available in iPhone OS 2.0 and later.

See Also

`awakeAfterUsingCoder:` (NSObject class)

`initWithCoder:` (NSCoding protocol)

`initialize` (NSObject class)

Declared In

`UINibLoading.h`

NSString UIKit Additions Reference

Inherits from:	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIStringDrawing.h

Overview

The UIKit framework adds methods to `NSString` to support the drawing of strings and to compute the bounding box of a string prior to drawing.

By default, strings are drawn using the native coordinate system of iPhone OS, where content is drawn down and to the right from the specified origin point. Whenever you are positioning string content, you should keep this orientation in mind and use the upper-left corner of the string's bounding box as the origin point for drawing.

Tasks

Getting the Drawing Rect of a String

`- sizeWithFont:` (page 47)
Returns the size of the string if it were to be rendered with the specified font.

`- sizeWithFont:forWidth:lineBreakMode:` (page 48)
Returns the size of the string if it were to be rendered with the specified font and line attributes.

`- sizeWithFont:constrainedToSize:` (page 47)
Returns the size of the string if it were rendered and constrained to the specified size.

`- sizeWithFont:constrainedToSize:lineBreakMode:` (page 48)
Returns the size of the string if it were rendered with the specified constraints.

- sizeWithFont:minFontSize:actualFontSize:forWidth:lineBreakMode: (page 49)
  Returns the size of the string if it were rendered with the specified constraints, including a variable font size.

Drawing String Objects

- drawAtPoint:withFont: (page 45)
  Draws a single line of text at the specified point in the current context using the specified font.
- drawAtPoint:forWidth:withFont:lineBreakMode: (page 43)
  Draws a single line of text at the specified point in the current context using the specified font and attributes.
- drawAtPoint:forWidth:withFont:fontSize:lineBreakMode:baselineAdjustment: (page 42)
  Draws the string at the specified point in the current context using the specified font and attributes.
- drawAtPoint:forWidth:withFont:minFontSize:actualFontSize:lineBreakMode:baselineAdjustment: (page 44)
  Draws the string with the specified font and attributes, adjusting the font attributes as needed to render as much of the text as possible.
- drawInRect:withFont: (page 45)
  Draws the string in the current context using the specified bounding rectangle and font.
- drawInRect:withFont:lineBreakMode: (page 46)
  Draws the string in the current context using the specified bounding rectangle, font, and attributes.
- drawInRect:withFont:lineBreakMode:alignment: (page 46)
  Draws the string in the current context using the specified bounding rectangle, font and attributes.

Instance Methods

drawAtPoint:forWidth:withFont:fontSize:lineBreakMode:baselineAdjustment:

Draws the string at the specified point in the current context using the specified font and attributes.

```
- (CGSize)drawAtPoint:(CGPoint)point forWidth:(CGFloat)width withFont:(UIFont *)font
    fontSize:(CGFloat)fontSize lineBreakMode:(UILineBreakMode)lineBreakMode
  baselineAdjustment:(UIBaselineAdjustment)baselineAdjustment
```

Parameters

point
  The location (in the coordinate system of the current graphics context) at which to draw the string. This point represents the top-left corner of the string's bounding box.

width
  The maximum width of the string.

font
  The font to use for rendering.

fontSize
The font size to use instead of the one associated with the font object in the *font* parameter.

lineBreakMode
The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

baselineAdjustment
Specifies the vertical text-adjustment rule to use. This rule is used to determine the position of the text in cases where the text must be drawn at a smaller size.

Return Value
The actual size of the rendered string.

Discussion
This method draws only a single line of text, drawing as much of the string as possible using the given font and constraints. This method does not perform any line wrapping during drawing.

Availability
Available in iPhone OS 2.0 and later.

See Also
- sizeWithFont:minFontSize:actualFontSize:forWidth:lineBreakMode: (page 49)

Declared In
UIStringDrawing.h

drawAtPoint:forWidth:withFont:lineBreakMode:

Draws a single line of text at the specified point in the current context using the specified font and attributes.

```
- (CGSize)drawAtPoint:(CGPoint)point forWidth:(CGFloat)width withFont:(UIFont *)font
    lineBreakMode:(UILineBreakMode)lineBreakMode
```

Parameters

point
The location (in the coordinate system of the current graphics context) at which to draw the string. This point represents the top-left corner of the string's bounding box.

width
The maximum width of the string.

font
The font to use for rendering.

lineBreakMode
The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

Return Value
The actual size of the rendered string.

Discussion
This method draws only a single line of text, drawing as much of the string as possible using the given font and constraints. This method does not perform any line wrapping during drawing.

If the value in the *width* parameter is smaller than actual width of the string, truncation may occur. In that situation, the options in the *lineBreakMode* parameter determine where to end the text.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIStringDrawing.h

drawAtPoint:forWidth:withFont:minFontSize:actualFontSize:lineBreakMode:baselineAdjustment:

Draws the string with the specified font and attributes, adjusting the font attributes as needed to render as much of the text as possible.

```
- (CGSize)drawAtPoint:(CGPoint)point forWidth:(CGFloat)width withFont:(UIFont *)font
    minFontSize:(CGFloat)minFontSize actualFontSize:(CGFloat *)actualFontSize
   lineBreakMode:(UILineBreakMode)lineBreakMode
   baselineAdjustment:(UIBaselineAdjustment)baselineAdjustment
```

Parameters

point
The location (in the coordinate system of the current graphics context) at which to draw the string. This point represents the top-left corner of the string's bounding box.

width
The maximum width of the string.

font
The font to use for rendering.

minFontSize
The minimum size to which the font may be reduced before resorting to truncation of the text.

actualFontSize
On input, a pointer to a floating-point value. On return, this value contains the actual font size that was used to render the string.

lineBreakMode
The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

baselineAdjustment
Specifies the vertical text-adjustment rule to use. This rule is used to determine the position of the text in cases where the text must be drawn at a smaller size.

Return Value
The actual size of the rendered string.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIStringDrawing.h

drawAtPoint:withFont:

Draws a single line of text at the specified point in the current context using the specified font.

```
- (CGSize)drawAtPoint:(CGPoint)point withFont:(UIFont *)font
```

Parameters

point

The location (in the coordinate system of the current graphics context) at which to draw the string. This point represents the top-left corner of the string's bounding box.

font

The font to use for rendering.

Return Value

The actual size of the rendered string.

Discussion

This method draws only a single line of text, drawing as much of the string as possible using the given font. This method does not perform any line wrapping during drawing.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIStringDrawing.h`

drawInRect:withFont:

Draws the string in the current context using the specified bounding rectangle and font.

```
- (CGSize)drawInRect:(CGRect)rect withFont:(UIFont *)font
```

Parameters

rect

The bounding rectangle (in the current context) in which to draw the string.

font

The font to use for rendering.

Return Value

The actual size of the rendered string.

Discussion

This method draws only a single line of text, drawing as much of the string as possible using the given font and constraints. This method uses word-based line wrapping and the text is drawn left-aligned.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIStringDrawing.h`

drawInRect:withFont:lineBreakMode:

Draws the string in the current context using the specified bounding rectangle, font, and attributes.

```
- (CGSize)drawInRect:(CGRect)rect withFont:(UIFont *)font
    lineBreakMode:(UILineBreakMode)lineBreakMode
```

Parameters

rect

The bounding rectangle (in the current context) in which to draw the string.

font

The font to use for rendering.

lineBreakMode

The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

Return Value

The actual size of the rendered string.

Discussion

This method draws only a single line of text, drawing as much of the string as possible using the given font, line break mode, and size constraints. The text is drawn left-aligned.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIStringDrawing.h

drawInRect:withFont:lineBreakMode:alignment:

Draws the string in the current context using the specified bounding rectangle, font and attributes.

```
- (CGSize)drawInRect:(CGRect)rect withFont:(UIFont *)font
    lineBreakMode:(UILineBreakMode)lineBreakMode alignment:(UITextAlignment)alignment
```

Parameters

rect

The bounding rectangle (in the current context) in which to draw the string.

font

The font to use for rendering.

lineBreakMode

The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

alignment

The alignment of the text inside the bounding rectangle. For a list of possible values, see "UITextAlignment" (page 51).

Return Value

The actual size of the rendered string.

Availability

Available in iPhone OS 2.0 and later.

Declared In
UIStringDrawing.h

sizeWithFont:

Returns the size of the string if it were to be rendered with the specified font.

```
- (CGSize)sizeWithFont:(UIFont *)font
```

Parameters

font

The font to use for computing the string size.

Return Value
The width and height of the resulting string's bounding box.

Discussion
This method does not perform any line wrapping and returns the absolute width of the string using the specified font.

This method does not actually draw the string. You can use it to obtain the layout metrics you need to draw the receiver in your user interface.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIStringDrawing.h

sizeWithFont:constrainedToSize:

Returns the size of the string if it were rendered and constrained to the specified size.

```
- (CGSize)sizeWithFont:(UIFont *)font constrainedToSize:(CGSize)size
```

Parameters

font

The font to use for computing the string size.

size

The maximum acceptable size for the string. This value is used to calculate where line breaks and wrapping would occur.

Return Value
The width and height of the resulting string's bounding box.

Discussion
This method does not actually draw the string. You can use it to obtain the layout metrics you need to draw the receiver in your user interface. Although this method uses the *size* parameter to compute the size based on where line breaks would occur, it does not actually wrap the text to multiple lines.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIStringDrawing.h

sizeWithFont:constrainedToSize:lineBreakMode:

Returns the size of the string if it were rendered with the specified constraints.

```
- (CGSize)sizeWithFont:(UIFont *)font constrainedToSize:(CGSize)size
    lineBreakMode:(UILineBreakMode)lineBreakMode
```

Parameters

font

The font to use for computing the string size.

size

The maximum acceptable size for the string. This value is used to calculate where line breaks and wrapping would occur.

lineBreakMode

The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

Return Value
The width and height of the resulting string's bounding box.

Discussion
This method wraps the receiver's text and truncates it, as needed, to fit the specified size. It then returns the actual size of the resulting text. If the height specified in the *size* parameter is less than a single line of text, this method may return a height value that is bigger than the one specified.

This method does not actually draw the string. You can use it to obtain the layout metrics you need to draw the receiver in your user interface.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIStringDrawing.h

sizeWithFont:forWidth:lineBreakMode:

Returns the size of the string if it were to be rendered with the specified font and line attributes.

```
- (CGSize)sizeWithFont:(UIFont *)font forWidth:(CGFloat)width
    lineBreakMode:(UILineBreakMode)lineBreakMode
```

Parameters

font

The font to use for computing the string size.

width

The maximum acceptable width for the string. This value is used to calculate where line breaks would be placed.

lineBreakMode

The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

Return Value

The width and height of the resulting string's bounding box.

Discussion

This method returns the width and height of the string constrained to the specified width. Although it computes where line breaks would occur, this method does not actually wrap the text to additional lines; therefore, drawing the string with the returned information may result in truncation.

This method does not actually draw the string. You can use it to obtain the layout metrics you need to draw the receiver in your user interface.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIStringDrawing.h

sizeWithFont:minFontSize:actualFontSize:forWidth:lineBreakMode:

Returns the size of the string if it were rendered with the specified constraints, including a variable font size.

```
- (CGSize)sizeWithFont:(UIFont *)font minFontSize:(CGFloat)minFontSize
    actualFontSize:(CGFloat *)actualFontSize forWidth:(CGFloat)width
    lineBreakMode:(UILineBreakMode)lineBreakMode
```

Parameters

font

The font to use for computing the string size.

minFontSize

The minimum size to which the font may be reduced before resorting to truncation of the text.

actualFontSize

On input, a pointer to a floating-point value. On return, this value contains the actual font size that was used to compute the size of the string.

width

The maximum acceptable width for the string. This value is used to calculate where line breaks would be placed.

lineBreakMode

The line break options for computing the size of the string. For a list of possible values, see "UILineBreakMode" (page 50).

Return Value

The width and height of the resulting string's bounding box.

Discussion

Although it computes where line breaks would occur, this method does not actually wrap the text to additional lines. If the entire string does not fit within the given width using the initial font size, this method reduces the font size until the string does fit or it reaches the specified minimum font size. If it reaches the minimum size, this method begins truncating the text to fit.

This method does not actually draw the string. You can use it to obtain the layout metrics you need to draw the receiver in your user interface.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIStringDrawing.h`

Constants

UILineBreakMode

Options for wrapping and truncating text.

```
typedef enum {
    UILineBreakModeWordWrap = 0,
    UILineBreakModeCharacterWrap,
    UILineBreakModeClip,
    UILineBreakModeHeadTruncation,
    UILineBreakModeTailTruncation,
    UILineBreakModeMiddleTruncation,
} UILineBreakMode;
```

Constants

`UILineBreakModeWordWrap`
Wrap or clip the string only at word boundaries. This is the default wrapping option.
Available in iPhone OS 2.0 and later.
Declared in `UIStringDrawing.h`

`UILineBreakModeCharacterWrap`
Wrap or clip the string at the closest character boundary.
Available in iPhone OS 2.0 and later.
Declared in `UIStringDrawing.h`

`UILineBreakModeClip`
Clip the text when the end of the drawing rectangle is reached. This option could result in a partially rendered character at the end of a string.
Available in iPhone OS 2.0 and later.
Declared in `UIStringDrawing.h`

`UILineBreakModeHeadTruncation`
Truncate text (as needed) from the beginning of the line. For multiple lines of text, only text on the first line is truncated.
Available in iPhone OS 2.0 and later.
Declared in `UIStringDrawing.h`

`UILineBreakModeTailTruncation`
: Truncate text (as needed) from the end of the line. For multiple lines of text, only text on the last line is truncated.

  Available in iPhone OS 2.0 and later.

  Declared in `UIStringDrawing.h`

`UILineBreakModeMiddleTruncation`
: Truncate text (as needed) from the middle of the line. For multiple lines of text, text is truncated only at the midpoint of the line.

  Available in iPhone OS 2.0 and later.

  Declared in `UIStringDrawing.h`

Discussion

For methods that draw at a specified point (as opposed to those that draw in a rectangular region), these options specify the clipping behavior that is applied to the string.

Declared In

`UIStringDrawing.h`

UITextAlignment

Options for aligning text horizontally.

```
typedef enum {
    UITextAlignmentLeft,
    UITextAlignmentCenter,
    UITextAlignmentRight,
} UITextAlignment;
```

Constants

`UITextAlignmentLeft`
: Align text along the left edge.

  Available in iPhone OS 2.0 and later.

  Declared in `UIStringDrawing.h`

`UITextAlignmentCenter`
: Align text equally along both sides of the center line.

  Available in iPhone OS 2.0 and later.

  Declared in `UIStringDrawing.h`

`UITextAlignmentRight`
: Align text along the right edge.

  Available in iPhone OS 2.0 and later.

  Declared in `UIStringDrawing.h`

Declared In

`UIStringDrawing.h`

UIBaselineAdjustment

Vertical adjustment options.

```
typedef enum {
    UIBaselineAdjustmentAlignBaselines,
    UIBaselineAdjustmentAlignCenters,
    UIBaselineAdjustmentNone,
} UIBaselineAdjustment
```

Constants

`UIBaselineAdjustmentAlignBaselines`

Adjust text relative to the position of its baseline.

Available in iPhone OS 2.0 and later.

Declared in `UIStringDrawing.h`

`UIBaselineAdjustmentAlignCenters`

Adjust text based relative to the center of its bounding box.

Available in iPhone OS 2.0 and later.

Declared in `UIStringDrawing.h`

`UIBaselineAdjustmentNone`

Adjust text relative to the top-left corner of the bounding box. This is the default adjustment.

Available in iPhone OS 2.0 and later.

Declared in `UIStringDrawing.h`

Discussion

Baseline adjustment options determine how to adjust the position of text in cases where the text must be drawn using a different font size than the one originally specified. For example, with the `UIBaselineAdjustmentAlignBaselines` option, the position of the baseline remains fixed at its initial location while the text appears to move toward that baseline. Similarly, the `UIBaselineAdjustmentNone` option makes it appear as if the text is moving upwards towards the top-left corner of the bounding box.

Declared In

`UIStringDrawing.h`

NSValue UIKit Additions Reference

Inherits from:	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIGeometry.h

Overview

This category adds methods to the Foundation framework's `NSValue` class. The methods in this category let you represent geometry-based data using an `NSValue` object.

Tasks

Creating an NSValue

`+ valueWithCGPoint:` (page 54)
Creates and returns a value object that contains the specified point structure.

`+ valueWithCGRect:` (page 55)
Creates and returns a value object that contains the specified rectangle structure.

`+ valueWithCGSize:` (page 55)
Creates and returns a value object that contains the specified size structure.

`+ valueWithCGAffineTransform:` (page 54)
Creates and returns a value object that contains the specified affine transform data.

`+ valueWithUIEdgeInsets:` (page 56)
Creates and returns a value object that contains the specified edge inset data.

Accessing Data

- `CGPointValue` (page 56)
  Returns a point structure representing the data in the receiver.
- `CGRectValue` (page 57)
  Returns a rectangle structure representing the data in the receiver.
- `CGSizeValue` (page 57)
  Returns a size structure representing the data in the receiver.
- `CGAffineTransformValue` (page 56)
  Returns an affine transform structure representing the data in the receiver.
- `UIEdgeInsetsValue` (page 57)
  Returns an edge insets structure representing the data in the receiver.

Class Methods

valueWithCGAffineTransform:

Creates and returns a value object that contains the specified affine transform data.

```
+ (NSValue *)valueWithCGAffineTransform:(CGAffineTransform)transform
```

Parameters

transform
  The value for the new object.

Return Value

A new value object that contains the affine transform data.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `CGAffineTransformValue` (page 56)

Declared In

`UIGeometry.h`

valueWithCGPoint:

Creates and returns a value object that contains the specified point structure.

```
+ (NSValue *)valueWithCGPoint:(CGPoint)point
```

Parameters

point
  The value for the new object.

Return Value

A new value object that contains the point information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- CGPointValue (page 56)

Declared In
UIGeometry.h

valueWithCGRect:

Creates and returns a value object that contains the specified rectangle structure.

```
+ (NSValue *)valueWithCGRect:(CGRect)rect
```

Parameters

rect
The value for the new object.

Return Value
A new value object that contains the rectangle information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- CGRectValue (page 57)

Declared In
UIGeometry.h

valueWithCGSize:

Creates and returns a value object that contains the specified size structure.

```
+ (NSValue *)valueWithCGSize:(CGSize)size
```

Parameters

size
The value for the new object.

Return Value
A new value object that contains the size information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- CGSizeValue (page 57)

Declared In
UIGeometry.h

valueWithUIEdgeInsets:

Creates and returns a value object that contains the specified edge inset data.

```
+ (NSValue *)valueWithUIEdgeInsets:(UIEdgeInsets)insets
```

Parameters

insets

The value for the new object.

Return Value

A new value object that contains the edge inset data.

Availability

Available in iPhone OS 2.0 and later.

See Also

- UIEdgeInsetsValue (page 57)

Declared In

UIGeometry.h

Instance Methods

CGAffineTransformValue

Returns an affine transform structure representing the data in the receiver.

```
- (CGAffineTransform)CGAffineTransformValue
```

Return Value

An affine transform structure containing the receiver's value.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ valueWithCGAffineTransform: (page 54)

Declared In

UIGeometry.h

CGPointValue

Returns a point structure representing the data in the receiver.

```
- (CGPoint)CGPointValue
```

Return Value

A point structure containing the receiver's value.

Availability

Available in iPhone OS 2.0 and later.

See Also
+ valueWithCGPoint: (page 54)

Declared In
UIGeometry.h

CGRectValue

Returns a rectangle structure representing the data in the receiver.

```
- (CGRect)CGRectValue
```

Return Value
A rectangle structure containing the receiver's value.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ valueWithCGRect: (page 55)

Declared In
UIGeometry.h

CGSizeValue

Returns a size structure representing the data in the receiver.

```
- (CGSize)CGSizeValue
```

Return Value
A size structure containing the receiver's value.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ valueWithCGSize: (page 55)

Declared In
UIGeometry.h

UIEdgeInsetsValue

Returns an edge insets structure representing the data in the receiver.

```
- (UIEdgeInsets)UIEdgeInsetsValue
```

Return Value
An edge insets structure containing the receiver's value.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ valueWithUIEdgeInsets:` (page 56)

Declared In
`UIGeometry.h`

UIAcceleration Class Reference

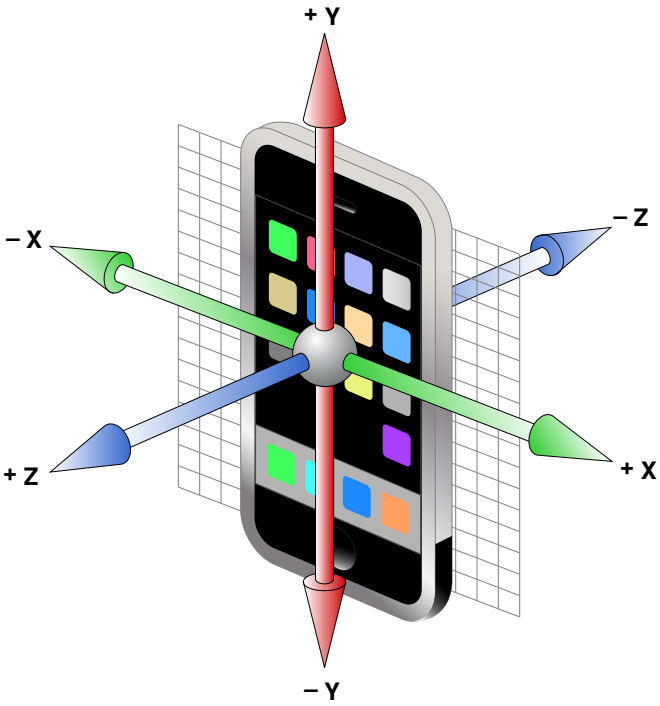
Inherits from:	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAccelerometer.h

Overview

The `UIAcceleration` class stores the data associated with an acceleration event. When your application receives an accelerometer notification, an instance of this class is stored in the `object` field of the notification. For information on how to receive accelerometer notifications, see `UIAccelerometer` class.

Each acceleration event relays the current acceleration readings along the three axes of the device (shown in Figure 7-1). Acceleration values for each axis are reported directly by the hardware as G-force values. Therefore, a value of `1.0` represents a load of about +1g along a given axis while a value of `-1.0` represents -1g.

Figure 7-1 Orientation of the device axes



Note: The values reported by the accelerometers are approximate and should not be used to make precise measurements. It is recommended that you average accelerometer data over time to extract the values you need for your usage.

Tasks

Accessing the Acceleration Data

`x` (page 61) *property*
The acceleration value for the x axis of the device. (read-only)

`y` (page 61) *property*
The acceleration value for the y axis of the device. (read-only)

`z` (page 62) *property*
The acceleration value for the z axis of the device. (read-only)

`timestamp` (page 61) *property*
The relative time at which the information was recorded. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

timestamp

The relative time at which the information was recorded. (read-only)

```
@property(nonatomic, readonly) NSTimeInterval timestamp
```

Discussion

This value records the time relative to the CPU time base register. You should not use it to determine the exact time at which the event occurred but should instead compare it to the timestamp of another acceleration event to determine the elapsed time between the events.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAccelerometer.h`

x

The acceleration value for the x axis of the device. (read-only)

```
@property(nonatomic, readonly) UIAccelerationValue x
```

Discussion

With the device held in portrait orientation and the screen facing you, the x axis runs from left to right across the face of the device.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAccelerometer.h`

y

The acceleration value for the y axis of the device. (read-only)

```
@property(nonatomic, readonly) UIAccelerationValue y
```

Discussion

With the device held in portrait orientation and the screen facing you, the y axis runs from top to bottom across the face of the device.

Availability

Available in iPhone OS 2.0 and later.

Declared In
UIAccelerometer.h

z

The acceleration value for the z axis of the device. (read-only)

```
@property(nonatomic, readonly) UIAccelerationValue z
```

Discussion
With the device held in portrait orientation and the screen facing you, the z axis runs from back to front through the device.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAccelerometer.h

Constants

UIAccelerationValue

The amount of acceleration in a single linear direction.

```
typedef double UIAccelerationValue;
```

Discussion
This type is used to store acceleration values, which are specified as G-force values. For example, the value 1.0 corresponds to the normal acceleration caused by gravity.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAccelerometer.h

UIAccelerometer Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAccelerometer.h

Overview

The `UIAccelerometer` class lets you register to receive acceleration-related data from the onboard hardware. As a device moves, its hardware reports linear acceleration changes along the primary axes in three-dimensional space. You can use this data to detect both the current orientation of the device (relative to the ground) and any instantaneous changes to that orientation. You might use instantaneous changes as input to a game or to initiate some action in your application.

You do not create accelerometer objects directly. Instead, you use the shared `UIAccelerometer` object to specify the interval at which you want to receive events and then set its `delegate` property. Upon assigning your delegate object, the accelerometer object begins delivering acceleration events to your delegate immediately at the specified interval. Events are always delivered on the main thread of your application.

The maximum frequency for accelerometer updates is based on the available hardware. You can request updates less frequently but cannot request them more frequently than the hardware maximum. Once you assign your delegate, however, updates are delivered regularly at the frequency you requested, whether or not the acceleration data actually changed. Your delegate is responsible for filtering out any unwanted updates and for ensuring that the amount of change is significant enough to warrant taking action.

For more information about the data delivered to your observer, see *UIAcceleration Class Reference*. For information about implementing your delegate object, see *UIAccelerometerDelegate Protocol Reference*.

Tasks

Getting the Shared Accelerometer Object

`+ sharedAccelerometer` (page 65)
Returns the shared accelerometer object for the system.

Accessing the Accelerometer Properties

`updateInterval` (page 64) *property*
The interval at which to deliver acceleration data to the delegate.

`delegate` (page 64) *property*
The delegate object you want to receive acceleration events.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

delegate

The delegate object you want to receive acceleration events.

```
@property(nonatomic, assign) id<UIAccelerometerDelegate> delegate
```

Discussion

The `UIAccelerometerDelegate` is a formal protocol, so your delegate object must implement the method it defines. The shared accelerometer object delivers the acceleration data to your delegate at the specified interval. It delivers these events on the main thread of your application when it is in the `NSDefaultRunLoopMode` run loop mode.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAccelerometer.h`

updateInterval

The interval at which to deliver acceleration data to the delegate.

```
@property(nonatomic) NSTimeInterval updateInterval
```

Discussion
This property is measured in seconds. The value of this property is capped to certain minimum and maximum values. The maximum value is determined by the maximum frequency supported by the hardware. To ensure that it can deliver device orientation events in a timely fashion, the system determines the appropriate minimum value based on its needs.

Changes to this property are delivered synchronously to the accelerometer hardware. You may change this property while the delegate is non-`nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAccelerometer.h`

Class Methods

sharedAccelerometer

Returns the shared accelerometer object for the system.

```
+ (UIAccelerometer *)sharedAccelerometer
```

Return Value
The systemwide accelerometer object.

Discussion
Always use this method to retrieve the shared system accelerometer object. Do not create new instances of the `UIAccelerometer` class.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAccelerometer.h`

UIActionSheet Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAlert.h

Overview

Use the `UIActionSheet` class to implement an action sheet that displays a message and presents buttons that let the user decide how to proceed. An action sheet is similar in function but differs in appearance from an alert view.

Use the properties and methods in this class to set the message, set the style, set the delegate, configure the buttons, and display the action sheet. You must set a delegate if you add custom buttons. The delegate should conform to the `UIActionSheetDelegate` protocol. When you display an action sheet, you can optionally animate it from the bottom bar or an arbitrary view. How the action sheet is animated depends on the bar style or the action sheet style you set.

Tasks

Creating Action Sheets

`- initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:` (page 72)

Convenience method for initializing an action sheet.

Setting Properties

`delegate` (page 69) *property*
The receiver's delegate or `nil` if it doesn't have a delegate.

`title` (page 70) *property*
The string that appears in the receiver's title bar.

`visible` (page 71) *property*
A Boolean value that indicates whether the receiver is displayed. (read-only)

`actionSheetStyle` (page 69) *property*
The receiver's style.

Configuring Buttons

\- `addButtonWithTitle:` (page 71)
Adds a button to the receiver with the given title.

`numberOfButtons` (page 70) *property*
The number of buttons on the action sheet. (read-only)

\- `buttonTitleAtIndex:` (page 71)
Returns the title of the button at the given index.

`cancelButtonIndex` (page 69) *property*
The index number of the cancel button.

`destructiveButtonIndex` (page 70) *property*
The index number of the destructive button.

`firstOtherButtonIndex` (page 70) *property*
The index of the first other button. (read-only)

Displaying

\- `showFromTabBar:` (page 73)
Displays the receiver using animation by originating it from the specified tab bar.

\- `showFromToolbar:` (page 73)
Displays the receiver using animation by originating it from the specified toolbar.

\- `showInView:` (page 74)
Displays the receiver using animation by originating it from the specified view.

Dismissing

\- `dismissWithClickedButtonIndex:animated:` (page 72)
Dismisses the receiver, optionally with animation.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

actionSheetStyle

The receiver's style.

```
@property(nonatomic) UIActionSheetStyle actionSheetStyle
```

Discussion
See `UIActionSheetStyle` (page 74) for the possible values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAlert.h`

cancelButtonIndex

The index number of the cancel button.

```
@property(nonatomic) NSInteger cancelButtonIndex
```

Discussion
The button indices start at `0`. If `-1`, then the index is not set.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAlert.h`

delegate

The receiver's delegate or `nil` if it doesn't have a delegate.

```
@property(nonatomic, assign) id<UIActionSheetDelegate> delegate
```

Discussion
See *UIActionSheetDelegate Protocol Reference* for the methods this delegate should implement.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAlert.h`

destructiveButtonIndex

The index number of the destructive button.

```
@property(nonatomic) NSInteger destructiveButtonIndex
```

Discussion

The button indices start at `0`. If `-1`, then the index is not set. This property is ignored if there is only one button. The default value is `-1`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAlert.h`

firstOtherButtonIndex

The index of the first other button. (read-only)

```
@property(nonatomic, readonly) NSInteger firstOtherButtonIndex
```

Discussion

The button indices start at `0`. If `-1`, then the index is not set. This property is ignored if there are no other buttons. The default value is `-1`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAlert.h`

numberOfButtons

The number of buttons on the action sheet. (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfButtons
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAlert.h`

title

The string that appears in the receiver's title bar.

```
@property(nonatomic, copy) NSString *title
```

Availability

Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

visible

A Boolean value that indicates whether the receiver is displayed. (read-only)

```
@property(nonatomic, readonly, getter=isVisible) BOOL visible
```

Discussion
If YES, the receiver is displayed; otherwise, NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

Instance Methods

addButtonWithTitle:

Adds a button to the receiver with the given title.

```
- (NSInteger)addButtonWithTitle:(NSString *)title
```

Parameters
title
The title of the new button.

Return Value
The index of the new button. Button indices start at 0 and increase in the order they are added.

Availability
Available in iPhone OS 2.0 and later.

See Also
UIActionSheet (page 67)

Declared In
UIAlert.h

buttonTitleAtIndex:

Returns the title of the button at the given index.

```
- (NSString *)buttonTitleAtIndex:(NSInteger)buttonIndex
```

Parameters
buttonIndex
The index of the button. The button indices start at 0.

Return Value
The title of the button specified by index *buttonIndex*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- showInView: (page 74)

Declared In
UIAlert.h

dismissWithClickedButtonIndex:animated:

Dismisses the receiver, optionally with animation.

```
- (void)dismissWithClickedButtonIndex:(NSInteger)buttonIndex animated:(BOOL)animated
```

Parameters

buttonIndex
The index of the button that was clicked just before invoking this method. The button indices start at 0.

animated
YES if the receiver should be removed by animating it first; otherwise, NO if it should be removed immediately with no animation.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle: otherButtonTitles:

Convenience method for initializing an action sheet.

```
- (id)initWithTitle:(NSString *)title delegate:(id < UIActionSheetDelegate >)delegate
    cancelButtonTitle:(NSString *)cancelButtonTitle destructiveButtonTitle:(NSString
    *)destructiveButtonTitle otherButtonTitles:(NSString *)otherButtonTitles, ...
```

Parameters

title
The string that appears in the receiver's title bar.

delegate
The receiver's delegate or nil if it doesn't have a delegate.

cancelButtonTitle
The title of the cancel button or nil if there is no cancel button.

Using this argument is equivalent to setting the cancel button index to the value returned by invoking showInView: (page 74) specifying this title.

destructiveButtonTitle
The index number of the destructive button.

otherButtonTitles,
The title of another button.

Using this argument is equivalent to invoking `showInView:` (page 74) with this title to add more buttons.

...
Titles of additional buttons to add to the receiver.

Return Value
Newly initialized action sheet.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- showInView:` (page 74)

Declared In
`UIAlert.h`

showFromTabBar:

Displays the receiver using animation by originating it from the specified tab bar.

```
- (void)showFromTabBar:(UITabBar *)view
```

Parameters
view
The tab bar that the receiver originates from.

Discussion
The style of the animation depends on the style of the tab bar, not the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- showFromToolbar:` (page 73)

Declared In
`UIAlert.h`

showFromToolbar:

Displays the receiver using animation by originating it from the specified toolbar.

```
- (void)showFromToolbar:(UIToolbar *)view
```

Parameters
view
The toolbar that the receiver originates from.

Discussion
The style of the animation depends on the style of the toolbar, not the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- showFromTabBar:` (page 73)

Declared In
`UIAlert.h`

showInView:

Displays the receiver using animation by originating it from the specified view.

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

Parameters

view
The view that the receiver originates from.

Discussion
The style of the animation depends on the `actionSheetStyle` (page 69) property.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAlert.h`

Constants

UIActionSheetStyle

Specifies the style of an action sheet.

```
typedef enum {
    UIActionSheetStyleAutomatic        = -1,
    UIActionSheetStyleDefault          = UIBarStyleDefault,
    UIActionSheetStyleBlackTranslucent = UIBarStyleBlackTranslucent,
    UIActionSheetStyleBlackOpaque      = UIBarStyleBlackOpaque,
} UIActionSheetStyle;
```

Constants

`UIActionSheetStyleAutomatic`
Takes the appearance of the bottom bar if specified; otherwise, same as `UIActionSheetStyleDefault` (page 75).

Available in iPhone OS 2.0 and later.

Declared in `UIAlert.h`

`UIActionSheetStyleDefault`
: The default style.

  Available in iPhone OS 2.0 and later.

  Declared in `UIAlert.h`

`UIActionSheetStyleBlackTranslucent`
: A black translucent style.

  Available in iPhone OS 2.0 and later.

  Declared in `UIAlert.h`

`UIActionSheetStyleBlackOpaque`
: A black opaque style.

  Available in iPhone OS 2.0 and later.

  Declared in `UIAlert.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAlert.h`

UIActivityIndicatorView Class Reference

Inherits from:	UIView
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIActivityIndicatorView.h

Overview

The `UIActivityIndicatorView` class creates and manages an indicator showing the indeterminate progress of a task. Visually, this indicator is a "gear" that is animated to spin.

You control when the progress indicator animates with the `startAnimating` (page 80) and `stopAnimating` (page 80) methods. If the `hidesWhenStopped` (page 78) property is set to `YES`, the indicator is automatically hidden when animation stops.

Tasks

Initializing an UIActivityIndicatorView Object

`- initWithActivityIndicatorStyle:` (page 79)
Initializes and returns an activity-indicator object.

Managing the Activity Indicator

`- startAnimating` (page 80)
Starts the animation of the progress indicator.

- `stopAnimating` (page 80)
  Stops the animation of the progress indicator.
- `isAnimating` (page 79)
  Returns whether the receiver is animating.

`hidesWhenStopped` (page 78) *property*
A Boolean value that controls whether the receiver is hidden when the animation is stopped.

Managing the Indicator Style

`activityIndicatorViewStyle` (page 78) *property*
The style of the indeterminate progress indicator.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

activityIndicatorViewStyle

The style of the indeterminate progress indicator.

```
@property UIActivityIndicatorViewStyle activityIndicatorViewStyle
```

Discussion
See `UIActivityIndicatorStyle` (page 80) for valid constants. The default value is `UIActivityIndicatorViewStyleWhite` (page 81).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIActivityIndicatorView.h`

hidesWhenStopped

A Boolean value that controls whether the receiver is hidden when the animation is stopped.

```
@property BOOL hidesWhenStopped
```

Discussion
If the value of this property is `YES` (the default), the receiver sets its `hidden` (page 450) property (`UIView`) to `YES` when receiver is not animating. If the `hidesWhenStopped` (page 78) property is `NO`, the receiver is not hidden when animation stops. You stop an animating progress indicator with the `stopAnimating` (page 80) method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIActivityIndicatorView.h

Instance Methods

initWithActivityIndicatorStyle:

Initializes and returns an activity-indicator object.

```
- (id)initWithActivityIndicatorStyle:(UIActivityIndicatorViewStyle)style
```

Parameters

style

A constant that specifies the style of the object to be created. See UIActivityIndicatorStyle (page 80) for descriptions of the style constants.

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

Discussion
UIActivityIndicatorView sizes the returned instance according to the specified *style*. You can set and retrieve the style of a activity indicator through the activityIndicatorViewStyle (page 78) property.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIActivityIndicatorView.h

isAnimating

Returns whether the receiver is animating.

```
- (BOOL)isAnimating
```

Return Value
YES if the receiver is animating, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- startAnimating (page 80)
- stopAnimating (page 80)

Declared In
UIActivityIndicatorView.h

startAnimating

Starts the animation of the progress indicator.

```
- (void)startAnimating
```

Discussion

When the progress indicator is animated, the gear spins to indicate indeterminate progress. The indicator is animated until `stopAnimating` (page 80) is called.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIActivityIndicatorView.h`

stopAnimating

Stops the animation of the progress indicator.

```
- (void)stopAnimating
```

Discussion

Call this method to stop the animation of the progress indicator started with a call to `startAnimating` (page 80). When animating is stopped, the indicator is hidden, unless unless `hidesWhenStopped` (page 78) is `NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIActivityIndicatorView.h`

Constants

UIActivityIndicatorStyle

The visual style of the progress indicator.

```
typedef enum {
    UIActivityIndicatorViewStyleWhiteLarge,
    UIActivityIndicatorViewStyleWhite,
    UIActivityIndicatorViewStyleGray,
} UIActivityIndicatorViewStyle;
```

Constants

`UIActivityIndicatorViewStyleWhiteLarge`

The large white style of indicator.

Available in iPhone OS 2.0 and later.

Declared in `UIActivityIndicatorView.h`

`UIActivityIndicatorViewStyleWhite`
The standard white style of indicator (the default).

Available in iPhone OS 2.0 and later.

Declared in `UIActivityIndicatorView.h`

`UIActivityIndicatorViewStyleGray`
The standard gray style of indicator.

Available in iPhone OS 2.0 and later.

Declared in `UIActivityIndicatorView.h`

Discussion
You set the value of the `activityIndicatorViewStyle` (page 78) property with these constants.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIActivityIndicatorView.h`

UIAlertView Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAlert.h

Overview

Use the `UIAlertView` class to display an alert message to the user. An alert view functions similar to but differs in appearance from an alert sheet.

Use the properties and methods defined in this class to set the title, message, and delegate of an alert view and configure the buttons. You must set a delegate if you add custom buttons. The delegate should conform to the `UIAlertViewDelegate` protocol. Use the `show` (page 88) method to display an alert view once it is configured.

Tasks

Creating Alert Views

`- initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:` (page 88)
Convenience method for initializing an alert view.

Setting Properties

`delegate` (page 85) *property*
The receiver's delegate or `nil` if it doesn't have a delegate.

`title` (page 86) *property*
The string that appears in the receiver's title bar.

`message` (page 85) *property*
Descriptive text that provides more details than the title.

`visible` (page 86) *property*
A Boolean value that indicates whether the receiver is displayed. (read-only)

Configuring Buttons

`- addButtonWithTitle:` (page 86)
Adds a button to the receiver with the given title.

`numberOfButtons` (page 85) *property*
The number of buttons on the alert view. (read-only)

`- buttonTitleAtIndex:` (page 87)
Returns the title of the button at the given index.

`cancelButtonIndex` (page 84) *property*
The index number of the cancel button.

`firstOtherButtonIndex` (page 85) *property*
The index of the first other button. (read-only)

Displaying

`- show` (page 88)
Displays the receiver using animation.

Dismissing

`- dismissWithClickedButtonIndex:animated:` (page 87)
Dismisses the receiver, optionally with animation.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

cancelButtonIndex

The index number of the cancel button.

```
@property(nonatomic) NSInteger cancelButtonIndex
```

Discussion

The button indices start at `0`. If `-1`, then the index is not set.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

delegate
The receiver's delegate or nil if it doesn't have a delegate.

```
@property(nonatomic, assign) id delegate
```

Discussion
See *UIAlertViewDelegate Protocol Reference* for the methods this delegate should implement.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

firstOtherButtonIndex
The index of the first other button. (read-only)

```
@property(nonatomic, readonly) NSInteger firstOtherButtonIndex
```

Discussion
The button indices start at 0. If -1, then the index is not set. This property is ignored if there are no other buttons. The default value is -1.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

message
Descriptive text that provides more details than the title.

```
@property(nonatomic, copy) NSString *message
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

numberOfButtons
The number of buttons on the alert view. (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfButtons
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

title

The string that appears in the receiver's title bar.

```
@property(nonatomic, copy) NSString *title
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

visible

A Boolean value that indicates whether the receiver is displayed. (read-only)

```
@property(nonatomic, readonly, getter=isVisible) BOOL visible
```

Discussion
If YES, the receiver is displayed; otherwise, NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

Instance Methods

addButtonWithTitle:

Adds a button to the receiver with the given title.

```
- (NSInteger)addButtonWithTitle:(NSString *)title
```

Parameters
title
The title of the new button.

Return Value
The index of the new button. Button indices start at 0 and increase in the order they are added.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property message` (page 85)

Declared In
`UIAlert.h`

buttonTitleAtIndex:

Returns the title of the button at the given index.

```
- (NSString *)buttonTitleAtIndex:(NSInteger)buttonIndex
```

Parameters

buttonIndex
The index of the button. The button indices start at `0`.

Return Value
The title of the button specified by index *buttonIndex*.

Availability
Available in iPhone OS 2.0 and later.

See Also
"Displaying" (page 84)

Declared In
`UIAlert.h`

dismissWithClickedButtonIndex:animated:

Dismisses the receiver, optionally with animation.

```
- (void)dismissWithClickedButtonIndex:(NSInteger)buttonIndex animated:(BOOL)animated
```

Parameters

buttonIndex
The index of the button that was clicked just before invoking this method. The button indices start at `0`.

animated
`YES` if the receiver should be removed by animating it first; otherwise, `NO` if it should be removed immediately with no animation.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIAlert.h`

initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:

Convenience method for initializing an alert view.

```
- (id)initWithTitle:(NSString *)title message:(NSString *)message
    delegate:(id)delegate cancelButtonTitle:(NSString *)cancelButtonTitle
    otherButtonTitles:(NSString *)otherButtonTitles, ...
```

Parameters

title

The string that appears in the receiver's title bar.

message

Descriptive text that provides more details than the title.

delegate

The receiver's delegate or `nil` if it doesn't have a delegate.

cancelButtonTitle

The title of the cancel button or `nil` if there is no cancel button.

Using this argument is equivalent to setting the cancel button index to the value returned by invoking `addButtonWithTitle:` (page 86) specifying this title.

otherButtonTitles,

The title of another button.

Using this argument is equivalent to invoking `addButtonWithTitle:` (page 86) with this title to add more buttons.

...

Titles of additional buttons to add to the receiver.

Return Value

Newly initialized alert view.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- addButtonWithTitle:` (page 86)

Declared In

`UIAlert.h`

show

Displays the receiver using animation.

```
- (void)show
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAlert.h`

UIApplication Class Reference

Inherits from:	UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIApplication.h

Overview

The `UIApplication` class provides a centralized point of control and coordination for applications running on iPhone OS.

Every application must have exactly one instance of `UIApplication` (or a subclass of `UIApplication`). When an application is launched, the `UIApplicationMain` (page 640) function is called; among its other tasks, this function create a singleton `UIApplication` object. Thereafter you can access this object by invoking the `sharedApplication` (page 96) class method.

A major role of a `UIApplication` object is to handle the initial routing of incoming user events. It also dispatches action messages forwarded to it by control objects (`UIControl`) to the appropriate target objects. In addition, the `UIApplication` object maintains a list of all the windows (`UIWindow` objects) currently open in the application, so through those it can retrieve any of the application's `UIView` objects. The application object is typically assigned a delegate, an object that the application informs of significant runtime events—for example, application launch, low-memory warnings, and application termination—giving it an opportunity to respond appropriately.

Applications can cooperatively handle a resource such as an email or an image file through the `openURL:` (page 98) method. For example, an application opening an email URL with this method may cause the mail client to launch and display the message.

`UIApplication` defines a delegate that must adopt the `UIApplicationDelegate` protocol implement one or more of the methods.

The programmatic interfaces of `UIApplication` and `UIApplicationDelegate` also allow you to manage behavior that is specific to the device. You can control application response to changes in interface orientation, temporarily suspend incoming user events, and turn proximity sensing (of the user's face) off and on again.

Subclassing Notes

You might decide to subclass `UIApplication` to override `sendEvent:` (page 99) or `sendAction:to:from:forEvent:` (page 98) to implement custom event and action dispatching.

Tasks

Getting the Application Instance

`+ sharedApplication` (page 96)
Returns the singleton application instance.

Getting Application Windows

`keyWindow` (page 93) *property*
The application's key window. (read-only)

`windows` (page 96) *property*
The application's visible windows. (read-only)

Controlling and Handling Events

`- sendEvent:` (page 99)
Dispatches an event to the appropriate responder objects in the application.

`- sendAction:to:from:forEvent:` (page 98)
Sends an action message identified by selector to a specified target.

`- beginIgnoringInteractionEvents` (page 97)
Tells the receiver to suspend the handling of touch-related events.

`- endIgnoringInteractionEvents` (page 97)
Tells the receiver to resume the handling of touch-related events.

`- isIgnoringInteractionEvents` (page 97)
Returns whether the receiver is ignoring events initiated by touches on the screen.

`proximitySensingEnabled` (page 94) *property*
A Boolean value that determines whether proximity sensing is enabled.

Opening a URL Resource

- `openURL:` (page 98)
  Opens the resource at the specified URL.

Managing Application Activity

`idleTimerDisabled` (page 92) *property*
A Boolean value that controls whether the idle timer is disabled for the application.

Managing Status Bar Orientation

- `setStatusBarOrientation:animated:` (page 100)
  Sets the application's status bar to the specified orientation, optionally animating the transition.

`statusBarOrientation` (page 95) *property*
The current orientation of the application's status bar.

`statusBarOrientationAnimationDuration` (page 95) *property*
The animation duration in seconds for the status bar during a 90 degree orientation change. (read-only)

Controlling Application Appearance

- `setStatusBarHidden:animated:` (page 99)
  Hides or shows the status bar, optionally animating the transition.

`statusBarHidden` (page 94) *property*
A Boolean value that determines whether the status bar is hidden.

- `setStatusBarStyle:animated:` (page 101)
  Sets the style of the status bar, optionally animating the transition to the new style

`statusBarStyle` (page 95) *property*
The current style of the status bar.

`statusBarFrame` (page 94) *property*
The frame rectangle defining the area of the status bar. (read-only)

`networkActivityIndicatorVisible` (page 93) *property*
A Boolean value that turns an indicator of network activity on or off.

`applicationIconBadgeNumber` (page 92) *property*
The number currently set as the badge of the application icon in Springboard.

Setting and Getting the Delegate

`delegate` (page 92) *property*
The delegate of the application object.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

applicationIconBadgeNumber

The number currently set as the badge of the application icon in Springboard.

```
@property(nonatomic) NSInteger applicationIconBadgeNumber
```

Discussion
Set to 0 (zero) to hide the badge number. The default is 0.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

delegate

The delegate of the application object.

```
@property(nonatomic, assign) id<UIApplicationDelegate> delegate
```

Discussion
The delegate must adopt the `UIApplicationDelegate` formal protocol. `UIApplication` assigns and does not retain the delegate.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

idleTimerDisabled

A Boolean value that controls whether the idle timer is disabled for the application.

```
@property(nonatomic, getter=isIdleTimerDisabled) BOOL idleTimerDisabled
```

Discussion
The default value of this property is `NO`. When most applications have no touches as user input for a short period, the system puts the device into a "sleep" state where the screen dims. This is done for the purposes of conserving power. However, applications that don't have user input except for the accelerometer—games, for instance—can, by setting this property to `YES`, disable the "idle timer" to avert system sleep.

Important: You should set this property only if necessary and should be sure to reset it to `NO` when the need no longer exists. Most applications should let the system turn off the screen when the idle timer elapses. The only applications that should disable the idle time are mapping applications, games, or similar programs with sporadic user interaction in the form of touches.

Availability
Available in iPhone OS 2.0 and later.

See Also

Declared In
`UIApplication.h`

keyWindow

The application's key window. (read-only)

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

Discussion
This property holds the `UIWindow` object in the `windows` (page 96) array that is most recently sent the `makeKeyAndVisible` (page 518) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property windows` (page 96)

Declared In
`UIApplication.h`

networkActivityIndicatorVisible

A Boolean value that turns an indicator of network activity on or off.

```
@property(nonatomic, getter=isNetworkActivityIndicatorVisible) BOOL
    networkActivityIndicatorVisible
```

Discussion
Specify `YES` if the application should show network activity and `NO` if it should not. The default value is `NO`. A spinning indicator in the status bar shows network activity. The application may explicitly hide or show this indicator.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

proximitySensingEnabled

A Boolean value that determines whether proximity sensing is enabled.

```
@property(nonatomic, getter=isProximitySensingEnabled) BOOL proximitySensingEnabled
```

Discussion

YES if proximity sensing is enabled; otherwise NO. Enabling proximity sensing tells iPhone OS that it may need to blank the screen if the user's face is near it. Proximity sensing is disabled by default.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIApplication.h

statusBarFrame

The frame rectangle defining the area of the status bar. (read-only)

```
@property(nonatomic, readonly) CGRect statusBarFrame
```

Discussion

The value of this property is CGRectZero if the status bar is hidden.

Availability

Available in iPhone OS 2.0 and later.

See Also

@property statusBarHidden (page 94)
@property statusBarStyle (page 95)

Declared In

UIApplication.h

statusBarHidden

A Boolean value that determines whether the status bar is hidden.

```
@property(nonatomic, getter=isStatusBarHidden) BOOL statusBarHidden
```

Return Value

YES means the status bar is hidden; NO means it's visible.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setStatusBarHidden:animated: (page 99)

Declared In

UIApplication.h

statusBarOrientation

The current orientation of the application's status bar.

```
@property(nonatomic) UIInterfaceOrientation statusBarOrientation
```

Discussion

The value of this property is a constant that indicates an orientation of the receiver's status bar. See UIInterfaceOrientation (page 101) for details. Setting this property rotates the status bar to the specified orientation without animating the transition. If your application has rotatable window content, however, you should not arbitrarily set status-bar orientation using this method. The status-bar orientation set by this method does not change if the device changes orientation. For more on rotatable window view, see the `UIWindow` class reference.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setStatusBarOrientation:animated:` (page 100)

Declared In

`UIApplication.h`

statusBarOrientationAnimationDuration

The animation duration in seconds for the status bar during a 90 degree orientation change. (read-only)

```
@property(nonatomic, readonly) NSTimeInterval statusBarOrientationAnimationDuration
```

Discussion

You should double the value of this property for a 180 degree orientation change in the status bar.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setStatusBarOrientation:animated:` (page 100)

Declared In

`UIApplication.h`

statusBarStyle

The current style of the status bar.

```
@property(nonatomic) UIStatusBarStyle statusBarStyle
```

Discussion

The value of the property is a `statusBarOrientationAnimationDuration` (page 95) constant that indicates the style of status; see UIStatusBarStyle (page 102) for a description of these constants. The default style is `UIStatusBarStyleDefault` (page 102). The animation slides the status bar out for the old orientation and slides it in for the new orientation.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property statusBarHidden (page 94)
@property statusBarFrame (page 94)

Declared In
UIApplication.h

windows

The application's visible windows. (read-only)

```
@property(nonatomic, readonly) NSArray *windows
```

Discussion
This property is an array holding the application's visible windows; the windows are ordered front to back.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property keyWindow (page 93)

Declared In
UIApplication.h

Class Methods

sharedApplication

Returns the singleton application instance.

```
+ (UIApplication *)sharedApplication
```

Return Value
The application instance is created in the UIApplicationMain (page 640) function.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIApplication.h

Instance Methods

beginIgnoringInteractionEvents

Tells the receiver to suspend the handling of touch-related events.

```
- (void)beginIgnoringInteractionEvents
```

Discussion

You typically call this method before starting an animation or transition. Calls are nested with the `endIgnoringInteractionEvents` (page 97) method.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isIgnoringInteractionEvents` (page 97)

Declared In

`UIApplication.h`

endIgnoringInteractionEvents

Tells the receiver to resume the handling of touch-related events.

```
- (void)endIgnoringInteractionEvents
```

Discussion

You typically call this method when, after calling the `beginIgnoringInteractionEvents` (page 97) method, the animation or transition concludes. Nested calls of this method should match nested calls of the `beginIgnoringInteractionEvents` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isIgnoringInteractionEvents` (page 97)

Declared In

`UIApplication.h`

isIgnoringInteractionEvents

Returns whether the receiver is ignoring events initiated by touches on the screen.

```
- (BOOL)isIgnoringInteractionEvents
```

Return Value

`YES` if the receiver is ignoring interaction events; otherwise `NO`. The method returns `YES` if the nested `beginIgnoringInteractionEvents` (page 97) and `endIgnoringInteractionEvents` (page 97) calls are at least one level deep.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIApplication.h

openURL:

Opens the resource at the specified URL.

```
- (BOOL)openURL:(NSURL *)url
```

Parameters
url

An object representing a URL (Universal Resource Locator). UIKit supports the http:, https:, tel:, and mailto: schemes.

Return Value
YES if the resource located by the URL was successfully opened; otherwise NO.

Discussion
The URL can locate a resource in the same or other application. If the resource is another application, invoking this method may cause the calling application to quit so the other one can be launched.

Availability
Available in iPhone OS 2.0 and later.

See Also
- application:handleOpenURL: (page ?)

Declared In
UIApplication.h

sendAction:to:from:forEvent:

Sends an action message identified by selector to a specified target.

```
- (BOOL)sendAction:(SEL)action to:(id)target from:(id)sender forEvent:(UIEvent
    *)event
```

Parameters
action

A selector identifying an action method. See the discussion for information on the permitted selector forms.

target

The object to receive the action message. If *target* is nil, the application sends the message to the first responder, from whence it progresses up the responder chain until it is handled.

sender

The object that is sending the action message. The default sender is the UIControl object that invokes this method.

event

A UIEvent object that encapsulates information about the event originating the action message.

Return Value
YES if a responder object handled the action message, NO if no object in the responder chain handled the message.

Discussion
Normally, this method is invoked by a UIControl object that the user has touched. The default implementation dispatches the action method to the given target object or, if no target is specified, to the first responder. Subclasses may override this method to perform special dispatching of action messages.

By default, this method pushes two parameters when calling the target. These last two parameteres are optional for the receiver because it is up to the caller (usually a UIControl object) to remove any parameters it added. This design enables the action selector to be one of the following:

```
- (void)action
- (void)action:(id)sender
- (void)action:(id)sender forEvent:(UIEvent *)event
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- sendEvent: (page 99)

Declared In
UIApplication.h

sendEvent:

Dispatches an event to the appropriate responder objects in the application.

```
- (void)sendEvent:(UIEvent *)event
```

Parameters
event

A UIEvent object encapsulating the information about an event, including the touches involved.

Discussion
Subclasses may override this method to intercept incoming events for inspection and special dispatching. Apsen calls this method for public events only.

Availability
Available in iPhone OS 2.0 and later.

See Also
- sendAction:to:from:forEvent: (page 98)

Declared In
UIApplication.h

setStatusBarHidden:animated:

Hides or shows the status bar, optionally animating the transition.

```
- (void)setStatusBarHidden:(BOOL)hidden animated:(BOOL)animated
```

Parameters

hidden

YES if the status bar should be hidden, NO if it should be visible. The default value is NO.

animated

YES if the transition to or from a hidden state should be animated, NO otherwise.

Discussion

The animation slides the status bar out toward the top of the interface.

Availability

Available in iPhone OS 2.0 and later.

See Also

@property statusBarHidden (page 94)

Declared In

UIApplication.h

setStatusBarOrientation:animated:

Sets the application's status bar to the specified orientation, optionally animating the transition.

```
- (void)setStatusBarOrientation:(UIInterfaceOrientation)interfaceOrientation
    animated:(BOOL)animated
```

Parameters

interfaceOrientation

A specific orientation of the status bar. See UIInterfaceOrientation (page 101) for details. The default value is UIInterfaceOrientationPortrait (page 101).

animated

YES if the transition to the new orientation should be animated; NO if it should be immediate, without animation.

Discussion

Calling this method changes the value of the statusBarOrientation (page 95) property and rotates the status bar, animating the transition if animated is YES . If your application has rotatable window content, however, you should not arbitrarily set status-bar orientation using this method. The status-bar orientation set by this method does not change if the device changes orientation.

Availability

Available in iPhone OS 2.0 and later.

See Also

@property statusBarOrientation (page 95)

@property statusBarOrientationAnimationDuration (page 95)

Declared In

UIApplication.h

setStatusBarStyle:animated:

Sets the style of the status bar, optionally animating the transition to the new style

```
- (void)setStatusBarStyle:(UIStatusBarStyle)statusBarStyle animated:(BOOL)animated
```

Parameters

statusBarStyle

A constant that specifies a style for the status bar. See the descriptions of the constants in UIStatusBarStyle (page 102) for details.

animated

`YES` if the transition to the new style should be animated; otherwise `NO` .

Discussion

The animation slides the status bar out toward the top of the interface.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property statusBarStyle` (page 95)

Declared In

`UIApplication.h`

Constants

UIInterfaceOrientation

The orientation of the application's user interface.

```
typedef enum {
    UIInterfaceOrientationPortrait           = UIDeviceOrientationPortrait,
   UIInterfaceOrientationPortraitUpsideDown = UIDeviceOrientationPortraitUpsideDown,
    UIInterfaceOrientationLandscapeLeft      = UIDeviceOrientationLandscapeLeft,
    UIInterfaceOrientationLandscapeRight     = UIDeviceOrientationLandscapeRight
} UIInterfaceOrientation;
```

Constants

`UIInterfaceOrientationPortrait`

Displays the user interface in regular portrait mode (home button on the bottom).

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIInterfaceOrientationPortraitUpsideDown`

Displays the user interface in upside-down portrait mode (home button on the top).

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIInterfaceOrientationLandscapeLeft`
Displays the user interface in landscape mode with the home button on the left).

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIInterfaceOrientationLandscapeRight`
Displays the user interface in landscape mode with the home button on the right).

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

Discussion
You use these constants in the `statusBarOrientation` (page 95) property and the `setStatusBarOrientation:animated:` (page 100) method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIStatusBarStyle

The style of the device's status bar.

```
typedef enum {
    UIStatusBarStyleDefault,
    UIStatusBarStyleBlackTranslucent,
    UIStatusBarStyleBlackOpaque
} UIStatusBarStyle;
```

Constants
`UIStatusBarStyleDefault`
A gray style (the default).

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIStatusBarStyleBlackTranslucent`
A transparent black style (specifically, black with an alpha of 0.5).

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIStatusBarStyleBlackOpaque`
An opaque black style.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

Run Loop Mode for Tracking

Mode while tracking in controls is taking place.

```
UIKIT_EXTERN NSString *UITrackingRunLoopMode;
```

Constants

`UITrackingRunLoopMode`

The mode set while tracking in controls takes place. You can use this mode to add timers that fire during tracking.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

Declared In

`UIApplication.h`

Interface Orientation Conveniences

Convenience macros for determining interface orientation status.

```
#define UIDeviceOrientationIsValidInterfaceOrientation(orientation) ((orientation)
 == UIDeviceOrientationPortrait || (orientation) ==
UIDeviceOrientationPortraitUpsideDown || (orientation) ==
UIDeviceOrientationLandscapeLeft || (orientation) ==
UIDeviceOrientationLandscapeRight)
#define UIInterfaceOrientationIsPortrait(orientation)  ((orientation) ==
UIInterfaceOrientationPortrait || (orientation) ==
UIInterfaceOrientationPortraitUpsideDown)
#define UIInterfaceOrientationIsLandscape(orientation) ((orientation) ==
UIInterfaceOrientationLandscapeLeft || (orientation) ==
UIInterfaceOrientationLandscapeRight)
```

Constants

`UIDeviceOrientationIsValidInterfaceOrientation`

Returns a Boolean value indicating whether the current orientation constant is valid.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIInterfaceOrientationIsPortrait`

Returns a Boolean value indicating whether the current orientation is portrait.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIInterfaceOrientationIsLandscape`

Returns a Boolean value indicating whether the current orientation is landscape.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

Declared In

`UIApplication.h`

Notification UserInfo Dictionary Keys

Keys used to access values in the `userInfo` dictionary of some `UIApplication`-posted notifications.

```
UIKIT_EXTERN NSString *const UIApplicationStatusBarOrientationUserInfoKey;
UIKIT_EXTERN NSString *const UIApplicationStatusBarFrameUserInfoKey;
```

Constants

`UIApplicationStatusBarOrientationUserInfoKey`

Accesses an `NSNumber` object that encapsulates a `UIInterfaceOrientation` value indicating the current orientation (see UIInterfaceOrientation (page 101)). This key is used with `UIApplicationDidChangeInterfaceOrientationNotification` (page 104) and `UIApplicationWillChangeInterfaceOrientationNotification` (page 105) notifications.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

`UIApplicationStatusBarFrameUserInfoKey`

Accesses an `NSValue` object that encapsulates a `CGRect` structure expressing the location and size of the new status bar frame. This key is used with `UIApplicationDidChangeStatusBarFrameNotification` (page 105) and `UIApplicationWillChangeStatusBarFrameNotification` (page 106) notifications.

Available in iPhone OS 2.0 and later.

Declared in `UIApplication.h`

Declared In

`UIApplication.h`

Notifications

All `UIApplication` notifications are posted by the application instance returned by `sharedApplication` (page 96).

UIApplicationDidBecomeActiveNotification

Posted when the application becomes active.

An application is active when it is receiving events. An active application can be said to have focus. It gains focus after being launched, loses focus when an overlay window pops up or when the device is locked, and gains focus when the device is unlocked.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIApplication.h`

UIApplicationDidChangeInterfaceOrientationNotification

Posted when the orientation of the application's user interface changes.

The userInfo dictionary contains an `NSNumber` object that encapsulates a `UIInterfaceOrientation` value (see UIInterfaceOrientation (page 101)). Use `UIApplicationStatusBarOrientationUserInfoKey` (page 104) to access this value

UIApplicationDidChangeStatusBarFrameNotification

Posted when the frame of the status bar changes.

The userInfo dictionary contains an `NSValue` object that encapsulates a `CGRect` structure expressing the location and size of the new status bar frame. Use `UIApplicationStatusBarFrameUserInfoKey` (page 104) to access this value.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationDidFinishLaunchingNotification

Posted immediately after the application finishes launching.

This notification does not contain a userInfo dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationDidReceiveMemoryWarningNotification

Posted when the application receives a warning from the operating system about low memory availability.

This notification does not contain a userInfo dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationSignificantTimeChangeNotification

Posted when there is a significant change in time, for example, change to a new day (midnight), carrier time update, and change to or from daylight savings time.

This notification does not contain a userInfo dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationWillChangeInterfaceOrientationNotification

Posted when the application is about to change the orientation of its interface.

The userInfo dictionary contains an `NSNumber` that encapsulates a `UIInterfaceOrientation` value (see UIInterfaceOrientation (page 101)). Use `UIApplicationStatusBarOrientationUserInfoKey` (page 104) to access this value.

UIApplicationWillChangeStatusBarFrameNotification

Posted when the application is about to change the frame of the status bar.

The userInfo dictionary contains an `NSValue` object that encapsulates a `CGRect` structure expressing the location and size of the new status bar frame. Use `UIApplicationStatusBarFrameUserInfoKey` (page 104) to access this value.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationWillResignActiveNotification

Posted when the application is no longer active and loses focus.

An application is active when it is receiving events. An active application can be said to have focus. It gains focus after being launched, loses focus when an overlay window pops up or when the device is locked, and gains focus when the device is unlocked.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationWillTerminateNotification

Posted when the application is about to terminate.

This notification does not contain a userInfo dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIApplication.h`

UIBarButtonItem Class Reference

Inherits from:	UIBarItem : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIBarButtonItem.h

Overview

The `UIBarButtonItem` class encapsulates the properties and behaviors of items added to `UIToolbar` and `UINavigationBar` objects. It inherits basic button behavior from its parent class. This class defines additional initialization methods and properties for use on tab bars and navigation bars that allow more custom views.

Tasks

Initializing an Item

`- initWithBarButtonSystemItem:target:action:` (page 110)
Creates and returns a new item containing the specified system item.

`- initWithCustomView:` (page 110)
Creates and returns a new item using the specified custom view.

`- initWithImage:style:target:action:` (page 111)
Creates and returns a new item using the specified image and other properties.

`- initWithTitle:style:target:action:` (page 111)
Creates and returns a new item using the specified title and other properties.

Getting and Setting Properties

`target` (page 109) *property*
The object that receives an action when the item is selected.

`action` (page 108) *property*
The selector defining the action message to send to the target object when the user taps this bar button item.

`style` (page 109) *property*
The style of the item.

`possibleTitles` (page 109) *property*
Collection of possible titles to display on the bar.

`width` (page 110) *property*
The width of the item.

`customView` (page 108) *property*
A custom view representing the item.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

action

The selector defining the action message to send to the target object when the user taps this bar button item.

```
@property(nonatomic) SEL action
```

Discussion
If the value of this property is `NULL`, no action message is sent. The default value is `NULL`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property target` (page 109)

Declared In
`UIBarButtonItem.h`

customView

A custom view representing the item.

```
@property(nonatomic, retain) UIView *customView
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIBarButtonItem.h

possibleTitles

Collection of possible titles to display on the bar.

```
@property(nonatomic, copy) NSSet *possibleTitles
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIBarButtonItem.h

style

The style of the item.

```
@property(nonatomic) UIBarButtonItemStyle style
```

Discussion
One of the constants defined in `UIBarButtonItemStyle` (page 115). The default value is `UIBarButtonItemStylePlain` (page 116).

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIBarButtonItem.h

target

The object that receives an action when the item is selected.

```
@property(nonatomic, assign) id target
```

Discussion
If `nil`, an object in the responder chain can respond to the `initWithBarButtonSystemItem:target:action:` (page 110) message. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property action` (page 108)

Declared In
UIBarButtonItem.h

width

The width of the item.

```
@property(nonatomic) CGFloat width
```

Discussion

If this property value is positive, the width of the combined image and title are fixed. If the value is `0.0` or negative, the item sets the width of the combined image and title to fit. This property is ignored if the style uses radio mode. The default value is `0.0`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIBarButtonItem.h`

Instance Methods

initWithBarButtonSystemItem:target:action:

Creates and returns a new item containing the specified system item.

```
- (id)initWithBarButtonSystemItem:(UIBarButtonSystemItem)systemItem target:(id)target
    action:(SEL)action
```

Parameters

systemItem

The system item to use as the first item on the bar. One of the constants defined in `UIBarButtonSystemItem` (page 112).

target

The object that receives the *action* message.

action

The action to send to *target* when this item is selected.

Return Value

A newly initialized item containing the specified system item. The item's target is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithImage:style:target:action:` (page 111)
`- initWithTitle:style:target:action:` (page 111)

Declared In

`UIBarButtonItem.h`

initWithCustomView:

Creates and returns a new item using the specified custom view.

```
- (id)initWithCustomView:(UIView *)customView
```

Parameters

customView

A custom view representing the item.

Return Value

Newly initialized item with the specified properties.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIBarButtonItem.h

initWithImage:style:target:action:

Creates and returns a new item using the specified image and other properties.

```
- (id)initWithImage:(UIImage *)image style:(UIBarButtonItemStyle)style
    target:(id)target action:(SEL)action
```

Parameters

image

The item's image. If `nil` an image is not displayed.

The images displayed on the bar are derived from this image. If this image is too large to fit on the bar, it is scaled to fit. Typically, the size of a toolbar and navigation bar image is 20 x 20 points. The alpha values in the source image are used to create the images—opaque values are ignored.

style

The style of the item. One of the constants defined in `UIBarButtonItemStyle` (page 115).

target

The object that receives the *action* message.

action

The action to send to *target* when this item is selected.

Return Value

Newly initialized item with the specified properties.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithBarButtonSystemItem:target:action:` (page 110)
- `initWithTitle:style:target:action:` (page 111)

Declared In

UIBarButtonItem.h

initWithTitle:style:target:action:

Creates and returns a new item using the specified title and other properties.

```
- (id)initWithTitle:(NSString *)title style:(UIBarButtonItemStyle)style
    target:(id)target action:(SEL)action
```

Parameters

title

The item's title. If `nil` a title is not displayed.

style

The style of the item. One of the constants defined in `UIBarButtonItemStyle` (page 115).

target

The object that receives the *action* message.

action

The action to send to *target* when this item is selected.

Return Value

Newly initialized item with the specified properties.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithBarButtonSystemItem:target:action:` (page 110)

`- initWithImage:style:target:action:` (page 111)

Declared In

`UIBarButtonItem.h`

Constants

UIBarButtonSystemItem

Defines system defaults for commonly used items.

```
typedef enum {
    UIBarButtonSystemItemDone,
    UIBarButtonSystemItemCancel,
    UIBarButtonSystemItemEdit,
    UIBarButtonSystemItemSave,
    UIBarButtonSystemItemAdd,
    UIBarButtonSystemItemFlexibleSpace,
    UIBarButtonSystemItemFixedSpace,
    UIBarButtonSystemItemCompose,
    UIBarButtonSystemItemReply,
    UIBarButtonSystemItemAction,
    UIBarButtonSystemItemOrganize,
    UIBarButtonSystemItemBookmarks,
    UIBarButtonSystemItemSearch,
    UIBarButtonSystemItemRefresh,
    UIBarButtonSystemItemStop,
    UIBarButtonSystemItemCamera,
    UIBarButtonSystemItemTrash,
    UIBarButtonSystemItemPlay,
    UIBarButtonSystemItemPause,
    UIBarButtonSystemItemRewind,
    UIBarButtonSystemItemFastForward,
} UIBarButtonSystemItem;
```

Constants

`UIBarButtonSystemItemDone`

The system Done button. Localized.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemCancel`

The system Cancel button. Localized.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemEdit`

The system Edit button. Localized.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemSave`

The system Save button. Localized.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemAdd`

The system plus button containing an icon of a plus sign. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemFlexibleSpace`

Blank space to add between other items. The space is distributed equally between the other items. Other item properties are ignored when this value is set.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemFixedSpace`

Blank space to add between other items. Only the `width` (page 110) property is used when this value is set.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemCompose`

The system compose button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemReply`

The system reply button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemAction`

The system action button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemOrganize`

The system organize button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemBookmarks`

The system bookmarks button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemSearch`

The system search button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemRefresh`

The system refresh button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemStop`

The system stop button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemCamera`

The system camera button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemTrash`

The system trash button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemPlay`

The system play button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemPause`

The system pause button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemRewind`

The system rewind button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonSystemItemFastForward`

The system fast forward button. 

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIBarButtonItem.h`

UIBarButtonItemStyle

Specifies the style of a item.

```
typedef enum {
    UIBarButtonItemStylePlain,
    UIBarButtonItemStyleBordered,
    UIBarButtonItemStyleDone,
} UIBarButtonItemStyle;
```

Constants

`UIBarButtonItemStylePlain`

Glows when tapped. The default item style.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonItemStyleBordered`

A simple button style with a border.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

`UIBarButtonItemStyleDone`

The style for a done button—for example, a button that completes some task and returns to the previous view.

Available in iPhone OS 2.0 and later.

Declared in `UIBarButtonItem.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIBarButtonItem.h`

UIBarItem Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIBarItem.h

Overview

`UIBarItem` is an abstract superclass for items added to a bar that appears at the bottom of the screen. Items on a bar behave in a way similar to buttons. They have a title, image, action, and target. You can also enable and disable an item on a bar.

Tasks

Getting and Setting Properties

`enabled` (page 118) *property*
A Boolean value indicating whether the item is enabled.

`image` (page 118) *property*
The image used to represent the item.

`imageInsets` (page 118) *property*
The image inset or outset for each edge.

`title` (page 119) *property*
The title displayed on the item.

`tag` (page 119) *property*
The receiver's tag, an application-supplied integer that you can use to identify bar item objects in your application.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

enabled

A Boolean value indicating whether the item is enabled.

```
@property(nonatomic, getter=isEnabled) BOOL enabled
```

Discussion

If `YES`, the item appears dim. The default value is `YES`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIBarItem.h`

image

The image used to represent the item.

```
@property(nonatomic, retain) UIImage *image
```

Discussion

This image can be used to create other images to represent this item on the bar—for example, a selected and unselected image may be derived from this image. You should set this property before adding the item to a bar. The default value is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property imageInsets` (page 118)

Declared In

`UIBarItem.h`

imageInsets

The image inset or outset for each edge.

```
@property(nonatomic) UIEdgeInsets imageInsets
```

Discussion

The default value is `UIEdgeInsetsZero` (page 629).

Availability

Available in iPhone OS 2.0 and later.

See Also
`@property image` (page 118)

Declared In
`UIBarItem.h`

tag

The receiver's tag, an application-supplied integer that you can use to identify bar item objects in your application.

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

Discussion
The default value is `0`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIBarItem.h`

title

The title displayed on the item.

```
@property(nonatomic, copy) NSString *title
```

Discussion
You should set this property before adding the item to a bar. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIBarItem.h`

UIButton Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIButton.h

Overview

The `UIButton` class is a `UIControl` subclass that implements a button on the touch screen. A button intercepts touch events and sends an action message to a target object when it's tapped. Methods for setting the target and action are inherited from `UIControl`. This class provides methods for setting the title, image, and other appearance properties of a button. By using the set methods, you can specify a different appearance for each button state.

Tasks

Creating Buttons

`+ buttonWithType:` (page 129)
Creates and returns a new button of the specified type.

Configuring Button Title

`buttonType` (page 124) *property*
The button type. (read-only)

`font` (page 126) *property*
The font used to display text on the button.

`lineBreakMode` (page 127) *property*
The line break mode to use when drawing text.

`titleShadowOffset` (page 129) *property*
The offset of the shadow used to display the receiver's title.

`reversesTitleShadowWhenHighlighted` (page 127) *property*
A Boolean value that determines whether the title shadow changes when the button is highlighted.

- `setTitle:forState:` (page 133)
Sets the title to use for the specified state.

- `setTitleColor:forState:` (page 133)
Sets the color of the title to use for the specified state.

- `setTitleShadowColor:forState:` (page 134)
Sets the color of the title shadow to use for the specified state.

- `titleColorForState:` (page 134)
Returns the title color used for a state.

- `titleForState:` (page 134)
Returns the title used for a state.

- `titleShadowColorForState:` (page 135)
Returns the shadow color of the title used for a state.

Configuring Button Images

`adjustsImageWhenHighlighted` (page 124) *property*
A Boolean value that determines whether the image changes when the button is highlighted.

`adjustsImageWhenDisabled` (page 123) *property*
A Boolean value that determines whether the image changes when the button is disabled.

`showsTouchWhenHighlighted` (page 128) *property*
A Boolean value that determines whether tapping the button causes it to glow.

- `backgroundImageForState:` (page 129)
Returns the background image used for a button state.

- `imageForState:` (page 131)
Returns the image used for a button state.

- `setBackgroundImage:forState:` (page 132)
Sets the background image to use for the specified button state.

- `setImage:forState:` (page 132)
Sets the image to use for the specified state.

Configuring Edge Insets

`titleEdgeInsets` (page 128) *property*
The title inset or outset for each edge.

`imageEdgeInsets` (page 127) *property*
The image inset or outset for each edge.

`contentEdgeInsets` (page 124) *property*
The content inset or outset for each edge.

Getting the Current State

`currentTitle` (page 125) *property*
The current title that is displayed on the button. (read-only)

`currentTitleColor` (page 126) *property*
The color used to display the title. (read-only)

`currentTitleShadowColor` (page 126) *property*
The color of the title's shadow. (read-only)

`currentImage` (page 125) *property*
The current image displayed on the button. (read-only)

`currentBackgroundImage` (page 125) *property*
The current background image displayed on the button. (read-only)

Getting Dimensions

`- backgroundRectForBounds:` (page 130)
Returns the rectangle where the receiver draws its background.

`- contentRectForBounds:` (page 130)
Returns the rectangle where the receiver draws its entire content.

`- titleRectForContentRect:` (page 135)
Returns the rectangle where the receiver draws its title.

`- imageRectForContentRect:` (page 131)
Returns the rectangle where the receiver draws its image.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

adjustsImageWhenDisabled

A Boolean value that determines whether the image changes when the button is disabled.

```
@property(nonatomic) BOOL adjustsImageWhenDisabled
```

Discussion

If `YES`, the image is drawn darker when the button is disabled. The default value is `YES`.

Availability

Available in iPhone OS 2.0 and later.

See Also
`@property adjustsImageWhenHighlighted` (page 124)

Declared In
`UIButton.h`

adjustsImageWhenHighlighted

A Boolean value that determines whether the image changes when the button is highlighted.

```
@property(nonatomic) BOOL adjustsImageWhenHighlighted
```

Discussion
If `YES`, the image is drawn lighter when the button is highlighted. The default value is `YES`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property adjustsImageWhenDisabled` (page 123)

Declared In
`UIButton.h`

buttonType

The button type. (read-only)

```
@property(nonatomic, readonly) UIButtonType buttonType
```

Discussion
See `UIButtonType` (page 136) for the possible values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIButton.h`

contentEdgeInsets

The content inset or outset for each edge.

```
@property(nonatomic) UIEdgeInsets contentEdgeInsets
```

Discussion
If the values for each edge are positive, specifies the inset; otherwise, specifies the outset. An inset is a margin around the drawing rectangle; each side (left, right, top, and bottom) can have a different value. Use the `UIEdgeInsetsMake` (page 642) function to set this property. The default value is `UIEdgeInsetsZero` (page 629).

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property imageEdgeInsets` (page 127)

Declared In
`UIButton.h`

currentBackgroundImage

The current background image displayed on the button. (read-only)

```
@property(nonatomic, readonly, retain) UIImage *currentBackgroundImage
```

Discussion
This value can be `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property currentImage` (page 125)

Declared In
`UIButton.h`

currentImage

The current image displayed on the button. (read-only)

```
@property(nonatomic, readonly, retain) UIImage *currentImage
```

Discussion
This value can be `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property currentBackgroundImage` (page 125)

Declared In
`UIButton.h`

currentTitle

The current title that is displayed on the button. (read-only)

```
@property(nonatomic, readonly, retain) NSString *currentTitle
```

Discussion
This value may be `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property currentTitleColor` (page 126)
`@property currentTitleShadowColor` (page 126)

Declared In
`UIButton.h`

currentTitleColor

The color used to display the title. (read-only)

```
@property(nonatomic, readonly, retain) UIColor *currentTitleColor
```

Discussion
This value is guaranteed not to be `nil`. The default value is `white`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property currentTitle` (page 125)
`@property currentTitleShadowColor` (page 126)

Declared In
`UIButton.h`

currentTitleShadowColor

The color of the title's shadow. (read-only)

```
@property(nonatomic, readonly, retain) UIColor *currentTitleShadowColor
```

Discussion
The default value is `white`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property currentTitle` (page 125)
`@property currentTitleColor` (page 126)

Declared In
`UIButton.h`

font

The font used to display text on the button.

```
@property(nonatomic, retain) UIFont *font
```

Discussion
If `nil`, a system font is used. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIButton.h`

imageEdgeInsets

The image inset or outset for each edge.

```
@property(nonatomic) UIEdgeInsets imageEdgeInsets
```

Discussion
If the values for each edge are positive, specifies the inset; otherwise, specifies the outset. An inset is a margin around the drawing rectangle; each side (left, right, top, and bottom) can have a different value. Use the `UIEdgeInsetsMake` (page 642) function to set this property. The default value is `UIEdgeInsetsZero` (page 629).

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property titleEdgeInsets` (page 128)

Declared In
`UIButton.h`

lineBreakMode

The line break mode to use when drawing text.

```
@property(nonatomic) UILineBreakMode lineBreakMode
```

Discussion
This property is one of the constants described in `UILineBreakMode` (page 50). The default value is `UILineBreakModeMiddleTruncation` (page 51).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIButton.h`

reversesTitleShadowWhenHighlighted

A Boolean value that determines whether the title shadow changes when the button is highlighted.

```
@property(nonatomic) BOOL reversesTitleShadowWhenHighlighted
```

Discussion
If `YES`, the shadow changes from engrave to emboss appearance when highlighted. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIButton.h`

showsTouchWhenHighlighted

A Boolean value that determines whether tapping the button causes it to glow.

```
@property(nonatomic) BOOL showsTouchWhenHighlighted
```

Discussion
If `YES`, the button glows when tapped; otherwise, it does not. The image and button behavior is not changed by the glow. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property adjustsImageWhenHighlighted` (page 124)

Declared In
`UIButton.h`

titleEdgeInsets

The title inset or outset for each edge.

```
@property(nonatomic) UIEdgeInsets titleEdgeInsets
```

Discussion
If the values for each edge are positive, specifies the inset; otherwise, specifies the outset. An inset is a margin around the drawing rectangle; each side (left, right, top, and bottom) can have a different value. Use the `UIEdgeInsetsMake` (page 642) function to set this property. The default value is `UIEdgeInsetsZero` (page 629).

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property imageEdgeInsets` (page 127)

Declared In
`UIButton.h`

titleShadowOffset

The offset of the shadow used to display the receiver's title.

```
@property(nonatomic) CGSize titleShadowOffset
```

Discussion

The horizontal and vertical offset values, specified using the `width` and `height` fields of the `CGSize` data type. Positive values always extend up and to the right from the user's perspective. The default value is `CGSizeZero`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIButton.h`

Class Methods

buttonWithType:

Creates and returns a new button of the specified type.

```
+ (id)buttonWithType:(UIButtonType)buttonType
```

Parameters

buttonType

The button type. See `UIButtonType` (page 136) for the possible values.

Return Value

A newly created button.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIButton.h`

Instance Methods

backgroundImageForState:

Returns the background image used for a button state.

```
- (UIImage *)backgroundImageForState:(UIControlState)state
```

Parameters

state

The state that uses the background image. Possible values are described in `UIControlState` (page 174).

Return Value
The background image used for the specified state.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setBackgroundImage:forState:` (page 132)

Declared In
`UIButton.h`

backgroundRectForBounds:

Returns the rectangle where the receiver draws its background.

```
- (CGRect)backgroundRectForBounds:(CGRect)bounds
```

Parameters
bounds
The bounding rectangle of the receiver.

Return Value
The rectangle where the receiver draws its background.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `contentRectForBounds:` (page 130)

Declared In
`UIButton.h`

contentRectForBounds:

Returns the rectangle where the receiver draws its entire content.

```
- (CGRect)contentRectForBounds:(CGRect)bounds
```

Parameters
bounds
The bounding rectangle for the receiver.

Return Value
The rectangle where the receiver draws its entire content.

Discussion
The content rectangle is the area needed to display the image and title including any padding and adjustments for alignment and other settings.

Availability
Available in iPhone OS 2.0 and later.

See Also
- titleRectForContentRect: (page 135)
- imageRectForContentRect: (page 131)
- backgroundRectForBounds: (page 130)

Declared In
UIButton.h

imageForState:

Returns the image used for a button state.

```
- (UIImage *)imageForState:(UIControlState)state
```

Parameters

state

The state that uses the image. Possible values are described in UIControlState (page 174).

Return Value
The image used for the specified state.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setImage:forState: (page 132)

Declared In
UIButton.h

imageRectForContentRect:

Returns the rectangle where the receiver draws its image.

```
- (CGRect)imageRectForContentRect:(CGRect)contentRect
```

Parameters

contentRect

The content rectangle for the receiver.

Return Value
The rectangle where the receiver draws its image.

Availability
Available in iPhone OS 2.0 and later.

See Also
- contentRectForBounds: (page 130)
- titleRectForContentRect: (page 135)

Declared In
UIButton.h

setBackgroundImage:forState:

Sets the background image to use for the specified button state.

```
- (void)setBackgroundImage:(UIImage *)image forState:(UIControlState)state
```

Parameters

image

The background image to use for the specified state.

state

The state that uses the specified image. The values are described in `UIControlState` (page 174).

Discussion

In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 173) value. If the `UIControlStateNormal` value is not set, then the property defaults to a system value. Therefore, at a minimum, you should set the value for the normal state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- backgroundImageForState:` (page 129)

Declared In

`UIButton.h`

setImage:forState:

Sets the image to use for the specified state.

```
- (void)setImage:(UIImage *)image forState:(UIControlState)state
```

Parameters

image

The image to use for the specified state.

state

The state that uses the specified title. The values are described in `UIControlState` (page 174).

Discussion

In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 173) value. If the `UIControlStateNormal` value is not set, then the property defaults to a system value. Therefore, at a minimum, you should set the value for the normal state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- imageForState:` (page 131)

Declared In

`UIButton.h`

setTitle:forState:

Sets the title to use for the specified state.

```
- (void)setTitle:(NSString *)title forState:(UIControlState)state
```

Parameters

title

The title to use for the specified state.

state

The state that uses the specified title. The values are described in `UIControlState` (page 174).

Discussion

In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 173) value. If the value for `UIControlStateNormal` is not set, then the property defaults to a system value. Therefore, at a minimum, you should set the value for the normal state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- titleForState:` (page 134)

Declared In

`UIButton.h`

setTitleColor:forState:

Sets the color of the title to use for the specified state.

```
- (void)setTitleColor:(UIColor *)color forState:(UIControlState)state
```

Parameters

color

The color of the title to use for the specified state.

state

The state that uses the specified color. The values are described in `UIControlState` (page 174).

Discussion

In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 173) value. If the `UIControlStateNormal` value is not set, then the property defaults to a system value. Therefore, at a minimum, you should set the value for the normal state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- titleColorForState:` (page 134)

Declared In

`UIButton.h`

setTitleShadowColor:forState:

Sets the color of the title shadow to use for the specified state.

```
- (void)setTitleShadowColor:(UIColor *)color forState:(UIControlState)state
```

Parameters

color

The color of the title shadow to use for the specified state.

state

The state that uses the specified color. The values are described in `UIControlState` (page 174).

Discussion

In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 173) value. If the `UIControlStateNormal` value is not set, then the property defaults to a system value. Therefore, at a minimum, you should set the value for the normal state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- titleShadowColorForState:` (page 135)

Declared In

`UIButton.h`

titleColorForState:

Returns the title color used for a state.

```
- (UIColor *)titleColorForState:(UIControlState)state
```

Parameters

state

The state that uses the title color. Possible values are described in `UIControlState` (page 174).

Return Value

The color of the title for the specified state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setTitleColor:forState:` (page 133)

Declared In

`UIButton.h`

titleForState:

Returns the title used for a state.

```
- (NSString *)titleForState:(UIControlState)state
```

Parameters

state

The state that uses the title. Possible values are described in `UIControlState` (page 174).

Return Value

The title for the specified state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setTitle:forState:` (page 133)

Declared In

`UIButton.h`

titleRectForContentRect:

Returns the rectangle where the receiver draws its title.

```
- (CGRect)titleRectForContentRect:(CGRect)contentRect
```

Parameters

contentRect

The content rectangle for the receiver.

Return Value

The rectangle where the receiver draws its title.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- contentRectForBounds:` (page 130)

`- imageRectForContentRect:` (page 131)

Declared In

`UIButton.h`

titleShadowColorForState:

Returns the shadow color of the title used for a state.

```
- (UIColor *)titleShadowColorForState:(UIControlState)state
```

Parameters

state

The state that uses the title shadow color. Possible values are described in `UIControlState` (page 174).

Return Value

The color of the title's shadow for the specified state.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setTitleShadowColor:forState:` (page 134)

Declared In
`UIButton.h`

Constants

UIButtonType

Specifies the style of a button.

```
typedef enum {
    UIButtonTypeCustom = 0,
    UIButtonTypeRoundedRect,
    UIButtonTypeDetailDisclosure,
    UIButtonTypeInfoLight,
    UIButtonTypeInfoDark,
    UIButtonTypeContactAdd,
} UIButtonType;
```

Constants

`UIButtonTypeCustom`
No button style.
Available in iPhone OS 2.0 and later.
Declared in `UIButton.h`

`UIButtonTypeRoundedRect`
A rounded-rectangle style button.
Available in iPhone OS 2.0 and later.
Declared in `UIButton.h`

`UIButtonTypeDetailDisclosure`
A detail disclosure button.
Available in iPhone OS 2.0 and later.
Declared in `UIButton.h`

`UIButtonTypeInfoLight`
An information button that has a light background.
Available in iPhone OS 2.0 and later.
Declared in `UIButton.h`

`UIButtonTypeInfoDark`
An information button that has a dark background.
Available in iPhone OS 2.0 and later.
Declared in `UIButton.h`

`UIButtonTypeContactAdd`

A contact add button.

Available in iPhone OS 2.0 and later.

Declared in `UIButton.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIButton.h`

UIColor Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIColor.h UIInterface.h

Overview

A `UIColor` object represents color and sometimes opacity (alpha value). You can use `UIColor` objects to store color data, and during drawing you can use them to set the current fill and stroke colors.

Many methods in UIKit require you to specify color data using a `UIColor` object, and for general color needs it should be your main way of specifying colors. The color spaces used by this object are optimized for use on iPhone OS–based devices and are therefore appropriate for most drawing needs. If you prefer to use Core Graphics colors and color spaces instead, however, you may do so.

Most developers should have no need to subclass `UIColor`. The only time doing so might be necessary is if you require support for additional colorspaces or color models.

Tasks

Creating a UIColor Object from Component Values

`+ colorWithWhite:alpha:` (page 145)
Creates and returns a color object using the specified opacity and grayscale values.

+ colorWithHue:saturation:brightness:alpha: (page 144)
Creates and returns a color object using the specified opacity and HSB color space component values.

+ colorWithRed:green:blue:alpha: (page 145)
Creates and returns a color object using the specified opacity and RGB component values.

+ colorWithCGColor: (page 143)
Creates and returns a color object using the specified Quartz color reference.

+ colorWithPatternImage: (page 144)
Creates and returns a color object using the specified image.

- colorWithAlphaComponent: (page 150)
Creates and returns a color object that has the same color space and component values as the receiver, but has the specified alpha component.

Initializing a UIColor Object

- initWithWhite:alpha: (page 153)
Initializes and returns a color object using the specified opacity and grayscale values.

- initWithHue:saturation:brightness:alpha: (page 151)
Initializes and returns a color object using the specified opacity and HSB color space component values.

- initWithRed:green:blue:alpha: (page 152)
Initializes and returns a color object using the specified opacity and RGB component values.

- initWithCGColor: (page 151)
Initializes and returns a color object using the specified Quartz color reference.

- initWithPatternImage: (page 152)
Initializes and returns a color object using the specified Quartz color reference.

Creating a UIColor with Preset Component Values

+ blackColor (page 142)
Returns a color object whose grayscale value is 0.0 and whose alpha value is 1.0.

+ darkGrayColor (page 146)
Returns a color object whose grayscale value is 1/3 and whose alpha value is 1.0.

+ lightGrayColor (page 148)
Returns a color object whose grayscale value is 2/3 and whose alpha value is 1.0.

+ whiteColor (page 150)
Returns a color object whose grayscale value is 1.0 and whose alpha value is 1.0.

+ grayColor (page 147)
Returns a color object whose grayscale value is 0.5 and whose alpha value is 1.0.

+ redColor (page 149)
Returns a color object whose RGB values are 1.0, 0.0, and 0.0 and whose alpha value is 1.0.

+ greenColor (page 147)
Returns a color object whose RGB values are 0.0, 1.0, and 0.0 and whose alpha value is 1.0.

+ blueColor (page 142)
Returns a color object whose RGB values are 0.0, 0.0, and 1.0 and whose alpha value is 1.0.

+ cyanColor (page 146)
Returns a color object whose RGB values are 0.0, 1.0, and 1.0 and whose alpha value is 1.0.

+ yellowColor (page 150)
Returns a color object whose RGB values are 1.0, 1.0, and 0.0 and whose alpha value is 1.0.

+ magentaColor (page 148)
Returns a color object whose RGB values are 1.0, 0.0, and 1.0 and whose alpha value is 1.0.

+ orangeColor (page 149)
Returns a color object whose RGB values are 1.0, 0.5, and 0.0 and whose alpha value is 1.0.

+ purpleColor (page 149)
Returns a color object whose RGB values are 0.5, 0.0, and 0.5 and whose alpha value is 1.0.

+ brownColor (page 143)
Returns a color object whose RGB values are 0.6, 0.4, and 0.2 and whose alpha value is 1.0.

+ clearColor (page 143)
Returns a color object whose grayscale and alpha values are both 0.0.

System Colors

+ lightTextColor (page 148)
Returns the system color used for displaying text on a dark background.

+ darkTextColor (page 147)
Returns the system color used for displaying text on a light background.

+ groupTableViewBackgroundColor (page 147)
Returns the system color used for the background of a grouped table.

+ viewFlipsideBackgroundColor (page 149)
Returns the system color used for the back side of a view while it is being flipped.

Retrieving Color Information

CGColor (page 142) *property*
The Quartz color reference that corresponds to the receiver's color. (read-only)

Drawing Operations

- set (page 154)
Sets the color of subsequent stroke and fill operations to the color that the receiver represents.

- setFill (page 154)
Sets the color of subsequent fill operations to the color that the receiver represents.

- setStroke (page 154)
Sets the color of subsequent stroke operations to the color that the receiver represents.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

CGColor

The Quartz color reference that corresponds to the receiver's color. (read-only)

```
@property(nonatomic,readonly) CGColorRef CGColor
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

Class Methods

blackColor

Returns a color object whose grayscale value is 0.0 and whose alpha value is 1.0.

```
+ (UIColor *)blackColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

blueColor

Returns a color object whose RGB values are 0.0, 0.0, and 1.0 and whose alpha value is 1.0.

```
+ (UIColor *)blueColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

brownColor

Returns a color object whose RGB values are 0.6, 0.4, and 0.2 and whose alpha value is 1.0.

```
+ (UIColor *)brownColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

clearColor

Returns a color object whose grayscale and alpha values are both 0.0.

```
+ (UIColor *)clearColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

colorWithCGColor:

Creates and returns a color object using the specified Quartz color reference.

```
+ (UIColor *)colorWithCGColor:(CGColorRef)cgColor
```

Parameters
cgColor
A reference to a Quartz color.

Return Value
The color object. The color information represented by this object is in the native colorspace of the specified Quartz color.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithCGColor:` (page 151)

Declared In
`UIColor.h`

colorWithHue:saturation:brightness:alpha:

Creates and returns a color object using the specified opacity and HSB color space component values.

```
+ (UIColor *)colorWithHue:(CGFloat)hue saturation:(CGFloat)saturation
    brightness:(CGFloat)brightness alpha:(CGFloat)alpha
```

Parameters

hue

The hue component of the color object in the HSB color space, specified as a value from 0.0 to 1.0.

saturation

The saturation component of the color object in the HSB color space, specified as a value from 0.0 to 1.0.

brightness

The brightness (or value) component of the color object in the HSB color space, specified as a value from 0.0 to 1.0.

alpha

The opacity value of the color object, specified as a value from 0.0 to 1.0.

Return Value

The color object. The color information represented by this object is in the device RGB colorspace.

Discussion

Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithHue:saturation:brightness:alpha: (page 151)

Declared In

UIColor.h

colorWithPatternImage:

Creates and returns a color object using the specified image.

```
+ (UIColor *)colorWithPatternImage:(UIImage *)image
```

Parameters

image

The image to use when creating the pattern color.

Return Value

The pattern color.

Discussion

You can use pattern colors to set the fill or stroke color just as you would a solid color. During drawing, the image in the pattern color is tiled as necessary to cover the given area.

By default, the phase of the returned color is 0, which causes the top-left corner of the image to be aligned with the drawing origin. To change the phase, make the color the current color and then use the `CGContextSetPatternPhase` function to change the phase.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithPatternImage:` (page 152)

Declared In
`UIColor.h`

colorWithRed:green:blue:alpha:

Creates and returns a color object using the specified opacity and RGB component values.

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green blue:(CGFloat)blue
    alpha:(CGFloat)alpha
```

Parameters

red
The red component of the color object, specified as a value from 0.0 to 1.0.

green
The green component of the color object, specified as a value from 0.0 to 1.0.

blue
The blue component of the color object, specified as a value from 0.0 to 1.0.

alpha
The opacity value of the color object, specified as a value from 0.0 to 1.0.

Return Value
The color object. The color information represented by this object is in the device RGB colorspace.

Discussion
Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithRed:green:blue:alpha:` (page 152)

Declared In
`UIColor.h`

colorWithWhite:alpha:

Creates and returns a color object using the specified opacity and grayscale values.

```
+ (UIColor *)colorWithWhite:(CGFloat)white alpha:(CGFloat)alpha
```

Parameters

white

The grayscale value of the color object, specified as a value from 0.0 to 1.0.

alpha

The opacity value of the color object, specified as a value from 0.0 to 1.0.

Return Value

The color object. The color information represented by this object is in the device gray colorspace.

Discussion

Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithWhite:alpha:` (page 153)

Declared In

`UIColor.h`

cyanColor

Returns a color object whose RGB values are 0.0, 1.0, and 1.0 and whose alpha value is 1.0.

```
+ (UIColor *)cyanColor
```

Return Value

The `UIColor` object.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIColor.h`

darkGrayColor

Returns a color object whose grayscale value is 1/3 and whose alpha value is 1.0.

```
+ (UIColor *)darkGrayColor
```

Return Value

The `UIColor` object.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIColor.h`

darkTextColor

Returns the system color used for displaying text on a light background.

```
+ (UIColor *)darkTextColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIInterface.h`

grayColor

Returns a color object whose grayscale value is 0.5 and whose alpha value is 1.0.

```
+ (UIColor *)grayColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

greenColor

Returns a color object whose RGB values are 0.0, 1.0, and 0.0 and whose alpha value is 1.0.

```
+ (UIColor *)greenColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

groupTableViewBackgroundColor

Returns the system color used for the background of a grouped table.

```
+ (UIColor *)groupTableViewBackgroundColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIInterface.h`

lightGrayColor

Returns a color object whose grayscale value is 2/3 and whose alpha value is 1.0.

```
+ (UIColor *)lightGrayColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

lightTextColor

Returns the system color used for displaying text on a dark background.

```
+ (UIColor *)lightTextColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIInterface.h`

magentaColor

Returns a color object whose RGB values are 1.0, 0.0, and 1.0 and whose alpha value is 1.0.

```
+ (UIColor *)magentaColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

orangeColor

Returns a color object whose RGB values are 1.0, 0.5, and 0.0 and whose alpha value is 1.0.

```
+ (UIColor *)orangeColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

purpleColor

Returns a color object whose RGB values are 0.5, 0.0, and 0.5 and whose alpha value is 1.0.

```
+ (UIColor *)purpleColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

redColor

Returns a color object whose RGB values are 1.0, 0.0, and 0.0 and whose alpha value is 1.0.

```
+ (UIColor *)redColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIColor.h`

viewFlipsideBackgroundColor

Returns the system color used for the back side of a view while it is being flipped.

```
+ (UIColor *)viewFlipsideBackgroundColor
```

Return Value
The `UIColor` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIInterface.h

whiteColor

Returns a color object whose grayscale value is 1.0 and whose alpha value is 1.0.

```
+ (UIColor *)whiteColor
```

Return Value
The UIColor object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIColor.h

yellowColor

Returns a color object whose RGB values are 1.0, 1.0, and 0.0 and whose alpha value is 1.0.

```
+ (UIColor *)yellowColor
```

Return Value
The UIColor object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIColor.h

Instance Methods

colorWithAlphaComponent:

Creates and returns a color object that has the same color space and component values as the receiver, but has the specified alpha component.

```
- (UIColor *)colorWithAlphaComponent:(CGFloat)alpha
```

Parameters
alpha
The opacity value of the new UIColor object.

Return Value
The new UIColor object.

Discussion
A subclass with explicit opacity components should override this method to return a color with the specified alpha.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIColor.h

initWithCGColor:
Initializes and returns a color object using the specified Quartz color reference.

```
- (UIColor *)initWithCGColor:(CGColorRef)cgColor
```

Parameters
cgColor
A reference to a Quartz color.

Return Value
An initialized color object. The color information represented by this object is in the native colorspace of the specified Quartz color.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ colorWithCGColor: (page 143)

Declared In
UIColor.h

initWithHue:saturation:brightness:alpha:
Initializes and returns a color object using the specified opacity and HSB color space component values.

```
- (UIColor *)initWithHue:(CGFloat)hue saturation:(CGFloat)saturation
    brightness:(CGFloat)brightness alpha:(CGFloat)alpha
```

Parameters
hue
The hue component of the color object in the HSB color space, specified as a value from 0.0 to 1.0.

saturation
The saturation component of the color object in the HSB color space, specified as a value from 0.0 to 1.0.

brightness
The brightness (or value) component of the color object in the HSB color space, specified as a value from 0.0 to 1.0.

alpha
The opacity value of the color object, specified as a value from 0.0 to 1.0.

Return Value
An initialized color object. The color information represented by this object is in the device RGB colorspace.

Discussion
Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ colorWithHue:saturation:brightness:alpha:` (page 144)

Declared In
`UIColor.h`

initWithPatternImage:

Initializes and returns a color object using the specified Quartz color reference.

```
- (UIColor *)initWithPatternImage:(UIImage *)image
```

Parameters
image
The image to use when creating the pattern color.

Return Value
The pattern color.

Discussion
You can use pattern colors to set the fill or stroke color just as you would a solid color. During drawing, the image in the pattern color is tiled as necessary to cover the given area.

By default, the phase of the returned color is 0, which causes the top-left corner of the image to be aligned with the drawing origin. To change the phase, make the color the current color and then use the `CGContextSetPatternPhase` function to change the phase.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ colorWithPatternImage:` (page 144)

Declared In
`UIColor.h`

initWithRed:green:blue:alpha:

Initializes and returns a color object using the specified opacity and RGB component values.

```
- (UIColor *)initWithRed:(CGFloat)red green:(CGFloat)green blue:(CGFloat)blue
    alpha:(CGFloat)alpha
```

Parameters

red

The red component of the color object, specified as a value from 0.0 to 1.0.

green

The green component of the color object, specified as a value from 0.0 to 1.0.

blue

The blue component of the color object, specified as a value from 0.0 to 1.0.

alpha

The opacity value of the color object, specified as a value from 0.0 to 1.0.

Return Value

An initialized color object. The color information represented by this object is in the device RGB colorspace.

Discussion

Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ colorWithRed:green:blue:alpha:` (page 145)

Declared In

`UIColor.h`

initWithWhite:alpha:

Initializes and returns a color object using the specified opacity and grayscale values.

```
- (UIColor *)initWithWhite:(CGFloat)white alpha:(CGFloat)alpha
```

Parameters

white

The grayscale value of the color object, specified as a value from 0.0 to 1.0.

alpha

The opacity value of the color object, specified as a value from 0.0 to 1.0.

Return Value

An initialized color object. The color information represented by this object is in the device gray colorspace.

Discussion

Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ colorWithWhite:alpha:` (page 145)

Declared In

`UIColor.h`

set

Sets the color of subsequent stroke and fill operations to the color that the receiver represents.

```
- (void)set
```

Discussion

If you subclass `UIColor`, you must implement this method in your subclass. Your custom implementation should modify both the stroke and fill color in the current graphics context by setting them both to the color represented by the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setFill` (page 154)
- `setStroke` (page 154)

Declared In

`UIColor.h`

setFill

Sets the color of subsequent fill operations to the color that the receiver represents.

```
- (void)setFill
```

Discussion

If you subclass `UIColor`, you must implement this method in your subclass. Your custom implementation should modify the fill color in the current graphics context by setting it to the color represented by the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `set` (page 154)

Declared In

`UIColor.h`

setStroke

Sets the color of subsequent stroke operations to the color that the receiver represents.

```
- (void)setStroke
```

Discussion

If you subclass `UIColor`, you must implement this method in your subclass. Your custom implementation should modify the stroke color in the current graphics context by setting it to the color represented by the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also
- set (page 154)

Declared In
UIColor.h

UIControl Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIControl.h

Overview

`UIControl` is the base class for controls: objects such as buttons and sliders that are used to convey user intent to the application. You cannot use `UIControl` directly to instantiate controls. It instead defines the common interface and behavioral structure for all subclasses of it.

The main role of `UIControl` is to define an interface and base implementation for preparing action messages and initially dispatching them to their targets when specified events occur. (See "The Target-Action Mechanism" (page 157) for an overview.) It also includes methods for getting and setting control state (for example, for determining whether a control is enabled or highlighted) and it defines methods for tracking touches within a control (the latter group of methods are for overriding by subclasses).

The Target-Action Mechanism

The design of the target-action mechanism in the UIKit framework is based on the Multi-Touch event model. In iPhone OS the user's fingers (or touches) convey intent (instead of mouse clicks and drags), and there can be multiple touches at any moment on a control going in different directions.

Note: For more information on the Multi-Touch event model, see Event Handling in *iPhone OS Programming Guide*.

The `UIControl.h` header file declares a large number of control events as constants for a bit mask described in "`Control Events`" (page 169). Some control events specify the behavior of touches in and around the control—various permutations of actions such a finger touching down in a control, dragging into and away from a control, and lifting up from a control. Other control events specify editing phases initiated when a finger touches down in a text field. And yet another control event, `UIControlEventValueChanged` (page 170), is for controls such as sliders, where a value continuously changes based on the manipulation of the control. For any particular action message, you can specify one or more control events as the trigger for sending that message. For example, you could request a certain action message be sent to a certain target when a finger touches down in a control or is dragged into it (`UIControlEventTouchDown` (page 169) | `UIControlEventTouchDragEnter` (page 170)).

You prepare a control for sending an action message by calling `addTarget:action:forControlEvents:` (page 164) for each target-action pair you want to specify. This method builds an internal dispatch table associating each target-action pair with a control event. When a user touches the control in a way that corresponds to one or more specified events, `UIControl` sends itself `sendActionsForControlEvents:` (page 168). This results in `UIControl` sending the action to `UIApplication` in a `sendAction:to:from:forEvent:` (page 98) message. `UIApplication` is the centralized dispatch point for action messages; if a `nil` target is specified for an action message, the application sends the action to the first responder where it travels up the responder chain until it finds an object willing to handle the action message—that is, object that implements a method corresponding to the action selector. (Event Handling gives an overview of the first responder and the responder chain.)

UIKit allows three different forms of action selector:

```
- (void)action
- (void)action:(id)sender
- (void)action:(id)sender forEvent:(UIEvent *)event
```

The `sendAction:to:fromSender:forEvent:` method of `UIApplication` pushes two parameters when calling the target. These last two parameters are optional for the application because it's up to the caller (usually a `UIControl` object) to remove any parameters it added.

Subclassing Notes

You may want to extend a `UIControl` subclass for two basic reasons:

- To observe or modify the dispatch of action messages to targets for particular events

  To do this, override `sendAction:to:forEvent:` (page 167), evaluate the passed-in selector, target object, or "`Note`" (page 158) bit mask and proceed as required.

- To provide custom tracking behavior (for example, to change the highlight appearance)

  To do this, override one or all of the following methods: `beginTrackingWithTouch:withEvent:` (page 165), `continueTrackingWithTouch:withEvent:` (page 166), `endTrackingWithTouch:withEvent:` (page 166).

Tasks

Preparing and Sending Action Messages

- `sendAction:to:forEvent:` (page 167)
  In response to a given event, forwards an action message to the application object for dispatching to a target.
- `sendActionsForControlEvents:` (page 168)
  Sends action messages for the given control events.
- `addTarget:action:forControlEvents:` (page 164)
  Adds a target and action for a particular event (or events) to an internal dispatch table.
- `removeTarget:action:forControlEvents:` (page 167)
  Removes a target and action for a particular event (or events) from an internal dispatch table.
- `actionsForTarget:forControlEvent:` (page 163)
  Returns the actions that are associated with a target and a particular control event.
- `allTargets` (page 165)
  Returns all target objects associated with the receiver.
- `allControlEvents` (page 165)
  Returns all control events associated with the receiver.

Setting and Getting Control Attributes

`state` (page 162) *property*
A Boolean value that indicates the state of the receiver. (read-only)

`enabled` (page 161) *property*
A Boolean value that determines whether the receiver is enabled.

`selected` (page 162) *property*
A Boolean value that determines the receiver's selected state.

`highlighted` (page 161) *property*
A Boolean value that determines whether the receiver is highlighted.

`contentVerticalAlignment` (page 160) *property*
The vertical alignment of content (text or image) within the receiver.

`contentHorizontalAlignment` (page 160) *property*
The horizontal alignment of content (text or image) within the receiver.

Tracking Touches and Redrawing Controls

- `beginTrackingWithTouch:withEvent:` (page 165)
  Sent the control when a touch related to the given event enters its bounds.
- `continueTrackingWithTouch:withEvent:` (page 166)
  Sent continuously to the control as it tracks a touch related to the given event within its bounds.

- `endTrackingWithTouch:withEvent:` (page 166)
  Sent to the control when the last touch for the given event leaves its bounds, telling it to stop tracking.
- `cancelTrackingWithEvent:` (page 166)
  Tells the control to cancel tracking related to the given event.

`tracking` (page 163) *property*
A Boolean value that indicates whether the receiver is currently tracking touches related to an event. (read-only)

`touchInside` (page 163) *property*
A Boolean value that indicates whether a touch is inside the bounds of the receiver. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

contentHorizontalAlignment

The horizontal alignment of content (text or image) within the receiver.

```
@property(nonatomic) UIControlContentHorizontalAlignment contentHorizontalAlignment
```

Parameters

contentAlignment
A constant that specifies the alignment of text or image within the receiver. See "Horizontal Content Alignment" (page 172) for descriptions of valid constants.

Discussion

The value of this property is a `UIControlContentAlignment` constant that specifies the alignment of text or image within the receiver. The default is `UIControlContentHorizontalAlignmentLeft` (page 172).

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property contentVerticalAlignment` (page 160)

Declared In

`UIControl.h`

contentVerticalAlignment

The vertical alignment of content (text or image) within the receiver.

```
@property(nonatomic) UIControlContentVerticalAlignment contentVerticalAlignment
```

Parameters

contentAlignment

A constant that specifies the alignment of text or image within the receiver. See "Vertical Content Alignment" (page 171) for descriptions of valid constants.

Discussion

This value of this property is a `UIControlContentAlignment` constant that specifies the alignment of text or image within the receiver. The default is `UIControlContentVerticalAlignmentTop` (page 171).

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property contentHorizontalAlignment` (page 160)

Declared In

`UIControl.h`

enabled

A Boolean value that determines whether the receiver is enabled.

```
@property(nonatomic, getter=isEnabled) BOOL enabled
```

Discussion

Specify `YES` to make the control enabled; otherwise, specify `NO` to make it disabled. The default value is `YES`. If the enabled state is `NO`, the control ignores touch events and subclasses may draw differently.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property state` (page 162)

Declared In

`UIControl.h`

highlighted

A Boolean value that determines whether the receiver is highlighted.

```
@property(nonatomic, getter=isHighlighted) BOOL highlighted
```

Discussion

Specify `YES` if the control is highlighted; otherwise `NO`. By default, a control is not highlighted. `UIControl` automatically sets and clears this state automatically when a touch enters and exits during tracking and and when there is a touch up.

Availability

Available in iPhone OS 2.0 and later.

See Also
`@property state` (page 162)

Declared In
`UIControl.h`

selected

A Boolean value that determines the receiver's selected state.

```
@property(nonatomic, getter=isSelected) BOOL selected
```

Discussion
Specify `YES` if the control is selected; otherwise `NO`. The default is `NO`. For many controls, this state has no effect on behavior or appearance. But other subclasses (for example, UISwitchControl) or the application object might read or set this control state.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property state` (page 162)

Declared In
`UIControl.h`

state

A Boolean value that indicates the state of the receiver. (read-only)

```
@property(nonatomic, readonly) UIControlState state
```

Discussion
One or more `UIControlState` (page 174) bit-mask constants that specify the state of the `UIControl` object; for information on these constants, see "Control State" (page 173). Note that the control can be in more than one state, for example, both disabled and selected (`UIControlStateDisabled` (page 173) | `UIControlStateSelected` (page 174)).This attribute is read only—there is no corresponding setter method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property enabled` (page 161)
`@property selected` (page 162)
`@property highlighted` (page 161)

Declared In
`UIControl.h`

touchInside

A Boolean value that indicates whether a touch is inside the bounds of the receiver. (read-only)

```
@property(nonatomic, readonly, getter=isTouchInside) BOOL touchInside
```

Return Value

`YES` if a touch is inside the receiver's bounds; otherwise `NO`.

Discussion

The value is `YES` if a touch is inside the receiver's bounds; otherwise the value is `NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIControl.h`

tracking

A Boolean value that indicates whether the receiver is currently tracking touches related to an event. (read-only)

```
@property(nonatomic, readonly, getter=isTracking) BOOL tracking
```

Discussion

The value is `YES` if the receiver is tracking touches; otherwise`NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIControl.h`

Instance Methods

actionsForTarget:forControlEvent:

Returns the actions that are associated with a target and a particular control event.

```
- (NSArray *)actionsForTarget:(id)target
    forControlEvent:(UIControlEvents)controlEvent
```

Parameters

target

The target object—that is, the object to which an action message is sent. If this is `nil`, all actions associated with the control event are returned.

controlEvent

A single constant of type "`Note`" (page 158) that specifies a particular user action on the control; for a list of these constants, see "Control Events" (page 169).

Return Value
An array of selector names as `NSString` objects or `nil` if there are no action selectors associated with the control event.

Discussion
Pass in a selector name to the `NSSelectorFromString` function to obtain the selector (`SEL`) value.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sendAction:to:forEvent:` (page 167)
- `sendActionsForControlEvents:` (page 168)
- `addTarget:action:forControlEvents:` (page 164)

Declared In
`UIControl.h`

addTarget:action:forControlEvents:
Adds a target and action for a particular event (or events) to an internal dispatch table.

```
- (void)addTarget:(id)target action:(SEL)action
    forControlEvents:(UIControlEvents)controlEvents
```

Parameters
target

The target object—that is, the object to which the action message is sent. If this is `nil`, the responder chain is searched for an object willing to respond to the action message.

action

A selector identifying an action message. It cannot be `NULL`.

controlEvents

A bitmask specifying the control events for which the action message is sent. See "Control Events" (page 169) for bitmask constants.

Discussion
You may call this method multiple times, and you may specify multiple target-action pairs for a particular event. The action message may optionally include the sender and the event as parameters, in that order.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeTarget:action:forControlEvents:` (page 167)
- `actionsForTarget:forControlEvent:` (page 163)

Declared In
`UIControl.h`

allControlEvents

Returns all control events associated with the receiver.

```
- (UIControlEvents)allControlEvents
```

Return Value

One or more "Note" (page 158) constants that specify the current control events associated with the receiver; for a list of these constants, see "Control Events" (page 169)list of all events that have at least one action.

Availability

Available in iPhone OS 2.0 and later.

See Also

- allTargets (page 165)

Declared In

UIControl.h

allTargets

Returns all target objects associated with the receiver.

```
- (NSSet *)allTargets
```

Return Value

A set of all targets—that is, the objects to which action messages are sent—for the receiver. The set may include NSNull to indicate at least one nil target (meaning, the responder chain is searched for a target).

Availability

Available in iPhone OS 2.0 and later.

See Also

- allControlEvents (page 165)

Declared In

UIControl.h

beginTrackingWithTouch:withEvent:

Sent the control when a touch related to the given event enters its bounds.

```
- (BOOL)beginTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event
```

Parameters

touch

A UITouch object that represents a touch on the receiving control during tracking.

event

An event object encapsulating the information specific to the user event.

Return Value
YES if the receiver is set to respond continuously or set to respond when a touch is dragged; otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIControl.h

cancelTrackingWithEvent:

Tells the control to cancel tracking related to the given event.

```
- (void)cancelTrackingWithEvent:(UIEvent *)event
```

Parameters

event
An event object encapsulating the information specific to the user event. This parameter might be nil, indicating that the cancelation was caused by something other than an event, such as the view being removed from the window.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIControl.h

continueTrackingWithTouch:withEvent:

Sent continuously to the control as it tracks a touch related to the given event within its bounds.

```
- (BOOL)continueTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event
```

Parameters

touch
A UITouch object that represents a touch on the receiving control during tracking.

event
An event object encapsulating the information specific to the user event

Return Value
YES if mouse tracking should continue; otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIControl.h

endTrackingWithTouch:withEvent:

Sent to the control when the last touch for the given event leaves its bounds, telling it to stop tracking.

```
- (void)endTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event
```

Parameters

touches

A `UITouch` object that represents a touch on the receiving control during tracking.

event

An event object encapsulating the information specific to the user event.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIControl.h`

removeTarget:action:forControlEvents:

Removes a target and action for a particular event (or events) from an internal dispatch table.

```
- (void)removeTarget:(id)target action:(SEL)action
    forControlEvents:(UIControlEvents)controlEvents
```

Parameters

target

The target object—that is, the object to which the action message is sent.

action

A selector identifying an action message. Pass `NULL` to remove all action messages paired with *target*.

controlEvents

A bitmask specifying the control events associated with *target* and *action*. See "Control Events" (page 169) for bitmask constants.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- addTarget:action:forControlEvents:` (page 164)

Declared In

`UIControl.h`

sendAction:to:forEvent:

In response to a given event, forwards an action message to the application object for dispatching to a target.

```
- (void)sendAction:(SEL)action to:(id)target forEvent:(UIEvent *)event
```

Parameters

action

A selector identifying an action message. It cannot be `NULL`.

target

The target object—that is, the object to which the action message is sent. If this is `nil`, the receiver traverses the responder chain and sends the action message to the first object willing to respond to it.

event

An object representing the event (typically in a `UIControl` object) that originated the action message. The event can be `nil` if the action is invoked directly instead of being caused by an event. For example, a value-changed message might be sent for programmatic reasons rather than as a result of the user touching the control.

Discussion

`UIControl` implements this method to forward an action message to the singleton `UIApplication` object (in its `sendAction:to:fromSender:forEvent:` method) for dispatching it to the target or, if there is no specified target, to the first object in the responder chain that is willing to handle it. Subclasses may override this method to observe or modify action-forwarding behavior. The implementation of `sendActionsForControlEvents:` (page 168) might call this method repeatedly, once for each specified control event.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addTarget:action:forControlEvents:` (page 164)
- `sendActionsForControlEvents:` (page 168)

Declared In

`UIControl.h`

sendActionsForControlEvents:

Sends action messages for the given control events.

```
- (void)sendActionsForControlEvents:(UIControlEvents)controlEvents
```

Parameters

controlEvents

A bitmask whose set flags specify the control events for which action messages are sent. See "Control Events" (page 169) for bitmask constants.

Discussion

`UIControl` implements this method to send all action messages associated with *controlEvents*, repeatedly invoking `sendAction:to:forEvent:` (page 167) in the process. The list of targets and actions it looks up is constructed from prior invocations of `addTarget:action:forControlEvents:` (page 164).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `sendAction:to:forEvent:` (page 167)
- `addTarget:action:forControlEvents:` (page 164)

Declared In

`UIControl.h`

Constants

Control Events

Kinds of events possible for control objects.

```
enum {
    UIControlEventTouchDown           = 1 <<  0,
    UIControlEventTouchDownRepeat     = 1 <<  1,
    UIControlEventTouchDragInside     = 1 <<  2,
    UIControlEventTouchDragOutside    = 1 <<  3,
    UIControlEventTouchDragEnter      = 1 <<  4,
    UIControlEventTouchDragExit       = 1 <<  5,
    UIControlEventTouchUpInside       = 1 <<  6,
    UIControlEventTouchUpOutside      = 1 <<  7,
    UIControlEventTouchCancel         = 1 <<  8,

    UIControlEventValueChanged        = 1 << 12,

    UIControlEventEditingDidBegin     = 1 << 16,
    UIControlEventEditingChanged      = 1 << 17,
    UIControlEventEditingDidEnd       = 1 << 18,
    UIControlEventEditingDidEndOnExit = 1 << 19,

    UIControlEventAllTouchEvents      = 0x00000FFF,
    UIControlEventAllEditingEvents    = 0x000F0000,
    UIControlEventApplicationReserved = 0x0F000000,
    UIControlEventSystemReserved      = 0xF0000000,
    UIControlEventAllEvents           = 0xFFFFFFFF
};
```

Constants

`UIControlEventTouchDown`

A touch-down event in the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchDownRepeat`

A repeated touch-down event in the control; for this event the value of the UITouch `tapCount` method is greater than one.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchDragInside`

An event where a finger is dragged inside the bounds of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchDragOutside`

An event where a finger is dragged just outside the bounds of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchDragEnter`
An event where a finger is dragged into the bounds of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchDragExit`
An event where a finger is dragged from within a control to outside its bounds.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchUpInside`
A touch-up event in the control where the finger is inside the bounds of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchUpOutside`
A touch-up event in the control where the finger is outside the bounds of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventTouchCancel`
A system event canceling the current touches for the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventValueChanged`
A touch dragging or otherwise manipulating a control, causing it to emit a series of different values.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventEditingDidBegin`
A touch initiating an editing session in a `UITextField` object by entering its bounds.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventEditingChanged`
A touch making an editing change in a `UITextField` objet.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventEditingDidEnd`
A touch ending an editing session in a `UITextField` object by leaving its bounds.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventEditingDidEndOnExit`
A touch ending an editing session in a `UITextField` object.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventAllTouchEvents`
All touch events.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventAllEditingEvents`
All editing touches for `UITextField` objects.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventApplicationReserved`
A range of control-event values available for application use.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventSystemReserved`
A range of control-event values reserved for internal framework use.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlEventAllEvents`
All events, including system events.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

Declared In
`UIControl.h`

Vertical Content Alignment

The vertical alignment of content (text and images) within a control.

```
typedef enum {
   UIControlContentVerticalAlignmentCenter  = 0,
   UIControlContentVerticalAlignmentTop     = 1,
   UIControlContentVerticalAlignmentBottom  = 2,
   UIControlContentVerticalAlignmentFill    = 3,
} UIControlContentVerticalAlignment;
```

Constants
`UIControlContentVerticalAlignmentCenter`
Aligns the content vertically in the center of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlContentVerticalAlignmentTop`
Aligns the content vertically at the top in the control (the default).

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlContentVerticalAlignmentBottom`
Aligns the content vertically at the bottom in the control

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlContentVerticalAlignmentFill`
Aligns the content vertically to fill the content rectangle; images may be stretched.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

Discussion
You use these constants as the value of the `contentVerticalAlignment` (page 160) property.

Declared In
`UITouch.h`

Horizontal Content Alignment

The horizontal alignment of content (text and images) within a control.

```
typedef enum {
   UIControlContentHorizontalAlignmentCenter = 0,
   UIControlContentHorizontalAlignmentLeft   = 1,
   UIControlContentHorizontalAlignmentRight = 2,
   UIControlContentHorizontalAlignmentFill   = 3,
} UIControlContentHorizontalAlignment;
```

Constants
`UIControlContentHorizontalAlignmentCenter`
Aligns the content horizontally in the center of the control.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlContentHorizontalAlignmentLeft`
Aligns the content horizontally from the left of the control (the default).

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlContentHorizontalAlignmentRight`
Aligns the content horizontally from the right of the control

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlContentHorizontalAlignmentFill`
Aligns the content horizontally to fill the content rectangles; text may wrap and images may be stretched.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

Discussion
You use these constants as the value of the `contentHorizontalAlignment` (page 160) property.

Declared In
`UIControl.h`

UIControlContentAlignment

The bit-mask type for the content-alignment constants.

```
typedef NSUInteger UIControlContentAlignment;
```

Discussion

This is the type for the constants listed in "Vertical Content Alignment" (page 171) and "Horizontal Content Alignment" (page 172). Use `contentVerticalAlignment` (page 160) and `contentHorizontalAlignment` (page 160) properties to retrieve the current control alignment and set it.

Declared In

`UITouch.h`

Control State

The state of a control; a control can have more than one state at a time. States are recognized differently depending on the control. For example, a UIButton instance may be configured (using the `setImage:forState:` (page 132) method) to display one image when it is in its normal state and a different image when it is highlighted.

```
enum {
    UIControlStateNormal               = 0,
    UIControlStateHighlighted          = 1 << 0,
    UIControlStateDisabled             = 1 << 1,
    UIControlStateSelected             = 1 << 2,
    UIControlStateApplicationReserved  = 0x00FF0000,
    UIControlStateReserved             = 0xFF000000
};
```

Constants

`UIControlStateNormal`

The normal, or default state of a control—that is, enabled but neither selected or highlighted.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlStateHighlighted`

Highlighted state of a control. A control enters this state when a touch enters and exits during tracking and and when there is a touch up. You can retrieve and set this value through the `highlighted` (page 161) property.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlStateDisabled`

Disabled state of a control. This state indicates that the control is currently disabled. You can retrieve and set this value through the `enabled` (page 161) property.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlStateSelected`
Selected state of a control. For many controls, this state has no effect on behavior or appearance. But other subclasses (for example, `UISwitchControl`). You can retrieve and set this value through the `selected` (page 162) property.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

`UIControlStateApplicationReserved`
Additional control-state flags available for application use.

`UIControlStateReserved`
Control-state flags reserved for internal framework use.

Available in iPhone OS 2.0 and later.

Declared in `UIControl.h`

Declared In
`UITouch.h`

UIControlState

The bit-mask type for control-state constants.

```
typedef NSUInteger UIControlState;
```

Discussion
The constants are listed in "Control State" (page 173). Use the `state` (page 162) property to retrieve the current state bits set for a control.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIControl.h`

UIDatePicker Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIDatePicker.h

Overview

The `UIDatePicker` class implements an object that uses multiple rotating wheels to allow users to select dates and times. iPhone examples of a date picker are the Timer and Alarm (Set Alarm) panes of the Clock application. You may also use a date picker as a countdown timer.

When properly configured, a `UIDatePicker` object sends an action message when a user finishes rotating one of the wheels to change the date or time; the associated control event is `UIControlEventValueChanged` (page 170). A `UIDatePicker` object presents the countdown timer but does not implement it; the application must set up an `NSTimer` object and update the seconds as they're counted down.

`UIDatePicker` does not inherit from `UIPickerView`, but it manages a custom picker-view object as a subview.

Tasks

Managing the Date and Calendar

`calendar` (page 176) *property*
The calendar to use for the date picker.

`date` (page 177) *property*
The date displayed by the date picker.

`locale` (page 178) *property*
The locale used by the date picker.

- `setDate:animated:` (page 180)
Sets the date to display in the date picker, with an option to animate the setting.

`timeZone` (page 179) *property*
The time zone reflected in the date displayed by the date picker.

Configuring the Date Picker Mode

`datePickerMode` (page 177) *property*
The mode of the date picker.

Configuring Temporal Attributes

`maximumDate` (page 178) *property*
The maximum date that a date picker can show.

`minimumDate` (page 179) *property*
The minimum date that a date picker can show.

`minuteInterval` (page 179) *property*
The interval at which the date picker should display minutes.

`countDownDuration` (page 177) *property*
The seconds from which the countdown timer counts down.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

calendar

The calendar to use for the date picker.

```
@property(nonatomic, copy) NSCalendar *calendar
```

Discussion

The default value is `nil`. which means use the user's current calendar (equivalent to calling the `NSCalendar` class method `currentCalendar`). Calendars specify the details of cultural systems used for reckoning time; they identify the beginning, length, and divisions of a year.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property locale` (page 178)
`@property timeZone` (page 179)

Declared In
UIDatePicker.h

countDownDuration

The seconds from which the countdown timer counts down.

```
@property(nonatomic) NSTimeInterval countDownDuration
```

Discussion
The NSTimeInterval value of this property indicates the seconds from which the date picker in countdown-timer mode counts down. If the mode of the date picker is not UIDatePickerModeCountDownTimer (page 181), this value is ignored. The default value is 0.0 and the maximum value is 23:59 (86,399 seconds).

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIDatePicker.h

date

The date displayed by the date picker.

```
@property(nonatomic, retain) NSDate *date
```

Discussion
The default is the date when the UIDatePicker object is created. The date is ignored in the mode UIDatePickerModeCountDownTimer (page 181); for that mode, the date picker starts at 0:00. Setting this property does not animate the date picker by spinning the wheels to the new date and time; to do that you must use the setDate:animated: (page 180) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setDate:animated: (page 180)

Declared In
UIDatePicker.h

datePickerMode

The mode of the date picker.

```
@property(nonatomic) UIDatePickerMode datePickerMode
```

Discussion
The value of this property indicates the mode of a date picker. It determines whether the date picker allows selection of a date, a time, both date and time, or a countdown time. The default mode is `UIDatePickerModeDateAndTime` (page 181). See "Date Picker Mode" (page 180) for a list of mode constants.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDatePicker.h`

locale

The locale used by the date picker.

```
@property(nonatomic, retain) NSLocale *locale
```

Discussion
The default value is `nil`. which tells the date picker to use the current locale as returned by `currentLocale` (`NSLocale`) or the locale used by the date picker's calendar. Locales encapsulate information about facets of a language or culture, such as the way dates are formatted.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property calendar` (page 176)
`@property timeZone` (page 179)

Declared In
`UIDatePicker.h`

maximumDate

The maximum date that a date picker can show.

```
@property(nonatomic, retain) NSDate *maximumDate
```

Discussion
The property is an `NSDate` object or `nil` (the default), which means no maximum date. This property, along with the `minimumDate` (page 179) property, lets you specify a valid date range. If the minimum date value is greater than the maximum date value, both properties are ignored. The minimum and maximum dates are also ignored in the coundown-timer mode (`UIDatePickerModeCountDownTimer` (page 181)).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDatePicker.h`

minimumDate

The minimum date that a date picker can show.

```
@property(nonatomic, retain) NSDate *minimumDate
```

Discussion

The property is an `NSDate` object or `nil` (the default), which means no minimum date. This property, along with the `maximumDate` (page 178) property, lets you specify a valid date range. If the minimum date value is greater than the maximum date value, both properties are ignored. The minimum and maximum dates are also ignored in the coundown-timer mode (`UIDatePickerModeCountDownTimer` (page 181)).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIDatePicker.h`

minuteInterval

The interval at which the date picker should display minutes.

```
@property(nonatomic) NSInteger minuteInterval
```

Discussion

You can use this property to set the interval displayed by the minutes wheel (for example, 15 minutes). The interval value must be evenly divided into 60; if it is not, the default value is used. The default and minimum values are 1; the maximum value is 30.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIDatePicker.h`

timeZone

The time zone reflected in the date displayed by the date picker.

```
@property(nonatomic, retain) NSTimeZone *timeZone
```

Discussion

The default value is `nil`. which tells the date picker to use the current time zone as returned by `localTimeZone` (`NSTimeZone`) or the time zone used by the date picker's calendar.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property calendar` (page 176)
`@property locale` (page 178)

Declared In
UIDatePicker.h

Instance Methods

setDate:animated:

Sets the date to display in the date picker, with an option to animate the setting.

```
- (void)setDate:(NSDate *)date animated:(BOOL)animated
```

Parameters

date

An NSDate object representing the new date to display in the date picker.

animated

YES to animate the setting of the new date, otherwise NO. The animation rotates the wheels until the new date and time is shown under the highlight rectangle.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property date (page 177)

Declared In
UIDatePicker.h

Constants

Date Picker Mode

The mode of the date picker.

```
typedef enum {
    UIDatePickerModeTime,
    UIDatePickerModeDate,
    UIDatePickerModeDateAndTime,
    UIDatePickerModeCountDownTimer
} UIDatePickerMode;
```

Constants

UIDatePickerModeTime

The date picker displays hours, minutes, and (optionally) an AM/PM designation. The exact items shown and their order depend upon the locale set. An example of this mode is [6 | 53 | PM].

Available in iPhone OS 2.0 and later.

Declared in UIDatePicker.h

`UIDatePickerModeDate`

The date picker displays months, days of the month, and years. The exact order of these items depends on the locale setting. An example of this mode is [November | 15 | 2007].

Available in iPhone OS 2.0 and later.

Declared in `UIDatePicker.h`

`UIDatePickerModeDateAndTime`

The date picker displays dates (as unified day of the week, month, and day of the month values) plus hours, minutes, and (optionally) an AM/PM designation. The exact order and format of these items depends on the locale set. An example of this mode is [Wed Nov 15 | 6 | 53 | PM].

Available in iPhone OS 2.0 and later.

Declared in `UIDatePicker.h`

`UIDatePickerModeCountDownTimer`

The date picker displays hour and minute values, for example [1 | 53]. The application must set a timer to fire at the proper interval and set the date picker as the seconds tick down.

Available in iPhone OS 2.0 and later.

Declared in `UIDatePicker.h`

Discussion

The mode determines whether dates, times, or both dates and times are displayed. You can also use it to specify the appearance of a countdown timer. You can set and retrieve the mode value through the `datePickerMode` (page 177) property.

UIDevice Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIDevice.h

Overview

The `UIDevice` class vends a singleton instance representing the current device. From this instance you can obtain information about the device, such as unique ID, assigned name, device model, and operating-system name and version.

You can also use the `UIDevice` class to detect changes in the device's physical orientation. You can get the current orientation using the `orientation` (page 186) property or receive change notifications by registering for the `UIDeviceOrientationDidChangeNotification` (page 190) notification. Before using either of these techniques to get orientation data, you must enable the delivery of the data using the `beginGeneratingDeviceOrientationNotifications` (page 188) method. When you no longer need to track the device orientation, you should similarly use the `endGeneratingDeviceOrientationNotifications` (page 188) method to disable the delivery of notifications.

Tasks

Getting the Shared Device Instance

`+ currentDevice` (page 187)
Returns an object representing the current device.

Identifying the Device and Operating System

`uniqueIdentifier` (page 187) *property*
A string unique to each device based on various hardware details. (read-only)

`name` (page 185) *property*
The name identifying the device. (read-only)

`systemName` (page 186) *property*
The name of the operating system running on the device represented by the receiver. (read-only)

`systemVersion` (page 186) *property*
The current version of the operating system. (read-only)

`model` (page 185) *property*
The model of the device. (read-only)

`localizedModel` (page 185) *property*
The model of the device as a localized string. (read-only)

Getting the Current Device Orientation

`orientation` (page 186) *property*
Returns the physical orientation of the device. (read-only)

`generatesDeviceOrientationNotifications` (page 184) *property*
A Boolean value that determines whether the receiver generates orientation notifications. (read-only)

- `beginGeneratingDeviceOrientationNotifications` (page 188)
Begins the generation of notifications of device orientation changes.

- `endGeneratingDeviceOrientationNotifications` (page 188)
Ends the generation of notifications of device orientation changes.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

generatesDeviceOrientationNotifications

A Boolean value that determines whether the receiver generates orientation notifications. (read-only)

```
@property(nonatomic, readonly, getter=isGeneratingDeviceOrientationNotifications)
    BOOL generatesDeviceOrientationNotifications
```

Discussion

If the value of this property is `YES`, the shared `UIDevice` object posts a `UIDeviceOrientationDidChangeNotification` (page 190) notification when the device changes orientation. If the value is `NO`, it generates no orientation notifications. Device orientation notifications can only be generated between calls to the `beginGeneratingDeviceOrientationNotifications` and `endGeneratingDeviceOrientationNotifications` methods.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `beginGeneratingDeviceOrientationNotifications` (page 188)
- `endGeneratingDeviceOrientationNotifications` (page 188)

Declared In
`UIDevice.h`

localizedModel

The model of the device as a localized string. (read-only)

```
@property(nonatomic, readonly, retain) NSString *localizedModel
```

Discussion
This string would be a localized version of the string returned from `model` (page 185).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDevice.h`

model

The model of the device. (read-only)

```
@property(nonatomic, readonly, retain) NSString *model
```

Discussion
Possible examples of model strings are @"iPhone" and @"iPod touch".

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDevice.h`

name

The name identifying the device. (read-only)

```
@property(nonatomic, readonly, retain) NSString *name
```

Discussion
The value of this property is an arbitrary string that is associated with the device as an identifier. For example, you can find the name of an iPhone in the General > About settings.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property systemName` (page 186)

Declared In
`UIDevice.h`

orientation

Returns the physical orientation of the device. (read-only)

```
@property(nonatomic, readonly) UIDeviceOrientation orientation
```

Discussion
The value of the property is a constant that indicates the current orientation of the device. This value represents the physical orientation of the device and may be different from the current orientation of your application's user interface. See "UIDeviceOrientation" (page 189) for descriptions of the possible values.

The value of this property always returns 0 unless orientation notifications have been enabled by calling `beginGeneratingDeviceOrientationNotifications` (page 188).

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property generatesDeviceOrientationNotifications` (page 184)
`- beginGeneratingDeviceOrientationNotifications` (page 188)

Declared In
`UIDevice.h`

systemName

The name of the operating system running on the device represented by the receiver. (read-only)

```
@property(nonatomic, readonly, retain) NSString *systemName
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property systemVersion` (page 186)

Declared In
`UIDevice.h`

systemVersion

The current version of the operating system. (read-only)

```
@property(nonatomic, readonly, retain) NSString *systemVersion
```

Discussion
An example of the system version is @"1.2".

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property systemName` (page 186)

Declared In
`UIDevice.h`

uniqueIdentifier

A string unique to each device based on various hardware details. (read-only)

```
@property(nonatomic, readonly, retain) NSString *uniqueIdentifier
```

Discussion
A unique device identifier is a hash value composed from various hardware identifiers such as the device's serial number. It is guaranteed to be unique for every device. You can use it, for example, to store high scores for a game in a central server or to control access to registered products.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDevice.h`

Class Methods

currentDevice

Returns an object representing the current device.

```
+ (UIDevice *)currentDevice
```

Return Value
A singleton object that represents the current device.

Discussion
You evaluate the properties of the returned `UIDevice` instance to obtain information about the device.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDevice.h`

Instance Methods

beginGeneratingDeviceOrientationNotifications

Begins the generation of notifications of device orientation changes.

```
- (void)beginGeneratingDeviceOrientationNotifications
```

Discussion

You must call this method before attempting to get orientation data from the receiver. This method enables the device's accelerometer hardware and begins the delivery of acceleration events to the receiver. The receiver subsequently uses these events to post `UIDeviceOrientationDidChangeNotification` (page 190) notifications when the device orientation changes and to update the `orientation` property.

You may nest calls to this method safely, but you should always match each call with a corresponding call to the `endGeneratingDeviceOrientationNotifications` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `endGeneratingDeviceOrientationNotifications` (page 188)
- `@property orientation` (page 186)
- `@property generatesDeviceOrientationNotifications` (page 184)

Declared In

`UIDevice.h`

endGeneratingDeviceOrientationNotifications

Ends the generation of notifications of device orientation changes.

```
- (void)endGeneratingDeviceOrientationNotifications
```

Discussion

This method stops the posting of `UIDeviceOrientationDidChangeNotification` (page 190) notifications and notifies the system that it can power down the accelerometer hardware if it is not in use elsewhere. You call this method after a previous call to the `beginGeneratingDeviceOrientationNotifications` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `beginGeneratingDeviceOrientationNotifications` (page 188)

Declared In

`UIDevice.h`

Constants

UIDeviceOrientation

The physical orientation of the current device.

```
typedef enum {
    UIDeviceOrientationUnknown,
    UIDeviceOrientationPortrait,
    UIDeviceOrientationPortraitUpsideDown,
    UIDeviceOrientationLandscapeLeft,
    UIDeviceOrientationLandscapeRight,
    UIDeviceOrientationFaceUp,
    UIDeviceOrientationFaceDown
} UIDeviceOrientation;
```

Constants

`UIDeviceOrientationUnknown`
The orientation of the device cannot be determined.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationPortrait`
The device is in portrait mode, with the device held upright and the home button at the bottom.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationPortraitUpsideDown`
The device is in portrait mode but upside down, with the device held upright and the home button at the top.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationLandscapeLeft`
The device is in landscape mode, with the device held upright and the home button on the right side.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationLandscapeRight`
The device is in landscape mode, with the device held upright and the home button on the left side.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationFaceUp`
The device is held perpendicular to the ground with the screen facing upwards.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationFaceDown`
The device is held perpendicular to the ground with the screen facing downwards.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

Discussion
The `orientation` (page 186) property uses these constants to identify the device orientation. These constants identify the physical orientation of the device and are not tied to the orientation of your application's user interface.

Device Orientation Convenience Macros

Macros for determining whether the device orientation is landscape or portrait.

```
#define UIDeviceOrientationIsPortrait(orientation)  ((orientation) ==
UIDeviceOrientationPortrait || (orientation) ==
UIDeviceOrientationPortraitUpsideDown)
#define UIDeviceOrientationIsLandscape(orientation) ((orientation) ==
UIDeviceOrientationLandscapeLeft || (orientation) ==
UIDeviceOrientationLandscapeRight)
```

Constants
`UIDeviceOrientationIsPortrait`
Returns `YES` if the device orientation is portrait, otherwise returns `NO`.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

`UIDeviceOrientationIsLandscape`
Returns `YES` if the device orientation is landscape, otherwise returns `NO`.

Available in iPhone OS 2.0 and later.

Declared in `UIDevice.h`

Notifications

UIDeviceOrientationDidChangeNotification

Posted when the orientation of the device changes.

You can obtain the new orientation by getting the current value of the `orientation` (page 186) property.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIDevice.h`

UIEvent Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIEvent.h

Overview

A `UIEvent` object (or, simply, an event object) represents an event in iPhone OS. An event object contains one or more touches (that is, finger gestures on the screen) that have some relation to the event. A touch is represented by a `UITouch` object.

When an event occurs, the system routes it to the appropriate responder and passes in the `UIEvent` object in a message invoking a `UIResponder` method such as `touchesBegan:withEvent:` (page 283). The responder can then evaluate the touches for the event or for a particular phase of the event and handle them appropriately. The methods of `UIEvent` allow you to obtain all touches for the event (`allTouches` (page 192)) or only those for a given view or window (`touchesForView:` (page 193) or `touchesForWindow:` (page 193)). It can also distinguish an event object from objects representing other events by querying an object for the time of its creation (`timestamp` (page 192)).

A `UIEvent` object is persistent throughout a multi-touch sequence; UIKit reuses the same `UIEvent` instance for every event delivered to the application. You should never retain an event object or any object returned from an event object. If you need to keep information from an event around from one phase to another, you should copy that information from the `UITouch` or `UIEvent` object.

See "Event Handling" in *iPhone OS Programming Guide* for further information on event handling.

Tasks

Getting the Touches for an Event

- `allTouches` (page 192)
  Returns all touch objects associated with the receiver.
- `touchesForView:` (page 193)
  Returns the touch objects that belong to a given view for the event represented by the receiver.
- `touchesForWindow:` (page 193)
  Returns the touch objects that belong to a given window for the event represented by the receiver.

Getting Event Attributes

`timestamp` (page 192) *property*
The time when the event occurred. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

timestamp

The time when the event occurred. (read-only)

```
@property(nonatomic, readonly) NSTimeInterval timestamp
```

Discussion
The property value is the number of seconds since system startup.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIEvent.h`

Instance Methods

allTouches

Returns all touch objects associated with the receiver.

```
- (NSSet *)allTouches
```

Return Value
A set of `UITouch` objects representing all touches associated with an event (represented by the receiver).

Discussion
If the touches of the event originate in different views and windows, the `UITouch` objects obtained from this method will have different responder objects associated with the touches.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `touchesForView:` (page 193)
- `touchesForWindow:` (page 193)

Declared In
`UIEvent.h`

touchesForView:

Returns the touch objects that belong to a given view for the event represented by the receiver.

```
- (NSSet *)touchesForView:(UIView *)view
```

Parameters

view
The `UIView` object on which the touches related to the event were made.

Return Value
An set of `UITouch` objects representing the touches for the specified view related to the event represented by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allTouches` (page 192)
- `touchesForWindow:` (page 193)

Declared In
`UIEvent.h`

touchesForWindow:

Returns the touch objects that belong to a given window for the event represented by the receiver.

```
- (NSSet *)touchesForWindow:(UIWindow *)window
```

Parameters

window
The `UIWindow` object on which the touches related to the event were made.

Return Value
An set of `UITouch` objects representing the touches for the specified window related to the event represented by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allTouches` (page 192)
- `touchesForView:` (page 193)

Declared In
`UIEvent.h`

UIFont Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIFont.h UIInterface.h

Overview

The `UIFont` class provides the interface for getting and setting font information. The class provides you with access to the font's characteristics and also provides the system with access to the font's glyph information, which is used during layout. You use font objects by passing them to methods that accept them as a parameter.

You do not create `UIFont` objects using the `alloc` and `init` methods. Instead, you use class methods of `UIFont` to look up and retrieve the desired font object. These methods check for an existing font object with the specified characteristics and return it if it exists. Otherwise, they create a new font object based on the desired font characteristics.

Tasks

Creating Arbitrary Fonts

`+ fontWithName:size:` (page 201)
Creates and returns a font object for the specified font name and size.

`- fontWithSize:` (page 203)
Returns a font object that is the same as the receiver but which has the specified size instead.

Creating System Fonts

`+ systemFontOfSize:` (page 203)
Returns the font object used for standard interface items in the specified size.

`+ boldSystemFontOfSize:` (page 199)
Returns the font object used for standard interface items that are rendered in boldface type in the specified size.

`+ italicSystemFontOfSize:` (page 202)
Returns the font object used for standard interface items that are rendered in italic type in the specified size.

Getting the Available Font Names

`+ familyNames` (page 200)
Returns an array of font family names available on the system.

`+ fontNamesForFamilyName:` (page 201)
Returns an array of font names available in a particular font family.

Getting Font Name Attributes

`familyName` (page 198) *property*
Specifies the receiver's family name. (read-only)

`fontName` (page 198) *property*
Specifies the font face name. (read-only)

Getting Font Metrics

`pointSize` (page 199) *property*
Specifies receiver's point size, or the effective vertical point size for a font with a nonstandard matrix. (read-only)

`ascender` (page 197) *property*
Specifies the top y-coordinate, offset from the baseline, of the receiver's longest ascender. (read-only)

`descender` (page 198) *property*
Specifies the bottom y-coordinate, offset from the baseline, of the receiver's longest descender. (read-only)

`leading` (page 199) *property*
Specifies the receiver's leading information. (read-only)

`capHeight` (page 197) *property*
Specifies the receiver's cap height information. (read-only)

`xHeight` (page 199) *property*
Specifies the x-height of the receiver. (read-only)

Getting System Font Information

`+ labelFontSize` (page 202)
Returns the standard font size used for labels.

`+ buttonFontSize` (page 200)
Returns the standard font size used for buttons.

`+ smallSystemFontSize` (page 202)
Returns the size of the standard small system font.

`+ systemFontSize` (page 203)
Returns the size of the standard system font.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

ascender

Specifies the top y-coordinate, offset from the baseline, of the receiver's longest ascender. (read-only)

```
@property(nonatomic, readonly) CGFloat ascender
```

Discussion
The ascender value is measured in points.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property descender` (page 198)

Declared In
`UIFont.h`

capHeight

Specifies the receiver's cap height information. (read-only)

```
@property(nonatomic, readonly) CGFloat capHeight
```

Discussion
This value measures (in points) the height of a capital character.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIFont.h`

descender

Specifies the bottom y-coordinate, offset from the baseline, of the receiver's longest descender. (read-only)

```
@property(nonatomic, readonly) CGFloat descender
```

Discussion

The descender value is measured in points. This value may be positive or negative. For example, if the longest descender extends 2 points below the baseline, this method returns `-2.0`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property ascender` (page 197)

Declared In

`UIFont.h`

familyName

Specifies the receiver's family name. (read-only)

```
@property(nonatomic, readonly, retain) NSString *familyName
```

Discussion

A family name is a name such as `Times New Roman` that identifies one or more specific fonts. The value in this property is intended for an application's internal usage only and should not be displayed.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIFont.h`

fontName

Specifies the font face name. (read-only)

```
@property(nonatomic, readonly, retain) NSString *fontName
```

Discussion

The font name is a name such as `HelveticaBold` that incorporates the family name and any specific style information for the font. The value in this property is intended for an application's internal usage only and should not be displayed.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIFont.h`

leading

Specifies the receiver's leading information. (read-only)

```
@property(nonatomic, readonly) CGFloat leading
```

Discussion
The leading value represents the spacing between lines of text and is measured (in points) from baseline to baseline.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIFont.h

pointSize

Specifies receiver's point size, or the effective vertical point size for a font with a nonstandard matrix. (read-only)

```
@property(nonatomic, readonly) CGFloat pointSize
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIFont.h

xHeight

Specifies the x-height of the receiver. (read-only)

```
@property(nonatomic, readonly) CGFloat xHeight
```

Discussion
This value measures (in points) the height of the lowercase character "x".

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIFont.h

Class Methods

boldSystemFontOfSize:

Returns the font object used for standard interface items that are rendered in boldface type in the specified size.

```
+ (UIFont *)boldSystemFontOfSize:(CGFloat)fontSize
```

Parameters

fontSize

The size (in points) to which the font is scaled. This value must be greater than 0.0.

Return Value

A font object of the specified size.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIFont.h

buttonFontSize

Returns the standard font size used for buttons.

```
+ (CGFloat)buttonFontSize
```

Return Value

The standard button font size in points.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIInterface.h

familyNames

Returns an array of font family names available on the system.

```
+ (NSArray *)familyNames
```

Return Value

An array of `NSString` objects, each of which contains the name of a font family.

Discussion

Font family names correspond to the base name of a font, such as `Times New Roman`. You can pass the returned strings to the `fontNamesForFamilyName:` (page 201) method to retrieve a list of font names available for that family. You can then use the corresponding font name to retrieve an actual font object.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ fontNamesForFamilyName:` (page 201)

Declared In

UIFont.h

fontNamesForFamilyName:

Returns an array of font names available in a particular font family.

```
+ (NSArray *)fontNamesForFamilyName:(NSString *)familyName
```

Parameters

familyName

The name of the font family. Use the `familyNames` method to get an array of the available font family names on the system.

Return Value

An array of `NSString` objects, each of which contains a font name associated with the specified family.

Discussion

You can pass the returned strings as parameters to the `fontWithName:size:` (page 201) method to retrieve an actual font object.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ familyNames` (page 200)

`+ fontWithName:size:` (page 201)

Declared In

`UIFont.h`

fontWithName:size:

Creates and returns a font object for the specified font name and size.

```
+ (UIFont *)fontWithName:(NSString *)fontName size:(CGFloat)fontSize
```

Parameters

fontName

The fully specified name of the font. This name incorporates both the font family name and the specific style information for the font.

fontSize

The size (in points) to which the font is scaled. This value must be greater than 0.0.

Return Value

A font object of the specified name and size.

Discussion

You can use the `fontNamesForFamilyName:` (page 201) method to retrieve the specific font names for a given font family.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ familyNames` (page 200)

`+ fontNamesForFamilyName:` (page 201)

Declared In
UIFont.h

italicSystemFontOfSize:

Returns the font object used for standard interface items that are rendered in italic type in the specified size.

```
+ (UIFont *)italicSystemFontOfSize:(CGFloat)fontSize
```

Parameters

fontSize
The size (in points) to which the font is scaled. This value must be greater than 0.0.

Return Value
A font object of the specified size.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIFont.h

labelFontSize

Returns the standard font size used for labels.

```
+ (CGFloat)labelFontSize
```

Return Value
The standard label font size in points.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIInterface.h

smallSystemFontSize

Returns the size of the standard small system font.

```
+ (CGFloat)smallSystemFontSize
```

Return Value
The standard small system font size in points.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIInterface.h

systemFontOfSize:

Returns the font object used for standard interface items in the specified size.

```
+ (UIFont *)systemFontOfSize:(CGFloat)fontSize
```

Parameters

fontSize

The size (in points) to which the font is scaled. This value must be greater than 0.0.

Return Value

A font object of the specified size.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIFont.h

systemFontSize

Returns the size of the standard system font.

```
+ (CGFloat)systemFontSize
```

Return Value

The standard system font size in points.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIInterface.h

Instance Methods

fontWithSize:

Returns a font object that is the same as the receiver but which has the specified size instead.

```
- (UIFont *)fontWithSize:(CGFloat)fontSize
```

Parameters

fontSize

The desired size (in points) of the new font object. This value must be greater than 0.0.

Return Value

A font object of the specified size.

Availability

Available in iPhone OS 2.0 and later.

Declared In
`UIFont.h`

UIImage Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIKit/UIImage.h UIKit/UIInterface.h

Overview

A `UIImage` object is a high-level way to display image data. You can create images from files, from Quartz image objects, or from raw image data you receive. The `UIImage` class also offers several options for drawing images to the current graphics context using different blend modes and opacity values.

Image objects are immutable, so you cannot change their properties after creation. This means that you generally specify an image's properties at initialization time or rely on the image's metadata to provide the property value. In some cases, however, the `UIImage` class provides convenience methods for obtaining a copy of the image that uses custom values for a property.

Because image objects are immutable, they also do not provide direct access to their underlying image data. However, you can get an `NSData` object containing either a PNG or JPEG representation of the image data using the `UIImagePNGRepresentation` (page 646) and `UIImageJPEGRepresentation` (page 646) functions.

Images and Memory Management

In low-memory situations, image data may be purged from a `UIImage` object to free up memory on the system. This purging behavior affects only the image data stored internally by the `UIImage` object and not the object itself. When you attempt to draw an image whose data has been purged, the image object automatically reloads the data from its original file. This extra load step, however, may incur a small performance penalty.

It is a programmer error to create a `UIImage` object with an image that is greater than 1024 x 1024 pixels in size. Besides the practical considerations of such an image consuming a large amount of memory, the graphics hardware does not support images greater than that size.

Supported Image Formats

Table 22-1 lists the file formats that can be read by the `UIImage` class.

Table 22-1 Supported file formats

Format	Filename extensions
Tagged Image File Format (TIFF)	`.tiff`, `.tif`
Joint Photographic Experts Group (JPEG)	`.jpg`, `.jpeg`
Graphic Interchange Format (GIF)	`.gif`
Portable Network Graphic (PNG)	`.png`
Windows Bitmap Format (DIB)	`.bmp`, `.BMPf`
Windows Icon Format	`.ico`
Windows Cursor	`.cur`
XWindow bitmap	`.xbm`

Note: Windows Bitmap Format (BMP) files that are formatted as RGB-565 are converted to ARGB-1555 when they are loaded.

Tasks

Cached Image Loading Routines

`+ imageNamed:` (page 210)
Returns the image object associated with the specified filename.

Creating New Images

`+ imageWithContentsOfFile:` (page 211)
Creates and returns an image object by loading the image data from the file at the specified path.

`+ imageWithData:` (page 211)
Creates and returns an image object that uses the specified image data.

+ imageWithCGImage: (page 210)
Creates and returns an image object representing the specified Quartz image.

- stretchableImageWithLeftCapWidth:topCapHeight: (page 215)
Creates and returns a new image object with the specified cap values.

Initializing Images

- initWithContentsOfFile: (page 214)
Initializes and returns the image object with the contents of the specified file.

- initWithData: (page 215)
Initializes and returns the image object with the specified data.

- initWithCGImage: (page 214)
Initializes and returns the image object with the specified Quartz image reference.

Image Attributes

imageOrientation (page 208) *property*
The orientation of the receiver's image. (read-only)

size (page 209) *property*
The dimensions of the image, taking orientation into account (read-only)

CGImage (page 208) *property*
The underlying Quartz image data. (read-only)

leftCapWidth (page 208) *property*
The horizontal end-cap size. (read-only)

topCapHeight (page 209) *property*
The vertical end-cap size. (read-only)

Drawing Images

- drawAtPoint: (page 212)
Draws the image at the specified point in the current context.

- drawAtPoint:blendMode:alpha: (page 212)
Draws the entire image at the specified point using the custom compositing options.

- drawInRect: (page 213)
Draws the entire image in the specified rectangle, scaling it as needed to fit.

- drawInRect:blendMode:alpha: (page 213)
Draws the entire image in the specified rectangle and using the specified compositing options.

- drawAsPatternInRect: (page 212)
Draws a tiled Quartz pattern using the receiver's contents as the tile pattern.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

CGImage

The underlying Quartz image data. (read-only)

```
@property(nonatomic, readonly) CGImageRef CGImage
```

Discussion
If the image data has been purged because of memory constraints, invoking this method forces that data to be loaded back into memory. Reloading the image data may incur a performance penalty.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

imageOrientation

The orientation of the receiver's image. (read-only)

```
@property(nonatomic, readonly) UIImageOrientation imageOrientation
```

Discussion
Image orientation affects the way the image data is displayed when drawn. By default, images are displayed in the "up" orientation. If the image has associated metadata (such as EXIF information), however, this property contains the orientation indicated by that metadata. For a list of possible values for this property, see "UIImageOrientation" (page 216).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

leftCapWidth

The horizontal end-cap size. (read-only)

```
@property(nonatomic, readonly) NSInteger leftCapWidth
```

Discussion
End caps specify the portion of an image that should not be resized when an image is stretched. This technique is used to implement buttons and other resizable image-based interface elements. When a button with end caps is resized, the resizing occurs only in the middle of the button, in the region between the end caps. The end caps themselves keep their original size and appearance.

This property specifies the size of the left end cap. The middle (stretchable) portion is assumed to be 1 pixel wide. The right end cap is therefore computed by adding the size of the left end cap and the middle portion together and then subtracting that value from the width of the image:

```
rightCapWidth = image.size.width - (image.leftCapWidth + 1);
```

By default, this property is set to 0, which indicates that the image does not use end caps and the entire image is subject to stretching. To create a new image with a nonzero value for this property, use the `stretchableImageWithLeftCapWidth:topCapHeight:` method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stretchableImageWithLeftCapWidth:topCapHeight:` (page 215)

Declared In
`UIImage.h`

size
The dimensions of the image, taking orientation into account (read-only)

```
@property(nonatomic, readonly) CGSize size
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

topCapHeight
The vertical end-cap size. (read-only)

```
@property(nonatomic, readonly) NSInteger topCapHeight
```

Discussion
End caps specify the portion of an image that should not be resized when an image is stretched. This technique is used to implement buttons and other resizable image-based interface elements. When a button with end caps is resized, the resizing occurs only in the middle of the button, in the region between the end caps. The end caps themselves keep their original size and appearance.

This property specifies the size of the top end cap. The middle (stretchable) portion is assumed to be 1 pixel wide. The bottom end cap is therefore computed by adding the size of the top end cap and the middle portion together and then subtracting that value from the height of the image:

```
bottomCapHeight = image.size.height - (image.topCapHeight + 1);
```

By default, this property is set to 0, which indicates that the image does not use end caps and the entire image is subject to stretching. To create a new image with a nonzero value for this property, use the `stretchableImageWithLeftCapWidth:topCapHeight:` method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stretchableImageWithLeftCapWidth:topCapHeight:` (page 215)

Declared In
`UIImage.h`

Class Methods

imageNamed:

Returns the image object associated with the specified filename.

```
+ (UIImage *)imageNamed:(NSString *)name
```

Parameters

name
The name of the file, including its filename extension. If this is the first time the image is being loaded, the method looks for an image with the specified name in the application's main bundle.

Return Value
The image object for the specified file, or `nil` if the method could not find the specified image.

Discussion
This method looks in the system caches for an image object with the specified name and returns that object if it exists. If a matching image object is not already in the cache, this method loads the image data from the specified file, caches it, and then returns the resulting object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

imageWithCGImage:

Creates and returns an image object representing the specified Quartz image.

```
+ (UIImage *)imageWithCGImage:(CGImageRef)cgImage
```

Parameters

cgImage
The Quartz image object.

Return Value
A new image object for the specified Quartz image, or `nil` if the method could not initialize the image from the specified image reference.

Discussion
This method does not cache the image object. You can use the methods of the Core Graphics framework to create a Quartz image reference.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImage.h

imageWithContentsOfFile:
Creates and returns an image object by loading the image data from the file at the specified path.

```
+ (UIImage *)imageWithContentsOfFile:(NSString *)path
```

Parameters
path
The full or partial path to the file.

Return Value
A new image object for the specified file, or `nil` if the method could not initialize the image from the specified file.

Discussion
This method does not cache the image object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImage.h

imageWithData:
Creates and returns an image object that uses the specified image data.

```
+ (UIImage *)imageWithData:(NSData *)data
```

Parameters
data
The image data. This can be data from a file or data you create programmatically.

Return Value
A new image object for the specified data, or `nil` if the method could not initialize the image from the specified data.

Discussion
This method does not cache the image object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImage.h

Instance Methods

drawAsPatternInRect:

Draws a tiled Quartz pattern using the receiver's contents as the tile pattern.

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

Parameters

rect

The rectangle (in the coordinate system of the graphics context) in which to draw the image.

Discussion

This method uses a Quartz pattern to tile the image in the specified rectangle. The image is tiled with no gaps and the fill color is ignored. In the default coordinate system, the image tiles are situated down and to the right of the origin of the specified rectangle. This method respects any transforms applied to the current graphics context, however.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

drawAtPoint:

Draws the image at the specified point in the current context.

```
- (void)drawAtPoint:(CGPoint)point
```

Parameters

point

The point at which to draw the top-left corner of the image.

Discussion

This method draws the entire image in the current graphics context, respecting the image's orientation setting. In the default coordinate system, images are situated down and to the right of the specified point. This method respects any transforms applied to the current graphics context, however.

This method draws the image at full opacity using the `kCGBlendModeNormal` blend mode.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

drawAtPoint:blendMode:alpha:

Draws the entire image at the specified point using the custom compositing options.

```
- (void)drawAtPoint:(CGPoint)point blendMode:(CGBlendMode)blendMode
    alpha:(CGFloat)alpha
```

Parameters

point

The point at which to draw the top-left corner of the image.

blendMode

The blend mode to use when compositing the image.

alpha

The desired opacity of the image, specified as a value between 0.0 and 1.0. A value of 0.0 renders the image totally transparent while 1.0 renders it fully opaque. Values larger than 1.0 are interpreted as 1.0.

Discussion

This method draws the entire image in the current graphics context, respecting the image's orientation setting. In the default coordinate system, images are situated down and to the right of the specified point. This method respects any transforms applied to the current graphics context, however.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

drawInRect:

Draws the entire image in the specified rectangle, scaling it as needed to fit.

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

Parameters

rect

The rectangle (in the coordinate system of the graphics context) in which to draw the image.

Discussion

This method draws the entire image in the current graphics context, respecting the image's orientation setting. In the default coordinate system, images are situated down and to the right of the origin of the specified rectangle. This method respects any transforms applied to the current graphics context, however.

This method draws the image at full opacity using the `kCGBlendModeNormal` blend mode.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

drawInRect:blendMode:alpha:

Draws the entire image in the specified rectangle and using the specified compositing options.

```
- (void)drawInRect:(CGRect)rect blendMode:(CGBlendMode)blendMode alpha:(CGFloat)alpha
```

Parameters

rect

The rectangle (in the coordinate system of the graphics context) in which to draw the image.

blendMode

The blend mode to use when compositing the image.

alpha

The desired opacity of the image, specified as a value between 0.0 and 1.0. A value of 0.0 renders the image totally transparent while 1.0 renders it fully opaque. Values larger than 1.0 are interpreted as 1.0.

Discussion

This method scales the image as needed to make it fit in the specified rectangle. This method draws the image in the current graphics context, respecting the image's orientation setting. In the default coordinate system, images are situated down and to the right of the origin of the specified rectangle. This method respects any transforms applied to the current graphics context, however.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

initWithCGImage:

Initializes and returns the image object with the specified Quartz image reference.

```
- (id)initWithCGImage:(CGImageRef)CGImage
```

Parameters

CGImage

A Quartz image reference.

Return Value

An initialized `UIImage` object, or `nil` if the method could not initialize the image from the specified data.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

initWithContentsOfFile:

Initializes and returns the image object with the contents of the specified file.

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

Parameters

path

The path to the file. This path should include the filename extension that identifies the type of the image data.

Return Value
An initialized `UIImage` object, or `nil` if the method could find the file or initialize the image from its contents.

Discussion
This method loads the image data into memory and marks it as purgeable. If the data is purged and needs to be reloaded, the image object loads that data again from the specified path.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

initWithData:
Initializes and returns the image object with the specified data.

```
- (id)initWithData:(NSData *)data
```

Parameters
data

The data object containing the image data.

Return Value
An initialized `UIImage` object, or `nil` if the method could not initialize the image from the specified data.

Discussion
The data in the *data* parameter must be formatted to match the file format of one of the system's supported image types.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

stretchableImageWithLeftCapWidth:topCapHeight:
Creates and returns a new image object with the specified cap values.

```
- (UIImage *)stretchableImageWithLeftCapWidth:(NSInteger)leftCapWidth
    topCapHeight:(NSInteger)topCapHeight
```

Parameters
leftCapWidth

The value to use for the left cap width. Specify `0` if you do not want the image to be horizontally stretchable. For a discussion of how this value affects the image, see the `leftCapWidth` (page 208) property.

topCapHeight

The value to use for the top cap width. Specify `0` if you do not want the image to be vertically stretchable. For a discussion of how this value affects the image, see the `topCapHeight` (page 209) property.

Return Value

A new image object with the specified cap values.

Discussion

During scaling or resizing of the image, areas covered by a cap are not scaled or resized. Instead, the 1-pixel wide area not covered by the cap in each direction is what is scaled or resized. This technique is often used to create variable-width buttons, which retain the same rounded corners but whose center region grows or shrinks as needed.

You use this method to add cap values to an image or to change the existing cap values of an image. In both cases, you get back a new image and the original image remains untouched.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImage.h`

Constants

UIImageOrientation

Specifies the possible orientations of an image.

```
typedef enum {
    UIImageOrientationUp,
    UIImageOrientationDown,   // 180 deg rotation
    UIImageOrientationLeft,   // 90 deg CCW
    UIImageOrientationRight,   // 90 deg CW
    UIImageOrientationUpMirrored,    // as above but image mirrored along
                                     // other axis. horizontal flip
    UIImageOrientationDownMirrored,  // horizontal flip
    UIImageOrientationLeftMirrored,  // vertical flip
    UIImageOrientationRightMirrored, // vertical flip
} UIImageOrientation;
```

Constants

`UIImageOrientationUp`

The default orientation of images. The image is drawn right-side up, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationDown`

The image is rotated 180 degrees, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationLeft`

The image is rotated 90 degrees counterclockwise, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationRight`

The image is rotated 90 degrees clockwise, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationUpMirrored`

The image is drawn as a mirror version of an image drawn with the `UIImageOrientationUp` value. In other words, the image is flipped along its horizontal axis, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationDownMirrored`

The image is drawn as a mirror version of an image drawn with the `UIImageOrientationDown` value. This is the equivalent to flipping an image in the “up” orientation along its horizontal axis and then rotating the image 180 degrees, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationLeftMirrored`

The image is drawn as a mirror version of an image drawn with the `UIImageOrientationLeft` value. This is the equivalent to flipping an image in the “up” orientation along its horizontal axis and then rotating the image 90 degrees counterclockwise, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

`UIImageOrientationRightMirrored`

The image is drawn as a mirror version of an image drawn with the `UIImageOrientationRight` value. This is the equivalent to flipping an image in the “up” orientation along its horizontal axis and then rotating the image 90 degrees clockwise, as shown here. 

Available in iPhone OS 2.0 and later.

Declared in `UIImage.h`

Declared In

`UIImage.h`

UIImagePickerController Class Reference

Inherits from:	UINavigationController : UIViewController : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIImagePickerController.h

Overview

The `UIImagePickerController` class manages the system-supplied user interfaces for choosing and taking pictures. You use this class in situations where you want to obtain a picture from the user. The class manages the actual user interactions with the views and reports the results of those interactions to your delegate object.

Because the `UIImagePickerController` class handles all of the user interactions, all you have to do is tell it which user interface to display, tell it to start, and then dismiss it when the user picks an image or cancels. Before starting an interface, however, you should always verify that the interface is supported by the current device by calling the `isSourceTypeAvailable:` (page 221) class method.

You must provide a delegate that conforms to the `UIImagePickerControllerDelegate` protocol in order to use this class. After the interface starts, this class notifies your delegate of the user's actions. Your delegate is then responsible for dismissing the picker and returning to your application's views.

Tasks

Setting the Picker Source

`+ isSourceTypeAvailable:` (page 221)
Returns a Boolean value indicating whether the device supports picking images using the specified source.

`sourceType` (page 221) *property*
The type of interface displayed by the controller.

Configuring the Picker

`delegate` (page 220) *property*
The receiver's delegate.

`allowsImageEditing` (page 220) *property*
A Boolean value indicating whether the user is allowed to edit a selected image.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

allowsImageEditing

A Boolean value indicating whether the user is allowed to edit a selected image.

```
@property(nonatomic) BOOL allowsImageEditing
```

Discussion

If you allow the user to edit images, the delegate may receive a dictionary with information about the edits that were made.

This property is set to `NO` by default.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImagePickerController.h`

delegate

The receiver's delegate.

```
@property(nonatomic,assign) id <UINavigationControllerDelegate,
    UIImagePickerControllerDelegate> delegate
```

Discussion
The delegate receives notifications when the user picks an image or exits the picker interface. The delegate also makes the decision on when to dismiss the picker interface, so you must provide this object. If this property is `nil`, the picker is dismissed immediately if you try to show it.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImagePickerController.h`

sourceType

The type of interface displayed by the controller.

```
@property(nonatomic) UIImagePickerControllerSourceType sourceType
```

Discussion
Prior to running the interface, set this value to the desired interface type. If you change this property while the picker is visible, the picker interface changes to match the new value in this property.

This property is set to `UIImagePickerControllerSourceTypePhotoLibrary` by default. For a list of possible constants, see "UIImagePickerControllerSourceType" (page 222).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImagePickerController.h`

Class Methods

isSourceTypeAvailable:

Returns a Boolean value indicating whether the device supports picking images using the specified source.

```
+ (BOOL)isSourceTypeAvailable:(UIImagePickerControllerSourceType)sourceType
```

Parameters
sourceType
The source to use to pick an image.

Return Value
`YES` if the device supports the given source; `NO` if the specified source type is not available.

Discussion
Because an image source may not be present or may be unavailable, devices may not always support all source types. For example, if you attempt to pick an image from the user's library and the library is empty, this method returns `NO`. Similarly, if the camera is already in use, this method returns `NO`.

Before attempting to use an `UIImagePickerController` object to pick an image, you must call this method to ensure that the desired source type is available.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImagePickerController.h`

Constants

UIImagePickerControllerSourceType
The source to use when picking an image.

```
enum {
    UIImagePickerControllerSourceTypePhotoLibrary,
    UIImagePickerControllerSourceTypeCamera,
    UIImagePickerControllerSourceTypeSavedPhotosAlbum
};
typedef NSUInteger UIImagePickerControllerSourceType;
```

Constants
`UIImagePickerControllerSourceTypePhotoLibrary`
Pick an image from the device's photo library.
Available in iPhone OS 2.0 and later.
Declared in `UIImagePickerController.h`

`UIImagePickerControllerSourceTypeCamera`
Take a new picture using the device's built-in camera.
Available in iPhone OS 2.0 and later.
Declared in `UIImagePickerController.h`

`UIImagePickerControllerSourceTypeSavedPhotosAlbum`
Pick an image from the device's camera roll. If the device does not have a camera, pick an image from the Saved Photos folder on the device.
Available in iPhone OS 2.0 and later.
Declared in `UIImagePickerController.h`

Discussion
A given source may not always be available on every device. This could be because the source is not physically present or because it cannot currently be accessed.

UIImageView Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIImageView.h

Overview

An image view object provides a view-based container for displaying either a single image or for animating a series of images. For animating the images, the `UIImageView` class provides controls to set the duration and frequency of the animation. You can also start and stop the animation freely.

New image view objects are configured to disregard user events by default. If you want to handle events in a custom subclass of `UIImageView`, you must explicitly change the value of the `userInteractionEnabled` property to `YES` after initializing the object.

Tasks

Initializing a UIImageView Object

`- initWithImage:` (page 226)
Returns an image view initialized with the specified image.

Image Data

`image` (page 225) *property*
The image displayed in the image view.

Animating Images

`animationImages` (page 224) *property*
An array of `UIImage` objects to use for an animation.

`animationDuration` (page 224) *property*
The amount of time it takes to go through one cycle of the images.

`animationRepeatCount` (page 225) *property*
Specifies the number of times to repeat the animation.

- `startAnimating` (page 227)
Starts animating the images in the receiver.

- `stopAnimating` (page 227)
Stops animating the images in the receiver.

- `isAnimating` (page 226)
Returns a Boolean value indicating whether the animation is running.

Setting and Getting Attributes

`userInteractionEnabled` (page 226) *property*
A Boolean value that determines whether user events are ignored and removed from the event queue.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

animationDuration

The amount of time it takes to go through one cycle of the images.

```
@property(nonatomic) NSTimeInterval animationDuration
```

Discussion

The time duration is measured in seconds. The default value of this property is equal to the number of images multiplied by 1/30th of a second. Thus, if you had 30 images, the value would be 1 second.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImageView.h`

animationImages

An array of `UIImage` objects to use for an animation.

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

Discussion
The array must contain `UIImage` objects. You may use the same image object more than once in the array. Setting this property to a value other than `nil` hides the image represented by the `image` property. The value of this property is `nil` by default.

The images assigned to this property are scaled, sized to fit, or positioned according to the `contentMode` property of the view. It is recommended (but not required) that you use images that are all the same size. If the images are different sizes, each will be adjusted to fit separately based on that mode.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property image` (page 225)
`@property contentMode` (page 448) (UIView)

Declared In
`UIImageView.h`

animationRepeatCount

Specifies the number of times to repeat the animation.

```
@property(nonatomic) NSInteger animationRepeatCount
```

Discussion
The default value is 0, which specifies to repeat the animation indefinitely.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImageView.h`

image

The image displayed in the image view.

```
@property(nonatomic, retain) UIImage *image
```

Discussion
The value of this property is `nil` by default. Setting this property automatically adjusts the frame of the receiver to match the size of the image.

If the `animationImages` property contains a value other than `nil`, the contents of this property are not used.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property animationImages` (page 224)

Declared In
UIImageView.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
This property is inherited from the UIView parent class. This class changes the default value of this property to NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImageView.h

Instance Methods

initWithImage:

Returns an image view initialized with the specified image.

```
- (id)initWithImage:(UIImage *)image
```

Parameters
image
The initial image to display in the image view.

Return Value
An initialized image view object.

Discussion
This method disables user interactions for the image view by default.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImageView.h

isAnimating

Returns a Boolean value indicating whether the animation is running.

```
- (BOOL)isAnimating
```

Return Value
YES if the animation is running; otherwise, NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImageView.h

startAnimating

Starts animating the images in the receiver.

```
- (void)startAnimating
```

Discussion
This method always starts the animation from the first image in the list.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImageView.h

stopAnimating

Stops animating the images in the receiver.

```
- (void)stopAnimating
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImageView.h

UILabel Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UILabel.h

Overview

The `UILabel` class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base `UILabel` class provides control over the appearance of your text, including whether it uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

The default content mode of the `UILabel` class is `UIViewContentModeRedraw` (page 479). This mode causes the view to redraw its contents every time its bounding rectangle changes. You can change this mode by modifying the inherited `contentMode` (page 448) property of the class.

New label objects are configured to disregard user events by default. If you want to handle events in a custom subclass of `UILabel`, you must explicitly change the value of the `userInteractionEnabled` property to `YES` after initializing the object.

Tasks

Accessing the Text Attributes

`text` (page 235) *property*
The text displayed by the label.

`font` (page 232) *property*
The font of the text.

`textColor` (page 236) *property*
The color of the text.

`textAlignment` (page 236) *property*
The technique to use for aligning the text.

`lineBreakMode` (page 233) *property*
The technique to use for wrapping and truncating the label's text.

`enabled` (page 232) *property*
The enabled state to use when drawing the label's text.

Sizing the Label's Text

`adjustsFontSizeToFitWidth` (page 231) *property*
A Boolean value indicating whether the font size should be reduced in order to fit the title string into the label's bounding rectangle.

`baselineAdjustment` (page 231) *property*
Controls how text baselines are adjusted when text needs to shrink to fit in the label.

`minimumFontSize` (page 234) *property*
The size of the smallest permissible font with which to draw the label's text.

`numberOfLines` (page 234) *property*
The maximum number of lines to use for rendering text.

Managing Highlight Values

`highlightedTextColor` (page 233) *property*
The highlight color applied to the label's text.

`highlighted` (page 232) *property*
A Boolean value indicating whether the receiver should be drawn with a highlight.

Drawing a Shadow

`shadowColor` (page 235) *property*
The shadow color of the text.

`shadowOffset` (page 235) *property*
The shadow offset (measured in points) for the text.

Drawing and Positioning Overrides

`- textRectForBounds:limitedToNumberOfLines:` (page 237)
Returns the drawing rectangle for the label's text.

`- drawTextInRect:` (page 237)
Draws the receiver's text (or its shadow) in the specified rectangle.

Setting and Getting Attributes

`userInteractionEnabled` (page 236) *property*
A Boolean value that determines whether user events are ignored and removed from the event queue.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

adjustsFontSizeToFitWidth

A Boolean value indicating whether the font size should be reduced in order to fit the title string into the label's bounding rectangle.

```
@property(nonatomic) BOOL adjustsFontSizeToFitWidth
```

Discussion

Normally, the label text is drawn with the font you specify in the `font` property. If this property is set to `YES`, however, and the text in the `text` property exceeds the label's bounding rectangle, the receiver starts reducing the font size until the string fits or the minimum font size is reached.

The default value for this property is `NO`. If you change it to `YES`, you should also set an appropriate minimum font size by modifying the `minimumFontSize` property.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property font` (page 232)
`@property minimumFontSize` (page 234)

Declared In

`UILabel.h`

baselineAdjustment

Controls how text baselines are adjusted when text needs to shrink to fit in the label.

```
@property(nonatomic) UIBaselineAdjustment baselineAdjustment
```

Discussion

If the `adjustsFontSizeToFit` property is set to `YES`, this property controls the behavior of the text baselines in situations where adjustment of the font size is required. The default value of this property is `UIBaselineAdjustmentAlignBaselines` (page 52).

Availability

Available in iPhone OS 2.0 and later.

See Also
`@property adjustsFontSizeToFit` (page 231)

Declared In
`UILabel.h`

enabled

The enabled state to use when drawing the label's text.

```
@property(nonatomic, getter=isEnabled) BOOL enabled
```

Discussion
This property determines only how the label is drawn. Disabled text is dimmed somewhat to indicate it is not active. This property is set to `YES` by default.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property adjustsFontSizeToFit` (page 231)

Declared In
`UILabel.h`

font

The font of the text.

```
@property(nonatomic, retain) UIFont *font
```

Discussion
This property applies to the entire text string. The default value for this property is `nil`, which results in text being drawing using the 17-point Helvetica plain font.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UILabel.h`

highlighted

A Boolean value indicating whether the receiver should be drawn with a highlight.

```
@property(nonatomic, getter=isHighlighted) BOOL highlighted
```

Discussion
Setting this property causes the receiver to redraw with the appropriate highlight state. A subclass implementing a text button might set this property to `YES` when the user presses the button and set it to `NO` at other times. In order for the highlight to be drawn, the `highlightedTextColor` property must contain a non-`nil` value.

The default value of this property is NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property highlightedTextColor (page 233)

Declared In
UILabel.h

highlightedTextColor

The highlight color applied to the label's text.

```
@property(nonatomic, retain) UIColor *highlightedTextColor
```

Discussion
Subclasses that use labels to implement a type of text button can use the value in this property when drawing the pressed state for the button. This color is applied to the label automatically whenever the highlighted property is set to YES.

The default value of this property is nil .

Availability
Available in iPhone OS 2.0 and later.

See Also
@property highlighted (page 232)

Declared In
UILabel.h

lineBreakMode

The technique to use for wrapping and truncating the label's text.

```
@property(nonatomic) UILineBreakMode lineBreakMode
```

Discussion
This property is in effect both during normal drawing and in cases where the font size must be reduced to fit the label's text in its bounding box. This property is set to UILineBreakModeTailTruncation (page 51) by default.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property adjustsFontSizeToFit (page 231)

Declared In
UILabel.h

minimumFontSize

The size of the smallest permissible font with which to draw the label's text.

```
@property(nonatomic) CGFloat minimumFontSize
```

Discussion

When drawing text that might not fit within the bounding rectangle of the label, you can use this property to prevent the receiver from reducing the font size to the point where it is no longer legible.

The default value for this property is 0.0. If you enable font adjustment for the label, you should always increase this value.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property adjustsFontSizeToFit` (page 231)

Declared In

`UILabel.h`

numberOfLines

The maximum number of lines to use for rendering text.

```
@property(nonatomic) NSInteger numberOfLines
```

Discussion

This property controls the maximum number of lines to use in order to fit the label's text into its bounding rectangle. The default value for this property is 1. To remove any maximum limit, and use as many lines as needed, set the value of this property to 0.

If you constrain your text using this property, any text that does not fit within the maximum number of lines and inside the bounding rectangle of the label is truncated using the appropriate line break mode.

When the receiver is resized using the `sizeToFit` (page 475) method, resizing takes into account the value stored in this property. For example, if this property is set to 3, the `sizeToFit` (page 475) method resizes the receiver so that it is big enough to display three lines of text.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property enabled` (page 232)

`@property adjustsFontSizeToFit` (page 231)

`- sizeToFit` (page 475) (UIView)

Declared In

`UILabel.h`

shadowColor

The shadow color of the text.

```
@property(nonatomic, retain) UIColor *shadowColor
```

Discussion

The default value for this property is `nil`, which indicates that no shadow is drawn. In addition to this property, you may also want to change the default shadow offset by modifying the `shadowOffset` property. Text shadows are drawn with the specified offset and color and no blurring.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property shadowOffset` (page 235)

Declared In

`UILabel.h`

shadowOffset

The shadow offset (measured in points) for the text.

```
@property(nonatomic) CGSize shadowOffset
```

Discussion

The shadow color must be non-`nil` for this property to have any effect. The default offset size is (0, -1), which indicates a shadow one point above the text. Text shadows are drawn with the specified offset and color and no blurring.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property shadowColor` (page 235)

Declared In

`UILabel.h`

text

The text displayed by the label.

```
@property(nonatomic, copy) NSString *text
```

Discussion

This string is `nil` by default.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UILabel.h`

textAlignment

The technique to use for aligning the text.

```
@property(nonatomic) UITextAlignment textAlignment
```

Discussion
This property applies to the entire text string. The default value of this property is `UITextAlignmentLeft`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UILabel.h`

textColor

The color of the text.

```
@property(nonatomic, retain) UIColor *textColor
```

Discussion
This property applies to the entire text string. The default value for this property is `nil`, which results in opaque black text.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UILabel.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
This property is inherited from the `UIView` parent class. This class changes the default value of this property to `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UILabel.h`

Instance Methods

drawTextInRect:

Draws the receiver's text (or its shadow) in the specified rectangle.

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

Parameters

rect

The rectangle in which to draw the text.

Discussion

You should not call this method directly. This method should only be overridden by subclasses that want to modify the default drawing behavior for the label's text.

By the time this method is called, the current graphics context is already configured with the default environment and text color for drawing. In your overridden method, you can configure the current context further and then invoke `super` to do the actual drawing or you can do the drawing yourself. If you do render the text yourself, you should not invoke `super`.

Note: In cases where the label draws its text with a shadow, this method may be called twice in succession to draw first the shadow and then the label text.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UILabel.h`

textRectForBounds:limitedToNumberOfLines:

Returns the drawing rectangle for the label's text.

```
- (CGRect)textRectForBounds:(CGRect)bounds
    limitedToNumberOfLines:(NSInteger)numberOfLines
```

Parameters

bounds

The bounding rectangle of the receiver.

numberOfLines

The maximum number of lines to use for the label. The value 0 indicates there is no maximum number of lines and that the rectangle should encompass all of the text.

Return Value

The computed drawing rectangle for the label's text.

Discussion

You should not call this method directly. This method should only be overridden by subclasses that want to change the receiver's bounding rectangle before performing any other computations. Use the value in the *numberOfLines* parameter to limit the height of the returned rectangle to the specified number of lines of text.

The default implementation of this method returns the original *bounds* rectangle.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UILabel.h`

UINavigationBar Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UINavigationBar.h

Overview

The `UINavigationBar` class implements a control for navigating hierarchical content. It's a bar, typically displayed at the top of the screen, containing buttons for navigating up and down a hierarchy. The primary properties are a left (back) button, a center title, and an optional right button. You can specify custom views for each of these.

The navigation bar represents only the bar at the top of the screen, not the view below. The view below the navigation bar usually lists children or properties of an object represented by the center title. For example, the user selects an object from a list to drill down in the hierarchy or see more details. It's the application's responsibility to implement this part of the behavior.

You create a navigation bar using the `alloc` and `init` methods. You then can specify the appearance using the `barStyle` (page 241) property.

A `UINavigationBar` object uses a stack to manage navigation items (instances of `UINavigationItem`) that represent a state of the navigation bar. You change the navigation bar by pushing navigation items using the `pushNavigationItem:animated:` (page 243) method or popping navigation items using the `popNavigationItemAnimated:` (page 243) method. Methods with an `animated:` argument allow you to animate the changes to the display.

You can also set items directly using the `items` (page 241) property, get the current item using the `topItem` (page 242) property, and get the previous item using the `backItem` (page 240) property.

You should set the delegate property to an object conforming to the `UINavigationBarDelegate` protocol. It is the responsibility of the delegate to update other views when items are pushed or popped from the stack—for example, it should display the previous view when the user clicks the back button.

Use the `UIBarButtonItem` class to create custom buttons for navigation items. Read *UINavigationItem Class Reference* for how to set custom views for the left, center, and right sides of a navigation bar.

Tasks

Configuring Navigation Bars

`barStyle` (page 241) *property*
The appearance of the navigation bar.

`tintColor` (page 242) *property*
The color used to tint the bar.

`delegate` (page 241) *property*
The navigation bar's delegate object.

Pushing and Popping Items

`- pushNavigationItem:animated:` (page 243)
Pushes the given navigation item onto the receiver's stack and updates the navigation bar.

`- popNavigationItemAnimated:` (page 243)
Pops the top item from the receiver's stack and updates the navigation bar.

`items` (page 241) *property*
An array of navigation items managed by the navigation bar.

`topItem` (page 242) *property*
The navigation item at the top of the navigation bar's stack. (read-only)

`backItem` (page 240) *property*
The navigation item that is next on the navigation bar's stack. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

backItem

The navigation item that is next on the navigation bar's stack. (read-only)

```
@property(nonatomic, readonly, retain) UINavigationItem *backItem
```

Discussion
If the top item doesn't have a left custom view, the back item's title is displayed on a back button on the left side of the navigation bar.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property topItem` (page 242)
`@property items` (page 241)

Declared In
`UINavigationBar.h`

barStyle

The appearance of the navigation bar.

```
@property(nonatomic, assign) UIBarStyle barStyle
```

Discussion
See `UIBarStyle` (page 625) for possible values. The default value is `UIBarStyleDefault`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationBar.h`

delegate

The navigation bar's delegate object.

```
@property(nonatomic, assign) id delegate
```

Discussion
The delegate should conform to the `UINavigationBarDelegate` protocol. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationBar.h`

items

An array of navigation items managed by the navigation bar.

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

Discussion
The bottom item is at index `0`, the back item is at index `n-2`, and the top item is at index `n-1`, where `n` is the number of items in the array.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property backItem` (page 240)
`@property topItem` (page 242)

Declared In
`UINavigationBar.h`

tintColor

The color used to tint the bar.

```
@property(nonatomic, retain) UIColor *tintColor
```

Discussion
The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationBar.h`

topItem

The navigation item at the top of the navigation bar's stack. (read-only)

```
@property(nonatomic, readonly, retain) UINavigationItem *topItem
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property backItem` (page 240)
`@property items` (page 241)

Declared In
`UINavigationBar.h`

Instance Methods

popNavigationItemAnimated:

Pops the top item from the receiver's stack and updates the navigation bar.

```
- (UINavigationItem *)popNavigationItemAnimated:(BOOL)animated
```

Parameters

animated

YES if the navigation bar should be animated; otherwise, NO.

Return Value

The top item that was popped.

Discussion

Popping a navigation item removes the top item from the stack and replaces it with the back item. The back item's title is centered on the navigation bar and its other properties are displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- pushNavigationItem:animated: (page 243)

Declared In

UINavigationBar.h

pushNavigationItem:animated:

Pushes the given navigation item onto the receiver's stack and updates the navigation bar.

```
- (void)pushNavigationItem:(UINavigationItem *)item animated:(BOOL)animated
```

Parameters

item

The navigation item to push on the stack.

animated

YES if the navigation bar should be animated; otherwise, NO.

Discussion

Pushing a navigation item displays the item's title in the center on the navigation bar. The previous top navigation item (if it exists) is displayed as a back button on the left side of the navigation bar. If the new top item has a left custom view, it is displayed instead of the back button.

Availability

Available in iPhone OS 2.0 and later.

See Also

- popNavigationItemAnimated: (page 243)

Declared In

UINavigationBar.h

UINavigationController Class Reference

Inherits from:	UIViewController : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UINavigationController.h
Companion guide:	View Controller Programming Guide for iPhone OS

Overview

The `UINavigationController` class implements a specialized controller for a navigation bar that manages all aspects of drilling down a hierarchy of objects. It uses application-supplied view controllers to configure the navigation bar and display views below it. Each view controller has a navigation item that provides information on how to set the navigation bar left, middle, and right views.

A navigation controller manages a stack of view controllers much as a navigation bar managing navigation bar items. You push and pop view controllers onto a navigation controller.

Since the `UINavigationController` class inherits from the `UIViewController` class, its instances also have a view property. A navigation controller's view contains a navigation bar and its top view controller's view in its hierarchy. Therefore, you just need to attach the navigation controller's view to a window to display it.

Although, navigation controller uses a navigation bar in its implementation, you should never need to, nor should you, access the navigation bar directly. You can, however, access the navigation items that represent the appearance of your view controllers.

You can also add navigation controllers to a toolbar controller. And even, present a navigation controller as a modal view on top of another view controller. In this case, the top view controller's view is displayed when the navigation controller is presented to the user, and its navigation bar covers any existing navigation bars.

When you pop and push view controllers, the navigation controller updates the navigation bar and view appropriately. At a minimum, the view controller sets the `title` (page 490) and `view` (page 491) properties.

The view should be a resizable view that can be attached to any view hierarchy. The navigation controller resizes and positions your views depending on where the navigation bar is and whether a toolbar is displayed. Therefore, set the `autoresizingMask` (page 446) properties of your views so they are resized appropriately when displayed by the navigation controller.

Read *View Controller Programming Guide for iPhone OS* to learn how to use this class.

This class is not intended to be subclassed.

Tasks

Creating Navigation Controllers

- `initWithRootViewController:` (page 249)
  Initializes and returns a newly created navigation controller.

Pushing and Popping Items

`topViewController` (page 248) *property*
The view controller at the top of the navigation controller's stack. (read-only)

`visibleViewController` (page 249) *property*
The view controller for the current visible view—that is, the modal view controller or the top view controller. (read-only)

`viewControllers` (page 248) *property*
The view controllers managed by the navigation controller.

- `pushViewController:animated:` (page 251)
  Pushes a view controller onto the receiver's stack and updates the display.
- `popViewControllerAnimated:` (page 251)
  Pops the top item from the receiver's stack and updates the display.
- `popToRootViewControllerAnimated:` (page 249)
  Pops all the view controllers on the stack except the root view controller and updates the display.
- `popToViewController:animated:` (page 250)
  Pops view controllers until the specified view controller is the top view controller and then updates the display.

Configuring Navigation Bars

`navigationBar` (page 247) *property*
The navigation bar managed by the navigation controller. (read-only)

`navigationBarHidden` (page 247) *property*
A Boolean value that determines whether the navigation bar is hidden.

`- setNavigationBarHidden:animated:` (page 252)
Sets whether the navigation bar is hidden.

Setting the Delegate

`delegate` (page 247) *property*
The receiver's delegate or `nil` if it doesn't have a delegate.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

delegate

The receiver's delegate or `nil` if it doesn't have a delegate.

```
@property(nonatomic, assign) id<UINavigationControllerDelegate> delegate
```

Discussion
See *UINavigationControllerDelegate Protocol Reference* for the methods this delegate should implement.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationController.h`

navigationBar

The navigation bar managed by the navigation controller. (read-only)

```
@property(nonatomic, readonly) UINavigationBar *navigationBar
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationController.h`

navigationBarHidden

A Boolean value that determines whether the navigation bar is hidden.

```
@property(nonatomic, getter=isNavigationBarHidden) BOOL navigationBarHidden
```

Discussion
If `YES`, the navigation bar is hidden. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setNavigationBarHidden:animated:` (page 252)

Declared In
`UINavigationController.h`

topViewController

The view controller at the top of the navigation controller's stack. (read-only)

```
@property(nonatomic, readonly, retain) UIViewController *topViewController
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property visibleViewController` (page 249)
`@property viewControllers` (page 248)

Declared In
`UINavigationController.h`

viewControllers

The view controllers managed by the navigation controller.

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

Discussion
The bottom view controller is at index `0` in the array, the back view controller is at index `n-2`, and the top controller is at index `n-1`, where `n` is the number of items in the array.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property topViewController` (page 248)
`@property visibleViewController` (page 249)

Declared In
`UINavigationController.h`

visibleViewController

The view controller for the current visible view—that is, the modal view controller or the top view controller. (read-only)

```
@property(nonatomic, readonly, retain) UIViewController *visibleViewController
```

Discussion

Use this property to get either the modal view controller if its view is visible; otherwise, get the top view controller.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property topViewController` (page 248)
`@property viewControllers` (page 248)

Declared In

`UINavigationController.h`

Instance Methods

initWithRootViewController:

Initializes and returns a newly created navigation controller.

```
- (id)initWithRootViewController:(UIViewController *)rootViewController
```

Parameters

rootViewController

The root view controller that is pushed on the stack with no animation. It cannot be an instance of tab bar controller.

Return Value

Newly initialized navigation controller.

Discussion

This is a convenience method for initializing the receiver and pushing a view controller on its stack.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UINavigationController.h`

popToRootViewControllerAnimated:

Pops all the view controllers on the stack except the root view controller and updates the display.

```
- (NSArray *)popToRootViewControllerAnimated:(BOOL)animated
```

Parameters

animated

Set this value to YES to animate the transition. Pass NO if you are setting up a navigation controller before its view is displayed.

Return Value

An array of view controllers that are popped from the stack.

Discussion

The root view controller becomes the top view controller.

Availability

Available in iPhone OS 2.0 and later.

See Also

- pushViewController:animated: (page 251)
- popViewControllerAnimated: (page 251)
- popToViewController:animated: (page 250)

Declared In

UINavigationController.h

popToViewController:animated:

Pops view controllers until the specified view controller is the top view controller and then updates the display.

```
- (NSArray *)popToViewController:(UIViewController *)viewController
    animated:(BOOL)animated
```

Parameters

viewController

The view controller that was popped from the stack.

animated

Set this value to YES to animate the transition. Pass NO if you are setting up a navigation controller before its view is displayed.

Return Value

An array of view controllers that were popped from the stack.

Availability

Available in iPhone OS 2.0 and later.

See Also

- pushViewController:animated: (page 251)
- popViewControllerAnimated: (page 251)
- popToRootViewControllerAnimated: (page 249)

Declared In

UINavigationController.h

popViewControllerAnimated:

Pops the top item from the receiver's stack and updates the display.

```
- (UIViewController *)popViewControllerAnimated:(BOOL)animated
```

Parameters

animated

Set this value to YES to animate the transition. Pass NO if you are setting up a navigation controller before its view is displayed.

Return Value

The view controller that was popped from the stack.

Discussion

This method removes the top view controller from the stack and makes the next controller the top view controller. It also replaces the view with the next view controller's view and updates the navigation bar accordingly. This method does nothing if the top view controller is the root view controller—you can't pop the last item on the stack.

Availability

Available in iPhone OS 2.0 and later.

See Also

- pushViewController:animated: (page 251)
- popToRootViewControllerAnimated: (page 249)
- popToViewController:animated: (page 250)

Declared In

UINavigationController.h

pushViewController:animated:

Pushes a view controller onto the receiver's stack and updates the display.

```
- (void)pushViewController:(UIViewController *)viewController animated:(BOOL)animated
```

Parameters

viewController

The view controller that is pushed onto the stack. It cannot be an instance of tab bar controller. This method does nothing if the view controller is already on the stack.

animated

Set this value to YES to animate the transition. Pass NO if you are setting up a navigation controller before its view is displayed.

Discussion

The view controller pushed onto the stack becomes the top view controller. Pushing a view controller displays the view it manages by replacing the top view controller's view in the hierarchy and updates the navigation bar accordingly. Views are always resized to fit between the navigation bar and toolbar if present before displaying.

The navigation controller updates the left side of the navigation bar as follows:

- The next view controller (if it exists) is represented as a back button on the left side of the navigation bar unless another view is specified. The back button pops the current view controller.

The `title` (page 490) property is used as the default button title. Set the `backBarButtonItem` (page 256) property of the next view controller's `navigationItem` (page 488) property to specify an alternate title or button.

- If a custom left view is specified, it is displayed instead of a back button.

  Set the `leftBarButtonItem` (page 257) property of the `navigationItem` (page 488) property to specify a custom left view. Use the `UIBarButtonItem` class to create a custom bar button item for the navigation item. Use the `initWithBarButtonSystemItem:target:action:` (page 110) method to create common system buttons.

The navigation controller updates the middle of the navigation bar as follows:

- The view controller's title is displayed in the middle of the navigation bar unless it has a custom title view.

  Set the view controller title using the `title` (page 490) property or if different, set the navigation item's `title` property.

- If a custom title view is specified, it is displayed in the middle of the navigation bar instead of the title.

  Set the `titleView` (page 259) property of the `navigationItem` (page 488) property to specify a custom title view.

The navigation controller updates the right side of the navigation bar as follows:

- If a custom right view is specified, it is displayed on the right side of the navigation bar; otherwise, nothing is displayed on the right side.

  Set the `rightBarButtonItem` (page 258) property of the `navigationItem` (page 488) property to specify a custom right view. Use the `UIBarButtonItem` class to create a bar button item for the navigation item. Use the `initWithBarButtonSystemItem:target:action:` (page 110) method of `UIBarButtonItem` to create common system buttons.

The navigation controller updates the navigation bar in the same manner each time the top view controller changes—for example, when a view controller is popped from the stack and the next view controller becomes the top view controller, the navigation bar is updated accordingly.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- popViewControllerAnimated:` (page 251)
- `- popToRootViewControllerAnimated:` (page 249)
- `- popToViewController:animated:` (page 250)

Declared In
`UINavigationController.h`

setNavigationBarHidden:animated:

Sets whether the navigation bar is hidden.

```
- (void)setNavigationBarHidden:(BOOL)hidden animated:(BOOL)animated
```

Parameters

hidden

YES if the navigation bar should be hidden; otherwise, NO.

animated

YES if the transition should be animated using "Setting the Delegate" (page 247) as the duration; otherwise, NO.

Availability

Available in iPhone OS 2.0 and later.

See Also

@property navigationBarHidden (page 247)

Declared In

UINavigationController.h

Constants

UINavigationControllerHideShowBarDuration

A global variable for specifying the duration when animating the navigation bar.

```
extern CGFloat UINavigationControllerHideShowBarDuration
```

Constants

UINavigationControllerHideShowBarDuration

Global variable for specifying the duration when animating the navigation bar.

Available in iPhone OS 2.0 and later.

Declared in UINavigationController.h

UINavigationItem Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UINavigationBar.h

Overview

The `UINavigationItem` class encapsulates information about a navigation item pushed on a `UINavigationBar` object's stack. A navigation bar is a control used to navigate hierarchical content. A `UINavigationItem` specifies what is displayed on the navigation bar when it is the top item and also how it is represented when it is the back item.

Use the `initWithTitle:` (page 259) method to create a navigation item specifying the item's title. The item cannot be represented on the navigation bar without a title. Use the `backBarButtonItem` (page 256) property if you want to use a different title when this item is the back item. The `backBarButtonItem` (page 256) property is displayed as the back button unless a custom left view is specified.

The navigation bar displays a back button on the left and the title in the center by default. You can change this behavior by specifying either a custom left, center, or right view. Use the `setLeftBarButtonItem:animated:` (page 260) and `setRightBarButtonItem:animated:` (page 261) methods to change the left and right views; you can specify that the change be animated. Use the `titleView` (page 259) method to change the center view to a custom view.

These custom views can be system buttons. Use the `UIBarButtonItem` class to create custom views to add to navigation items.

Tasks

Initializing an Item

- `initWithTitle:` (page 259)
  Returns a navigation item initialized with the specified title.

Getting and Setting Properties

`title` (page 258) *property*
The navigation item's title displayed in the center of the navigation bar.

`prompt` (page 258) *property*
A single line of text displayed at the top of the navigation bar.

`backBarButtonItem` (page 256) *property*
The bar button item to use when this item is represented by a back button on the navigation bar.

`hidesBackButton` (page 257) *property*
A Boolean value that determines whether the back button is hidden.

- `setHidesBackButton:animated:` (page 260)
  Sets whether the back button is hidden, optionally animating the transition.

Customizing Views

`titleView` (page 259) *property*
A custom view displayed in the center of the navigation bar when this item is the top item.

`leftBarButtonItem` (page 257) *property*
A custom bar item displayed on the left of the navigation bar when this item is the top item.

`rightBarButtonItem` (page 258) *property*
A custom bar item displayed on the right of the navigation bar when this item is the top item.

- `setLeftBarButtonItem:animated:` (page 260)
  Sets the custom bar item, optionally animating the transition to the view.
- `setRightBarButtonItem:animated:` (page 261)
  Sets the custom bar item, optionally animating the transition to the view.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

backBarButtonItem

The bar button item to use when this item is represented by a back button on the navigation bar.

```
@property(nonatomic, retain) UIBarButtonItem *backBarButtonItem
```

Discussion
When this item is the back item of the navigation bar—when it is the next item below the top item—it may be represented as a back button on the navigation bar. Use this property to specify the back button. The target and action of the back bar button item you set should be `nil`. The default value is a bar button item displaying the navigation item's title.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property backItem` (page 240)
`@property hidesBackButton` (page 257)
`- setHidesBackButton:animated:` (page 260)

Declared In
`UINavigationBar.h`

hidesBackButton

A Boolean value that determines whether the back button is hidden.

```
@property(nonatomic, assign) BOOL hidesBackButton
```

Discussion
`YES` if the back button is hidden when this navigation item is the top item; otherwise, `NO`. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property backItem` (page 240)
`@property backBarButtonItem` (page 256)
`- setHidesBackButton:animated:` (page 260)

Declared In
`UINavigationBar.h`

leftBarButtonItem

A custom bar item displayed on the left of the navigation bar when this item is the top item.

```
@property(nonatomic, retain) UIBarButtonItem *leftBarButtonItem
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property rightBarButtonItem` (page 258)
`- setLeftBarButtonItem:animated:` (page 260)

- setRightBarButtonItem:animated: (page 261)

Declared In
UINavigationBar.h

prompt

A single line of text displayed at the top of the navigation bar.

```
@property(nonatomic, copy) NSString *prompt
```

Discussion
The default value is nil.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UINavigationBar.h

rightBarButtonItem

A custom bar item displayed on the right of the navigation bar when this item is the top item.

```
@property(nonatomic, retain) UIBarButtonItem *rightBarButtonItem
```

Availability
Available in iPhone OS 2.0 and later.

See Also
@property leftBarButtonItem (page 257)
- setLeftBarButtonItem:animated: (page 260)
- setRightBarButtonItem:animated: (page 261)

Declared In
UINavigationBar.h

title

The navigation item's title displayed in the center of the navigation bar.

```
@property(nonatomic, copy) NSString *title
```

Discussion
The default value is nil.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithTitle: (page 259)
UINavigationItem (page 255)

@property titleView (page 259)

Declared In
UINavigationBar.h

titleView

A custom view displayed in the center of the navigation bar when this item is the top item.

```
@property(nonatomic, retain) UIView *titleView
```

Discussion
If this property value is nil, the navigation item's title is displayed in the center of the navigation bar when this item is the top item. If you set this property to a custom title, it is displayed instead of the title. This property is ignored if leftBarButtonItem (page 257) is not nil.

Custom views can contain buttons. Use the buttonWithType: (page 129) method in UIButton class to add buttons to your custom view in the style of the navigation bar. Custom title views are centered on the navigation bar and may be resized to fit.

The default value is nil.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UINavigationBar.h

Instance Methods

initWithTitle:

Returns a navigation item initialized with the specified title.

```
- (id)initWithTitle:(NSString *)title
```

Parameters
title
The string to set as the navigation item's title displayed in the center of the navigation bar.

Return Value
A new UINavigationItem object initialized with the specified title.

Discussion
This is the designated initializer for this class.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property title (page 258)

Declared In
UINavigationBar.h

setHidesBackButton:animated:

Sets whether the back button is hidden, optionally animating the transition.

```
- (void)setHidesBackButton:(BOOL)hidesBackButton animated:(BOOL)animated
```

Parameters

hidesBackButton
YES if the back button is hidden when this navigation item is the top item; otherwise, NO.

animated
YES to animate the transition; otherwise, NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property backItem (page 240)
@property backBarButtonItem (page 256)
@property hidesBackButton (page 257)

Declared In
UINavigationBar.h

setLeftBarButtonItem:animated:

Sets the custom bar item, optionally animating the transition to the view.

```
- (void)setLeftBarButtonItem:(UIBarButtonItem *)item animated:(BOOL)animated
```

Parameters

item
A custom bar item to display on the left of the navigation bar.

animated
YES to animate the transition to the custom bar item when this item becomes the top item; otherwise, NO.

Discussion
If two navigation items have the same custom left or right bar items, the views remain stationary during the transition when an item is pushed or popped.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property leftBarButtonItem (page 257)
@property rightBarButtonItem (page 258)
- setRightBarButtonItem:animated: (page 261)

Declared In
UINavigationBar.h

setRightBarButtonItem:animated:

Sets the custom bar item, optionally animating the transition to the view.

```
- (void)setRightBarButtonItem:(UIBarButtonItem *)item animated:(BOOL)animated
```

Parameters

item

A custom bar item to display on the right of the navigation bar.

animated

YES to animate the transition to the custom bar item when this item becomes the top item; otherwise, NO.

Discussion
If two navigation items have the same custom left or right bar items, the bar items remain stationary during the transition when an item is pushed or popped.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property leftBarButtonItem (page 257)
@property rightBarButtonItem (page 258)
- setLeftBarButtonItem:animated: (page 260)

Declared In
UINavigationBar.h

UIPageControl Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIPageControl.h

Overview

You use the `UIPageControl` class to create and manage page controls. A page control is a succession of dots centered in the control. Each dot corresponds to a page in the application's document (or other data-model entity), with the white dot indicating the currently viewed page.

For an example of a page control, see the Weather application (with a number of locations configured) or Safari (with a number of tab views set).

When a user taps a page control to move to the next or previous page, the control sends the `UIControlEventValueChanged` (page 170) event for handling by the delegate. The delegate can then evaluate the `currentPage` (page 264) property to determine the page to display. The page control advances only one page in either direction.

Note: Because of physical factors—namely the size of the device screen and the size and layout of the page indicators—there is a limit of about 20 page indicators on the screen before they are clipped.

Tasks

Managing the Page Navigation

`currentPage` (page 264) *property*
The current page, shown by the receiver as a white dot.

`numberOfPages` (page 265) *property*
The number of pages the receiver shows (as dots).

`hidesForSinglePage` (page 265) *property*
A Boolean value that controls whether the page indicator is hidden when there is only one page.

Updating the Page Display

`defersCurrentPageDisplay` (page 265) *property*
A Boolean value that controls when the current page is displayed.

- `updateCurrentPageDisplay` (page 266)
Updates the page indicator to the current page.

Resizing the Control

- `sizeForNumberOfPages:` (page 266)
Returns the size the receiver's bounds should be to accommodate the given number of pages.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

currentPage

The current page, shown by the receiver as a white dot.

```
@property(nonatomic) NSInteger currentPage
```

Discussion

The property value is an integer specifying the current page shown minus one; thus a value of zero (the default) indicates the first page. A page control shows the current page as a white dot. Values outside the possible range are pinned to either 0 or `numberOfPages` (page 265) minus 1.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIPageControl.h`

defersCurrentPageDisplay

A Boolean value that controls when the current page is displayed.

```
@property(nonatomic) BOOL defersCurrentPageDisplay
```

Discussion
Set the value of this property to `YES` so that, when the user clicks the control to go to a new page, the class defers updating the page indicator until it calls `updatePageIndicator` (page 266). Set the value to `NO` (the default) to have the page indicator updated immediately.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIPageControl.h`

hidesForSinglePage

A Boolean value that controls whether the page indicator is hidden when there is only one page.

```
@property(nonatomic) BOOL hidesForSinglePage
```

Discussion
Assign a value of `YES` to hide the page indicator when there is only one page; assign `NO` (the default) to show the page indicator if there is only one page.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIPageControl.h`

numberOfPages

The number of pages the receiver shows (as dots).

```
@property(nonatomic) NSInteger numberOfPages
```

Discussion
The value of the property is the number of pages for the page control to show as dots. The default value is 0.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIPageControl.h

Instance Methods

sizeForNumberOfPages:

Returns the size the receiver's bounds should be to accommodate the given number of pages.

```
- (CGSize)sizeForNumberOfPages:(NSInteger)pageCount
```

Parameters

pageCount
The number of pages to fit in the receiver's bounds.

Return Value
The minimum size required to display dots for the page count.

Discussion
Subclasses that customize the appearance of the page control can use this method to resize the page control when the page count changes.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIPageControl.h

updateCurrentPageDisplay

Updates the page indicator to the current page.

```
- (void)updateCurrentPageDisplay
```

Discussion
This method updates the page indicator so that the current page (the white dot) matches the value returned from `currentPage` (page 264). The class ignores this method if the value of `defersPageIndicatorUpdate` (page 265) is `NO`. Setting the `currentPage` value directly updates the indicator immediately.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIPageControl.h

UIPickerView Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIPickerView.h

Overview

The `UIPickerView` class implements objects that use a spinning-wheel or slot-machine motif to show one or more sets of values. Users select values by rotating the wheels until a row of values is aligned with a selection indicator.

The `UIDatePicker` class uses a custom subclass of `UIPickerView` to display dates and times. To see an example, tap the add ("+") button in the the Alarm pane of the Clock application.

A `UIPickerView` object is a potentially multidimensional user-interface element consisting of rows and components. A component is a wheel, which has a series of items (rows) at indexed locations on the wheel. Each component also has an indexed location (left to right) in a picker view. Each row on a component has content, which is either a string or a view object such as a label (`UILabel`) or an image (`UIImageView`).

A `UIPickerView` object requires the cooperation of a delegate for constructing its components and a data source for providing the number of rows and components. The delegate must adopt the `UIPickerViewDelegate` protocol and implement the required methods to return the drawing rectangle for rows in each component. It also provides the content for each component's row, either as a string or a view, and it typically responds to new selections or deselections. The data source must adopt the `UIPickerViewDataSource` protocol and implement the required methods to return number of components and the number of rows in each component.

You can dynamically change the rows of one or all components by calling the `reloadComponent:` (page 271) or `reloadAllComponents` (page 271) methods, respectively. These methods cause `UIPickerView` to request the delegate to provide new component and row data. You reload a picker view when a

selected value in one component changes the set of values in another component; for example, changing a row value from February to March in one component would change a related component representing the days of the month.

Tasks

Getting the Dimensions of the View Picker

`numberOfComponents` (page 269) *property*
Returns the number of components displayed by the receiver. (read-only)

- `numberOfRowsInComponent:` (page 270)
Returns the number of rows for a component.

- `rowSizeForComponent:` (page 271)
Returns the size of a row for a component.

Reloading the View Picker

- `reloadAllComponents` (page 271)
Reloads all components of the receiver.

- `reloadComponent:` (page 271)
Reloads a particular component of the receiver.

Selecting Rows in the View Picker

- `selectRow:inComponent:animated:` (page 272)
Selects a row in a specified component of the receiver.

- `selectedRowInComponent:` (page 272)
Returns the index of the selected row in a given component.

Returning the View for a Row and Component

- `viewForRow:forComponent:` (page 273)
Returns the view used by the receiver for a given row and component.

Managing the Delegate

`delegate` (page 269) *property*
The delegate for the receiver.

Managing the Data Source

`dataSource` (page 269) *property*
The data source for the receiver.

Managing the Appearance of the Picker View

`showsSelectionIndicator` (page 270) *property*
A Boolean value that determines whether the selection indicator is displayed.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

dataSource

The data source for the receiver.

```
@property(nonatomic, assign) id<UIPickerViewDataSource> dataSource
```

Discussion
The data source must adopt the `UIPickerViewDataSource` protocol.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIPickerView.h`

delegate

The delegate for the receiver.

```
@property(nonatomic, assign) id<UIPickerViewDelegate> delegate
```

Discussion
The delegate must adopt the `UIPickerViewDelegate` protocol.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIPickerView.h`

numberOfComponents

Returns the number of components displayed by the receiver. (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfComponents
```

Discussion
`UIPickerView` fetches the value of this property from the delegate and and caches it. The default value is zero.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `numberOfRowsInComponent:` (page 270)
- `rowSizeForComponent:` (page 271)

Declared In
`UIPickerView.h`

showsSelectionIndicator

A Boolean value that determines whether the selection indicator is displayed.

```
@property(nonatomic) BOOL showsSelectionIndicator
```

Discussion
If the value of the property is YES, the picker view shows a clear overlay across the current row. The default value of this property is `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIPickerView.h`

Instance Methods

numberOfRowsInComponent:

Returns the number of rows for a component.

```
- (NSInteger)numberOfRowsInComponent:(NSInteger)component
```

Parameters
component
An index number identifying a component.

Return Value
The number of rows in the given component.

Discussion
`UIPickerView` fetches the value of this property from the delegate and and caches it. The default value is zero.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property numberOfComponents` (page 269)
`- rowSizeForComponent:` (page 271)

Declared In
`UIPickerView.h`

reloadAllComponents

Reloads all components of the receiver.

```
- (void)reloadAllComponents
```

Discussion
Calling this method causes the receiver to query the delegate for new data for all components.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- reloadComponent:` (page 271)

Declared In
`UIPickerView.h`

reloadComponent:

Reloads a particular component of the receiver.

```
- (void)reloadComponent:(NSInteger)component
```

Parameters
component
An index number identifying a component of the receiver.

Discussion
Calling this method causes the receiver to query the delegate for new data for the given component.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- reloadAllComponents` (page 271)

Declared In
`UIPickerView.h`

rowSizeForComponent:

Returns the size of a row for a component.

```
- (CGSize)rowSizeForComponent:(NSInteger)component
```

Parameters

component

An index number identifying a component.

Return Value

The size of rows in the given component. This is generally the size required to display the largest string or view used as a row in the component.

Discussion

`UIPickerView` fetches the value of this property from the delegate and and caches it. The default value is zero.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property numberOfComponents` (page 269)

`- numberOfRowsInComponent:` (page 270)

Declared In

`UIPickerView.h`

selectedRowInComponent:

Returns the index of the selected row in a given component.

```
- (NSInteger)selectedRowInComponent:(NSInteger)component
```

Parameters

component

An index number identifying a component of the picker view.

Return Value

An index number identifying the selected row, or `-1` if no row is selected.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- selectRow:inComponent:animated:` (page 272)

Declared In

`UIPickerView.h`

selectRow:inComponent:animated:

Selects a row in a specified component of the receiver.

```
- (void)selectRow:(NSInteger)row inComponent:(NSInteger)component
    animated:(BOOL)animated
```

Parameters

row

An index number identifying a row of *component*.

component

An index number identifying a component of the picker view.

animated

YES to animate the selection by spinning the wheel (component) to the new value; if you specify NO, the new selection is shown immediately.

Availability

Available in iPhone OS 2.0 and later.

See Also

- selectedRowInComponent: (page 272)

Declared In

UIPickerView.h

viewForRow:forComponent:

Returns the view used by the receiver for a given row and component.

```
- (UIView *)viewForRow:(NSInteger)row forComponent:(NSInteger)component
```

Parameters

row

The index number of a row of the receiver.

component

The index number of a component of the receiver.

Return Value

The view provided by the delegate in the pickerView:viewForRow:forComponent:reusingView: (page 559) method. Returns nil if the specified row of the component is not visible or if the delegate does not implement pickerView:viewForRow:forComponent:reusingView:.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIPickerView.h

UIProgressView Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIProgressView.h

Overview

You use the `UIProgressView` class to depict the progress of a task over time. An example of a progress bar is the one shown at the bottom of the Mail application when it's downloading messages.

The `UIProgressView` class provides properties for managing the style of the progress bar and for getting and setting values that are pinned to the progress of a task.

For an indeterminate progress indicator—or, informally, a "spinner"—use an instance of the `UIActivityIndicatorView` class.

Tasks

Initializing the UIProgressView Object

`- initWithProgressViewStyle:` (page 277)
Initializes and returns an progress-view object.

Managing the Progress Bar

`progress` (page 276) *property*
The current progress shown by the receiver.

Configuring the Bar Style

`progressViewStyle` (page 276) *property*
The current graphical style of the receiver.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

progress

The current progress shown by the receiver.

```
@property(nonatomic) float progress
```

Discussion
The current progress is represented by a floating-point value between 0.0 and 1.0, inclusive, where 1.0 indicates the completion of the task. The default value is 0.0. Values less than 0.0 and greater than 1.0 are pinned to those limits.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIProgressView.h`

progressViewStyle

The current graphical style of the receiver.

```
@property(nonatomic) UIProgressViewStyle progressViewStyle
```

Discussion
The value of this property is a constant that specifies the style of the progress view. The default style is `UIProgressViewStyleDefault` (page 277). For more on these constants, see `UIProgressViewStyle` (page 277).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIProgressView.h`

Instance Methods

initWithProgressViewStyle:

Initializes and returns an progress-view object.

```
- (id)initWithProgressViewStyle:(UIProgressViewStyle)style
```

Parameters

style

A constant that specifies the style of the object to be created. See `UIProgressViewStyle` (page 277) for descriptions of the style constants.

Return Value

An initialized `UIProgressView` object or `nil` if the object couldn't be created.

Discussion

`UIProgressView` sets the height of the returned view according to the specified *style*. You can set and retrieve the style of a progress view through the `progressViewStyle` (page 276) property.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIProgressView.h`

Constants

UIProgressViewStyle

The styles permitted for the progress bar.

```
typedef enum {
    UIProgressViewStyleDefault,
    UIProgressViewStyleBar,
} UIProgressViewStyle;
```

Constants

`UIProgressViewStyleDefault`

The standard progress-view style. This is the default.

Available in iPhone OS 2.0 and later.

Declared in `UIProgressView.h`

`UIProgressViewStyleBar`

The style of progress view that is used in a toolbar.

Available in iPhone OS 2.0 and later.

Declared in `UIProgressView.h`

Discussion
You can set and retrieve the current style of progress view through the `progressViewStyle` (page 276) property.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIProgressView.h`

UIResponder Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIResponder.h

Overview

The `UIResponder` class defines an interface for objects that respond to and handle events. It is the superclass of `UIApplication`, `UIView` and its subclasses (which include `UIWindow`). Instances of these classes are sometimes referred to as responder objects.

The primary event-handling methods are `touchesBegan:withEvent:` (page 283), `touchesMoved:withEvent:` (page 285), `touchesEnded:withEvent:` (page 284), and `touchesCancelled:withEvent:` (page 283). The parameters of these methods associate touches with their events—especially touches that are new or have changed—and thus allow responder objects to track and handle the touches as the delivered events progress through the phases of a multi-touch sequence. Any time a finger touches the screen, is dragged on the screen, or lifts from the screen, a `UIEvent` object is generated. The event object contains `UITouch` objects for all fingers on the screen or just lifted from it.

See Event Handling in *iPhone OS Programming Guide* for further information on event handling.

Tasks

Managing the Responder Chain

`- nextResponder` (page 282)
Returns the receiver's next responder, or `nil` if it has none.

- `isFirstResponder` (page 282)
  Returns a Boolean value indicating whether the receiver is the first responder.
- `canBecomeFirstResponder` (page 281)
  Returns a Boolean value indicating whether the receiver can become first responder.
- `becomeFirstResponder` (page 280)
  Notifies the receiver that it's about to become first responder in its window.
- `canResignFirstResponder` (page 281)
  Returns a Boolean value indicating whether the receiver is willing to relinquish first-responder status.
- `resignFirstResponder` (page 282)
  Notifies the receiver that it's been asked to relinquish its status as first responder in its window.

Responding to Events

- `touchesBegan:withEvent:` (page 283)
  Tells the receiver when one or more fingers touch down in a view or window.
- `touchesMoved:withEvent:` (page 285)
  Tells the receiver when one or more fingers associated with an event move within a view or window.
- `touchesEnded:withEvent:` (page 284)
  Tells the receiver when one or more fingers are raised from a view or window.
- `touchesCancelled:withEvent:` (page 283)
  Sent to the receiver when a system event (such as a low-memory warning) cancels an event.

Instance Methods

becomeFirstResponder

Notifies the receiver that it's about to become first responder in its window.

```
- (BOOL)becomeFirstResponder
```

Discussion

The default implementation returns `YES`, accepting first responder status. Subclasses can override this method to update state or perform some action such as highlighting the selection, or to return `NO`, refusing first responder status.

Sending this message to a view object, for example a `UITextField` instance, will cause it to become the first responder and begin editing.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isFirstResponder` (page 282)
- `canBecomeFirstResponder` (page 281)

Declared In
UIResponder.h

canBecomeFirstResponder

Returns a Boolean value indicating whether the receiver can become first responder.

```
- (BOOL)canBecomeFirstResponder
```

Return Value
YES if the receiver can become the first responder, NO otherwise.

Discussion
If a responder object returns YES from this method, it can thereafter join the responder chain and receive touch events and action messages. Subclasses must override this method to be able to become first responder.

You must not send this message to a view that is not currently attached to the view hierarchy. The result is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- becomeFirstResponder (page 280)

Declared In
UIResponder.h

canResignFirstResponder

Returns a Boolean value indicating whether the receiver is willing to relinquish first-responder status.

```
- (BOOL)canResignFirstResponder
```

Return Value
YES if the receiver can resign first-responder status, NO otherwise.

Discussion
As an example, a text field in the middle of editing might want to implement this method to return NO to force the user to finish editing.

Availability
Available in iPhone OS 2.0 and later.

See Also
- resignFirstResponder (page 282)

Declared In
UIResponder.h

isFirstResponder

Returns a Boolean value indicating whether the receiver is the first responder.

```
- (BOOL)isFirstResponder
```

Return Value

`YES` if the receiver is the first responder, `NO` otherwise.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `becomeFirstResponder` (page 280)
- `resignFirstResponder` (page 282)

Declared In

`UIResponder.h`

nextResponder

Returns the receiver's next responder, or `nil` if it has none.

```
- (UIResponder *)nextResponder
```

Return Value

The next object in the responder chain to be presented with an event for handling.

Discussion

The `UIResponder` class does not store or set the next responder automatically, instead returning `nil` by default. Subclasses must override this method to set the next responder. `UIView` implements this method by returning the superview,`UIWindow` returns the application object, and `UIApplication` returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isFirstResponder` (page 282)

Declared In

`UIResponder.h`

resignFirstResponder

Notifies the receiver that it's been asked to relinquish its status as first responder in its window.

```
- (BOOL)resignFirstResponder
```

Discussion

The default implementation returns `YES`, resigning first responder status. Subclasses can override this method to update state or perform some action such as unhighlighting the selection, or to return `NO`, refusing to relinquish first responder status.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `isFirstResponder` (page 282)
- `canResignFirstResponder` (page 281)

Declared In
`UIResponder.h`

touchesBegan:withEvent:

Tells the receiver when one or more fingers touch down in a view or window.

```
- (void)touchesBegan:(NSSet *)touches
    withEvent:(UIEvent *)event
```

Parameters

touches
An set of `UITouch` instances that represent the touches for the starting phase of the event represented by *event*.

event
An object representing the event to which the touches belong.

Discussion
The default implementation of this method does nothing.

Multiple touches are disabled by default. In order to receive multiple touch events you must send a `setMultipleTouchEnabled:` message to the corresponding view instance, passing `YES` as the parameter.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `touchesMoved:withEvent:` (page 285)
- `touchesEnded:withEvent:` (page 284)
- `touchesCancelled:withEvent:` (page 283)

Declared In
`UIResponder.h`

touchesCancelled:withEvent:

Sent to the receiver when a system event (such as a low-memory warning) cancels an event.

```
- (void)touchesCancelled:(NSSet *)touches
    withEvent:(UIEvent *)event
```

Parameters

touches

An set of `UITouch` instances that represent the touches for the ending phase of the event represented by *event*.

event

An object representing the event to which the touches belong.

Discussion

This method is invoked when the Cocoa Touch framework receives a system event requiring cancellation of the user event; for this, it generates a `UITouch` object with a phase of `UITouchPhaseCancel`.

When an object receives a `touchesCancelled:withEvent:` message it should clean up any state information that was established in its `touchesBegan:withEvent:` implementation.

The default implementation of this method does nothing.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIResponder.h`

touchesEnded:withEvent:

Tells the receiver when one or more fingers are raised from a view or window.

```
- (void)touchesEnded:(NSSet *)touches
    withEvent:(UIEvent *)event
```

Parameters

touches

An set of `UITouch` instances that represent the touches for the ending phase of the event represented by *event*.

event

An object representing the event to which the touches belong.

Discussion

The default implementation of this method does nothing.

When an object receives a `touchesEnded:withEvent:` message it should clean up any state information that was established in its `touchesBegan:withEvent:` implementation.

Multiple touches are disabled by default. In order to receive multiple touch events you must send a `setMultipleTouchEnabled:` message to the corresponding view instance, passing `YES` as the parameter.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `touchesBegan:withEvent:` (page 283)
- `touchesMoved:withEvent:` (page 285)
- `touchesCancelled:withEvent:` (page 283)

Declared In
UIResponder.h

touchesMoved:withEvent:

Tells the receiver when one or more fingers associated with an event move within a view or window.

```
- (void)touchesMoved:(NSSet *)touches
    withEvent:(UIEvent *)event
```

Parameters

touches

An set of UITouch instances that represent the touches that are moving during the event represented by *event*.

event

An object representing the event to which the touches belong.

Discussion
The default implementation of this method does nothing.

Multiple touches are disabled by default. In order to receive multiple touch events you must send a setMultipleTouchEnabled: message to the corresponding view instance, passing YES as the parameter.

Availability
Available in iPhone OS 2.0 and later.

See Also
- touchesBegan:withEvent: (page 283)
- touchesEnded:withEvent: (page 284)
- touchesCancelled:withEvent: (page 283)

Declared In
UIResponder.h

UIScreen Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIScreen.h

Overview

A `UIScreen` object contains the bounding rectangle of the device's entire screen. When setting up your application's user interface, you should use the properties of this object to get the recommended frame rectangles for your application's window.

Tasks

Getting the Available Screens

`+ mainScreen` (page 288)
Returns the screen object representing the device's screen.

Getting the Bounds Information

`bounds` (page 288) *property*
Contains the bounding rectangle of the screen, measured in points. (read-only)

`applicationFrame` (page 288) *property*
The frame rectangle to use for your application's window. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

applicationFrame

The frame rectangle to use for your application's window. (read-only)

```
@property(nonatomic, readonly) CGRect applicationFrame
```

Discussion
This property contains the screen bounds minus the area occupied by the status bar, if it is visible. Using this property is the recommended way to retrieve your application's initial window size. The rectangle is specified in points.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIScreen.h

bounds

Contains the bounding rectangle of the screen, measured in points. (read-only)

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

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIScreen.h

Class Methods

mainScreen

Returns the screen object representing the device's screen.

```
+ (UIScreen *)mainScreen
```

Return Value
The screen object for the device

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIScreen.h

UIScrollView Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIScrollView.h

Overview

`UIScrollView` is the base class for any class that needs to display content that is larger than the size of the application's window. It enables users to scroll within that content by making swiping gestures, and to zoom in and back from portions of the content by making pinching gestures.

`UIScrollView` is the superclass of several UIKit classes including `UITableView` and `UITextView`.

The central notion of a `UIScrollView` object (or, simply, a scroll view) is that it is a view whose origin is adjustable over the content view. It clips the content to its frame, which generally (but not necessarily) coincides that of the application's main window. A scroll view tracks the movements of fingers and adjusts the origin accordingly. The view that is showing its content "through" the scroll view draws that portion of itself based on the new origin, which is pinned to an offset in the content view. The scroll view itself does no drawing except for displaying vertical and horizontal scroll indicators. The scroll view must know the size of the content view so it knows when to stop scrolling; by default, it "bounces" back when scrolling exceeds the bounds of the content.

The object that manages the drawing of content displayed in a scroll view should tile the content's subviews so that no view exceeds the size of the screen. As users scroll in the scroll view, this object should add and remove subviews as necessary.

Because a scroll view has no scroll bars, it must know whether a touch signals an intent to scroll versus an intent to track a subview in the content. To make this determination, it temporarily intercepts a touch-down event by starting a timer and, before the timer fires, seeing if the touching finger makes any movement. If the time fires without a significant change in position, the scroll view sends tracking events to the touched subview of the content view. If the user then drags his or her finger far enough before the timer elapses, the scroll view cancels any tracking in the subview and performs the scrolling

itself. Subclasses can override the `touchesShouldBegin:withEvent:inContentView:` (page 302), `pagingEnabled` (page 298), and `touchesShouldCancelInContentView:` (page 303) methods (which are called by the scroll view) to affect how the scroll view handles scrolling gestures.

A scroll view also handles zooming and panning of content. As the user makes a pinch-in or pinch-out gesture, the scroll view adjusts the offset and the scale of the content. When the gesture ends, the object managing the content view should should update subviews of the content as necessary. (Note that the gesture can end and a finger could still be down.) While the gesture is in progress, the scroll view does not send any tracking calls to the subview.

The `UIScrollView` class can have a delegate that must adopt the `UIScrollViewDelegate` protocol. For zooming and panning to work, the delegate must implement both `viewForZoomingInScrollView:` (page 566) and `scrollViewDidEndZooming:withView:atScale:` (page 563); in addition, the maximum (`maximumZoomScale` (page 297)) and minimum (`minimumZoomScale` (page 297)) zoom scale must be different.

Tasks

Managing the Display of Content

- `setContentOffset:animated:` (page 302)
  Sets the offset from the content view's origin that corresponds to the receiver's origin.

`contentOffset` (page 294) *property*
The point at which the origin of the content view is offset from the origin of the scroll view.

`contentSize` (page 295) *property*
The size of the content view.

`contentInset` (page 294) *property*
The distance that the content view is inset from the enclosing scroll view.

Managing Scrolling

`scrollEnabled` (page 298) *property*
A Boolean value that determines whether scrolling is enabled.

`directionalLockEnabled` (page 296) *property*
A Boolean value that determines whether scrolling is disabled in a particular direction

`scrollsToTop` (page 299) *property*
A Boolean value that controls whether the scroll-to-top gesture is effective

- `scrollRectToVisible:animated:` (page 301)
  Scrolls a specific area of the content so that it is visible in the recevier.

`pagingEnabled` (page 298) *property*
A Boolean value that determines whether paging is enabled for the scroll view.

`bounces` (page 293) *property*
A Boolean value that controls whether the scroll view bounces past the edge of content and back again.

`alwaysBounceVertical` (page 292) *property*
A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content.

`alwaysBounceHorizontal` (page 292) *property*
A Boolean value that determines whether whether bouncing always occurs when horizontal scrolling reaches the end of the content view.

- `touchesShouldBegin:withEvent:inContentView:` (page 302)
Overridden by subclasses to customize the default behavior when a finger touches down in displayed content.

- `touchesShouldCancelInContentView:` (page 303)
Returns whether to cancel touches related to the content subview and start dragging.

`canCancelContentTouches` (page 294) *property*
A Boolean value that controls whether touches in the content view always lead to tracking.

`delaysContentTouches` (page 295) *property*
A Boolean value that determines whether the scroll view delays the handling of touch-down gestures.

`dragging` (page 296) *property*
A Boolean value that indicates whether the user has begun scrolling the content. (read-only)

`tracking` (page 300) *property*
Returns whether the user has touched the content to initiate scrolling. (read-only)

`decelerating` (page 295) *property*
Returns whether the content is moving in the scroll view after the user lifted his or her finger. (read-only)

Managing the Scroll Indicator

`indicatorStyle` (page 297) *property*
The style of the scroll indicators.

`scrollIndicatorInsets` (page 299) *property*
The distance the scroll indicators are inset from the edge of the scroll view.

`showsHorizontalScrollIndicator` (page 299) *property*
A Boolean value that controls whether the horizontal scroll indicator is visible.

`showsVerticalScrollIndicator` (page 300) *property*
A Boolean value that controls whether the vertical scroll indicator is visible.

- `flashScrollIndicators` (page 301)
Displays the scroll indicators momentarily.

Zooming and Panning

`maximumZoomScale` (page 297) *property*
A floating-point value that specifies the maximum zoom scale.

`minimumZoomScale` (page 297) *property*
A floating-point value that specifies the minimum zoom scale.

`zoomBouncing` (page 300) *property*
A Boolean value that indicates that zooming has exceeded the scaling limits specified for the receiver. (read-only)

`zooming` (page 301) *property*
A Boolean value that indicates whether the content view is currently zooming in or out. (read-only)

`bouncesZoom` (page 293) *property*
A Boolean value that determines whether the scroll view bounces zooming exceeds the maximum or minimum limiits.

Managing the Delegate

`delegate` (page 296) *property*
The delegate of the scroll-view object.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

alwaysBounceHorizontal

A Boolean value that determines whether whether bouncing always occurs when horizontal scrolling reaches the end of the content view.

```
@property(nonatomic) BOOL alwaysBounceHorizontal
```

Discussion

The default value is `NO`. if this property is set to YES and `bounces` (page 293) is `YES`, horizontal dragging is allowed even if the content is smaller than the bounds of the scroll view.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property alwaysBounceVertical` (page 292)

Declared In

`UIScrollView.h`

alwaysBounceVertical

A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content.

```
@property(nonatomic) BOOL alwaysBounceVertical
```

Discussion
The default value is NO. if this property is set to YES and bounces (page 293) is YES, vertical dragging is allowed even if the content is smaller than the bounds of the scroll view.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property alwaysBounceHorizontal (page 292)

Declared In
UIScrollView.h

bounces

A Boolean value that controls whether the scroll view bounces past the edge of content and back again.

```
@property(nonatomic) BOOL bounces
```

Discussion
If the value of the property is YES (the default), the scroll view bounces when it encounters a boundary of the content. Bouncing visually indicates that scrolling has reached an edge of the content. If the value is NO, scrolling stops immediately at the content boundary without bouncing.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property alwaysBounceVertical (page 292)
@property alwaysBounceHorizontal (page 292)

Declared In
UIScrollView.h

bouncesZoom

A Boolean value that determines whether the scroll view bounces zooming exceeds the maximum or minimum limiits.

```
@property(nonatomic) BOOL bouncesZoom
```

Discussion
If this property is set to YES and zooming exceeds either the maximum or minimum limits for scaling, the scroll view temporarily animates the content scaling just past these limits before returning to them. If the property is set to NO, zooming stops immediately at one a scaling limits. The default is YES .

Availability
Available in iPhone OS 2.0 and later.

See Also

`@property maximumZoomScale` (page 297)
`@property minimumZoomScale` (page 297)
`@property zoomBouncing` (page 300)
`@property zooming` (page 301)

Declared In
`UIScrollView.h`

canCancelContentTouches

A Boolean value that controls whether touches in the content view always lead to tracking.

```
@property(nonatomic) BOOL canCancelContentTouches
```

Discussion
If the value of the property is `YES` and a view in the content has begun tracking a finger touching it, if the user drags the finger enough to initiate a scroll, the view receives a `touchesCancelled:withEvent:` (page 283) message and the scroll view handles the touch as a scroll. If the value of the property is NO, the scroll view does not scroll regardless of finger movement once the content view starts tracking.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

contentInset

The distance that the content view is inset from the enclosing scroll view.

```
@property(nonatomic) UIEdgeInsets contentInset
```

Discussion
Use this property to add to the scrolling area around the content. The unit of size is points. The default value is `UIEdgeInsetsZero`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

contentOffset

The point at which the origin of the content view is offset from the origin of the scroll view.

```
@property(nonatomic) CGPoint contentOffset
```

Discussion
The default value is `CGPointZero`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setContentOffset:animated:` (page 302)

Declared In
`UIScrollView.h`

contentSize

The size of the content view.

```
@property(nonatomic) CGSize contentSize
```

Discussion
The unit of size is points. The default size is `CGSizeZero`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

decelerating

Returns whether the content is moving in the scroll view after the user lifted his or her finger. (read-only)

```
@property(nonatomic, readonly, getter=isDecelerating) BOOL decelerating
```

Discussion
The returned value is `YES` if user isn't dragging the content but scrolling is still occurring.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

delaysContentTouches

A Boolean value that determines whether the scroll view delays the handling of touch-down gestures.

```
@property(nonatomic) BOOL delaysContentTouches
```

Discussion
If the value of this property is `YES` (the default), the scroll view delays handling the touch-down gesture until it can determine if scrolling is the intent. if the value is `NO` , the scroll view immediately calls `touchesShouldBegin:withEvent:inContentView:` (page 302). See the class description for a fuller discussion.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

delegate

The delegate of the scroll-view object.

```
@property(nonatomic, assign) id<UIScrollViewDelegate> delegate
```

Discussion
The delegate must adopt the `UIScrollViewDelegate` protocol.The `UIScrollView` class, which does not retain the delegate, invokes each protocol method the delegate implements.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

directionalLockEnabled

A Boolean value that determines whether scrolling is disabled in a particular direction

```
@property(nonatomic, getter=isDirectionalLockEnabled) BOOL directionalLockEnabled
```

Discussion
The default value is `NO`, which means that scrolling is permitted in both horizontal and vertical directions. If the value is `YES` and the user begins dragging in one general direction (horizontally or vertically), the scroll view disables scrolling in the other direction.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

dragging

A Boolean value that indicates whether the user has begun scrolling the content. (read-only)

```
@property(nonatomic, readonly, getter=isDragging) BOOL dragging
```

Discussion
The value held by this property might require some time or distance of scrolling before it is set to `YES`.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property tracking (page 300)

Declared In
UIScrollView.h

indicatorStyle

The style of the scroll indicators.

```
@property(nonatomic) UIScrollViewIndicatorStyle indicatorStyle
```

Discussion
The default style is UIScrollViewIndicatorStyleDefault (page 304). See "Scroll Indicator Style" (page 304) for descriptions of these constants.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIScrollView.h

maximumZoomScale

A floating-point value that specifies the maximum zoom scale.

```
@property(nonatomic) float maximumZoomScale
```

Discussion
This value determines how large the content can be scaled. It must be greater than the minimum zoom scale for zooming to be enabled. The default value is 1.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property minimumZoomScale (page 297)
@property zoomBouncing (page 300)
@property bouncesZoom (page 293)
@property zoomBouncing (page 300)
@property zooming (page 301)

Declared In
UIScrollView.h

minimumZoomScale

A floating-point value that specifies the minimum zoom scale.

```
@property(nonatomic) float minimumZoomScale
```

Discussion
This value determines how small the content can be scaled. The default value is 1.0

Availability
Available in iPhone OS 2.0 and later.

See Also
- `@property maximumZoomScale` (page 297)
- `@property zoomBouncing` (page 300)
- `@property bouncesZoom` (page 293)
- `@property zoomBouncing` (page 300)
- `@property zooming` (page 301)

Declared In
`UIScrollView.h`

pagingEnabled

A Boolean value that determines whether paging is enabled for the scroll view.

```
@property(nonatomic, getter=isPagingEnabled) BOOL pagingEnabled
```

Discussion
If the value of the property is `YES`, the scroll view stops on multiples of the view bounds when the user scrolls. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

scrollEnabled

A Boolean value that determines whether scrolling is enabled.

```
@property(nonatomic, getter=isScrollEnabled) BOOL scrollEnabled
```

Discussion
If the value of the property is `YES` , scrolling is enabled, and if it is `NO` , scrolling is disabled. The default is `YES`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIScrollView.h`

scrollIndicatorInsets

The distance the scroll indicators are inset from the edge of the scroll view.

```
@property(nonatomic) UIEdgeInsets scrollIndicatorInsets
```

Discussion

The default value is *UIEdgeInsetsZero*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIScrollView.h

scrollsToTop

A Boolean value that controls whether the scroll-to-top gesture is effective

```
@property(nonatomic) BOOL scrollsToTop
```

Discussion

The scroll-to-top gesture is a tap on the status bar; when this property is YES, the scroll view jumps to the top of the content when this gesture occurs. The default value of this property is YES. By default, this gesture works on a single visible scroll view; if there are multiple scroll views (for example, a date picker) with this property set, or if the delegate returns NO in scrollViewWillScrollToTop: (page 566), UIScrollView ignores the request. After the scroll view scrolls to the top of the content view, it sends the delegate a scrollViewDidScrollToTop: (page 564) message.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIScrollView.h

showsHorizontalScrollIndicator

A Boolean value that controls whether the horizontal scroll indicator is visible.

```
@property(nonatomic) BOOL showsHorizontalScrollIndicator
```

Discussion

The default value is YES. The indicator is visible while tracking is underway and fades out after tracking.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIScrollView.h

showsVerticalScrollIndicator

A Boolean value that controls whether the vertical scroll indicator is visible.

```
@property(nonatomic) BOOL showsVerticalScrollIndicator
```

Discussion

The default value is `YES`. The indicator is visible while tracking is underway and fades out after tracking.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIScrollView.h`

tracking

Returns whether the user has touched the content to initiate scrolling. (read-only)

```
@property(nonatomic, readonly, getter=isTracking) BOOL tracking
```

Discussion

The property holds `YES` if the user has touched the content view but might not have yet have started dragging it.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property dragging` (page 296)

Declared In

`UIScrollView.h`

zoomBouncing

A Boolean value that indicates that zooming has exceeded the scaling limits specified for the receiver. (read-only)

```
@property(nonatomic, readonly, getter=isZoomBouncing) BOOL zoomBouncing
```

Discussion

The value of this property is `YES` if the scroll view is zooming back to a minimum or maximum zoom scaling value; otherwise the value is `NO` .

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property maximumZoomScale` (page 297)
`@property minimumZoomScale` (page 297)
`@property zooming` (page 301)
`@property bouncesZoom` (page 293)

Declared In
UIScrollView.h

zooming

A Boolean value that indicates whether the content view is currently zooming in or out. (read-only)

```
@property(nonatomic, readonly, getter=isZooming) BOOL zooming
```

Discussion
The value of this property is YES if user is making a zoom gesture, otherwise it is NO .

Availability
Available in iPhone OS 2.0 and later.

See Also
- @property maximumZoomScale (page 297)
- @property minimumZoomScale (page 297)
- @property zoomBouncing (page 300)
- @property bouncesZoom (page 293)

Declared In
UIScrollView.h

Instance Methods

flashScrollIndicators

Displays the scroll indicators momentarily.

```
- (void)flashScrollIndicators
```

Discussion
You should call this method whenever you bring the scroll view to front.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIScrollView.h

scrollRectToVisible:animated:

Scrolls a specific area of the content so that it is visible in the recevier.

```
- (void)scrollRectToVisible:(CGRect)rect animated:(BOOL)animated
```

Parameters
rect
A rectangle defining an area of the content view.

animated
YES if the scrolling should be animated, NO if it should be immediate.

Discussion
This method scrolls the content view so that the area defined by *rect* is just visible inside the scroll view. If the area is already visible, the method does nothing.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIScrollView.h

setContentOffset:animated:

Sets the offset from the content view's origin that corresponds to the receiver's origin.

```
- (void)setContentOffset:(CGPoint)contentOffset animated:(BOOL)animated
```

Parameters

contentOffset
A point (expressed in points) that is offset from the content view's origin.

animated
YES to animate the transition at a constant velocity to the new offset, NO to make the transition immediate.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property contentOffset (page 294)

Declared In
UIScrollView.h

touchesShouldBegin:withEvent:inContentView:

Overridden by subclasses to customize the default behavior when a finger touches down in displayed content.

```
- (BOOL)touchesShouldBegin:(NSSet *)touches withEvent:(UIEvent *)event
    inContentView:(UIView *)view
```

Parameters

touches
A set of UITouch instances that represent the touches for the starting phase of the event represented by *event*.

event
An object representing the event to which the touch objects in *touches* belong.

view
The subview in the content where the touch-down gesture occurred.

Return Value
Return NO if you don't want the scroll view to send event messages to *view*. If you want *view* to receive those messages, return YES (the default).

Discussion
The default behavior of UIScrollView is to invoke the UIResponder event-handling methods of the target subview that the touches occur in.

Availability
Available in iPhone OS 2.0 and later.

See Also
- touchesShouldCancelInContentView: (page 303)

Declared In
UIScrollView.h

touchesShouldCancelInContentView:
Returns whether to cancel touches related to the content subview and start dragging.

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

Parameters
view

The view object in the content that is being touched.

Return Value
YES to cancel further touch messages to *view*, NO to have view continue to receive those messages. The default returned value is YES if *view* is not a UIControl object; otherwise, it returns NO.

Discussion
The subview calls this method just after it starts sending tracking messages to the content view. If it receives NO from this method, it stops dragging and forwards the touch events to the content subview. The subview does not call this method if the value of the canCancelContentTouches (page 294) property is NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- touchesShouldBegin:withEvent:inContentView: (page 302)

Declared In
UIScrollView.h

Constants

Scroll Indicator Style

The style of the scroll indicators. You use these constants to set the value of the `indicatorStyle` (page 297) style.

```
typedef enum {
UIScrollViewIndicatorStyleDefault,
UIScrollViewIndicatorStyleBlack,
UIScrollViewIndicatorStyleWhite
} UIScrollViewIndicatorStyle;
```

Constants

`UIScrollViewIndicatorStyleDefault`

The default style of scroll indicator, which is black with a white border. This style is good against any content background.

Available in iPhone OS 2.0 and later.

Declared in `UIScrollView.h`

`UIScrollViewIndicatorStyleBlack`

A style of indicator which is black smaller than the default style. This style is good against a white content background.

Available in iPhone OS 2.0 and later.

Declared in `UIScrollView.h`

`UIScrollViewIndicatorStyleWhite`

A style of indicator is white and smaller than the default style. This style is good against a black content background.

Available in iPhone OS 2.0 and later.

Declared in `UIScrollView.h`

UISearchBar Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UISearchBar.h

Overview

The `UISearchBar` class implements a text field control for text-based searches. The control provides a text field for entering text, a search button, a bookmark button, and a cancel button. You use a delegate, an object conforming to the `UISearchBarDelegate` protocol, to implement the actions when text is entered and buttons are clicked. The `UISearchBar` object does not actually perform any searches.

Tasks

Setting Properties

`barStyle` (page 307) *property*
The style that specifies the receiver's appearance.

`delegate` (page 307) *property*
The search bar's delegate object.

`placeholder` (page 308) *property*
The string that is displayed when there is no other text in the text field.

`prompt` (page 308) *property*
A single line of text displayed at the top of the search bar.

`text` (page 309) *property*
The current or starting search text.

`tintColor` (page 309) *property*
The color used to tint the bar.

Setting Text Input Properties

`autocapitalizationType` (page 306) *property*
The auto-capitalization style for the text object.

`autocorrectionType` (page 306) *property*
The auto-correction style for the text object.

`keyboardType` (page 307) *property*
The keyboard style associated with the text object.

Configuring Buttons

`showsBookmarkButton` (page 308) *property*
A Boolean value indicating whether the bookmark button is displayed.

`showsCancelButton` (page 309) *property*
A Boolean value indicating whether the cancel button is displayed.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

autocapitalizationType

The auto-capitalization style for the text object.

```
@property(nonatomic) UITextAutocapitalizationType autocapitalizationType
```

Discussion

This property determines at what times the Shift key is automatically pressed, thereby making the typed character a capital letter. The default value for this property is `UITextAutocapitalizationTypeNone` (page 608).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UISearchBar.h`

autocorrectionType

The auto-correction style for the text object.

```
@property(nonatomic) UITextAutocorrectionType autocorrectionType
```

Discussion
This property determines whether auto-correction is enabled or disabled during typing. With auto-correction enabled, the text object tracks unknown words and suggests a replacement candidate to the user, replacing the typed text automatically unless the user explicitly overrides the action.

The default value for this property is `UITextAutocorrectionTypeDefault` (page 609), which for most input methods results in auto-correction being enabled.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

barStyle

The style that specifies the receiver's appearance.

```
@property(nonatomic) UIBarStyle barStyle
```

Discussion
See `UIBarStyle` (page 625) for possible values. The default value is `UIBarStyleDefault`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

delegate

The search bar's delegate object.

```
@property(nonatomic, assign) id<UISearchBarDelegate> delegate
```

Discussion
The delegate should conform to the `UISearchBarDelegate` protocol. Set this property to further modify the behavior. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

keyboardType

The keyboard style associated with the text object.

```
@property(nonatomic) UIKeyboardType keyboardType
```

Discussion
Text objects can be targeted for specific types of input, such as plain text, email, numeric entry, and so on. The keyboard style identifies what keys are available on the keyboard and which ones appear by default. The default value for this property is `UIKeyboardTypeDefault` (page 610).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

placeholder

The string that is displayed when there is no other text in the text field.

```
@property(nonatomic, copy) NSString *placeholder
```

Discussion
The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

prompt

A single line of text displayed at the top of the search bar.

```
@property(nonatomic, copy) NSString *prompt
```

Discussion
The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

showsBookmarkButton

A Boolean value indicating whether the bookmark button is displayed.

```
@property(nonatomic) BOOL showsBookmarkButton
```

Discussion
The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

showsCancelButton

A Boolean value indicating whether the cancel button is displayed.

```
@property(nonatomic) BOOL showsCancelButton
```

Discussion
The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

text

The current or starting search text.

```
@property(nonatomic, copy) NSString *text
```

Discussion
The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

tintColor

The color used to tint the bar.

```
@property(nonatomic, retain) UIColor *tintColor
```

Discussion
The bar style is ignored if this property is not `nil`. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

UISegmentedControl Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UISegmentedControl.h

Overview

A `UISegmentedControl` object is a horizontal control made of multiple segments, each segment functioning as a discrete button. A segmented control affords a compact means to group together a number of controls.

A segmented control can display a title (an `NSString` object) or an image (`UIImage` object). The `UISegmentedControl` object automatically resizes segments to fit proportionally within their superview unless they have a specific width set. If you set a segmented control to have a momentary style, a segment doesn't show itself as selected (blue background) when the user touches it. The disclosure button is always momentary and doesn't affect the actual selection. When you add and remove segments, you can request that the action be animated with sliding and fading effects.

In order to have the segmented control send a target object an action message you should register the target-action method using the `UIControlEventValueChanged` constant as shown below.

```
[segmentedControl addTarget:self
                     action:@selector(action:)
           forControlEvents:UIControlEventValueChanged];
```

Tasks

Initializing a Segmented Control

`- initWithItems:` (page 315)
Initializes and returns a segmented control with segments having the given titles or images.

Managing Segment Content

- `setImage:forSegmentAtIndex:` (page 319)
  Sets the content of a segment to a given image.
- `imageForSegmentAtIndex:` (page 315)
  Returns the image for a specific segment
- `setTitle:forSegmentAtIndex:` (page 319)
  Sets the title of a segment.
- `titleForSegmentAtIndex:` (page 320)
  Returns the title of the specified segment.

Managing Segments

- `insertSegmentWithImage:atIndex:animated:` (page 316)
  Inserts a segment at a specified position in the receiver and gives it an image as content.
- `insertSegmentWithTitle:atIndex:animated:` (page 316)
  Inserts a segment at a specific position in the receiver and gives it a title as content.

`numberOfSegments` (page 313) *property*
Returns the number of segments the receiver has. (read-only)

- `removeAllSegments` (page 317)
  Removes all segments of the receiver
- `removeSegmentAtIndex:animated:` (page 318)
  Removes the specified segment from the receiver, optionally animating the transition.

`selectedSegmentIndex` (page 314) *property*
The index number identifying the selected segment (that is, the last segment touched).

Managing Segment Behavior and Appearance

`momentary` (page 313) *property*
A Boolean value that determines whether segments in the receiver show selected state.

`segmentedControlStyle` (page 313) *property*
The style of the segmented control.

`tintColor` (page 314) *property*
The tint color of the segmented control.

- `setEnabled:forSegmentAtIndex:` (page 319)
  Enables the specified segment.
- `isEnabledForSegmentAtIndex:` (page 317)
  Returns whether the indicated segment is enabled.
- `setContentOffset:forSegmentAtIndex:` (page 318)
  Adjusts the offset for drawing the content (image or text) of the specified segment.
- `contentOffsetForSegmentAtIndex:` (page 314)
  Returns the offset for drawing the content (image or text) of the specified segment.
- `setWidth:forSegmentAtIndex:` (page 320)
  Sets the width of the specified segment of the receiver.

- widthForSegmentAtIndex: (page 321)
  Returns the width of the indicated segment of the receiver.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

momentary

A Boolean value that determines whether segments in the receiver show selected state.

```
@property(nonatomic, getter=isMomentary) BOOL momentary
```

Discussion

The default value of this property is NO. If it is set to YES, segments in the control do not show selected state and do not update the value of selectedSegment (page 314) after tracking ends.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UISegmentedControl.h

numberOfSegments

Returns the number of segments the receiver has. (read-only)

```
@property(nonatomic, readonly) NSUInteger numberOfSegments
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

UISegmentedControl.h

segmentedControlStyle

The style of the segmented control.

```
@property(nonatomic) UISegmentedControlStyle segmentedControlStyle
```

Discussion

The default style is UISegmentedControlStylePlain (page 322). See "Segmented Control Style" (page 321) for descriptions of valid constants.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UISegmentedControl.h

selectedSegmentIndex

The index number identifying the selected segment (that is, the last segment touched).

```
@property(nonatomic) NSInteger selectedSegmentIndex
```

Discussion

The default value is `UISegmentedControlNoSegment` (page 322) (no segment selected) until the user touches a segment. Set this property to `UISegmentedControlNoSegment` to turn off the current selection. `UISegmentedControl` ignores this property when the control is in momentary mode. When the user touches a segment to change the selection, the control event `UIControlEventValueChanged` (page 170) is generated; if the segmented control is set up to respond to this control event, it sends a action message to its target.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property momentary` (page 313)

Declared In

`UISegmentedControl.h`

tintColor

The tint color of the segmented control.

```
@property(nonatomic, retain) UIColor *tintColor
```

Discussion

The default value of this property is `nil` (no color). `UISegmentedControl` uses this property only if the style of the segmented control is `UISegmentedControlStyleBar`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UISegmentedControl.h`

Instance Methods

contentOffsetForSegmentAtIndex:

Returns the offset for drawing the content (image or text) of the specified segment.

```
- (CGSize)contentOffsetForSegmentAtIndex:(NSUInteger)segment
```

Parameters

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Return Value

The offset (specified as a `CGSize` structure) from the origin of the segment at which to draw the segment's content.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setContentOffset:forSegment:` (page 318)

Declared In

`UISegmentedControl.h`

imageForSegmentAtIndex:

Returns the image for a specific segment

```
- (UIImage *)imageForSegmentAtIndex:(NSUInteger)segment.
```

Parameters

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Return Value

Returns the image assigned to the receiver as content. If no image has been set, it returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setImage:forSegment:` (page 319)

Declared In

`UISegmentedControl.h`

initWithItems:

Initializes and returns a segmented control with segments having the given titles or images.

```
- (id)initWithItems:(NSArray *)items
```

Parameters

items

An array of `NSString` objects (for segment titles) or `UIImage` objects (for segment images).

Return Value
A `UISegmentedControl` object or `nil` if there was a problem in initializing the object.

Discussion
The returned segmented control is automatically sized to fit its content within the width of its superview.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISegmentedControl.h`

insertSegmentWithImage:atIndex:animated:

Inserts a segment at a specified position in the receiver and gives it an image as content.

```
- (void)insertSegmentWithImage:(UIImage *)image atIndex:(NSUInteger)segment
    animated:(BOOL)animated
```

Parameters
image
: An image object to use as the content of the segment.

segment
: An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it. The new segment is inserted just before the designated one.

animated
: `YES` if the insertion of the new segment should be animated, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `insertSegment:withTitle:animated:` (page 316)
- `removeSegment:animated:` (page 318)

Declared In
`UISegmentedControl.h`

insertSegmentWithTitle:atIndex:animated:

Inserts a segment at a specific position in the receiver and gives it a title as content.

```
- (void)insertSegmentWithTitle:(NSString *)title atIndex:(NSUInteger)segment
    animated:(BOOL)animated
```

Parameters
title
: A string to use as the segment's title.

segment
An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it. The new segment is inserted just before the designated one.

animated
`YES` if the insertion of the new segment should be animated, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `insertSegment:withImage:animated:` (page 316)
- `removeSegment:animated:` (page 318)

Declared In
`UISegmentedControl.h`

isEnabledForSegmentAtIndex:

Returns whether the indicated segment is enabled.

```
- (BOOL)isEnabledForSegmentAtIndex:(NSUInteger)segment
```

Parameters

segment
An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Return Value
`YES` if the given segment is enabled and `NO` if the segment is disabled. By default, segments are enabled.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setEnabled:forSegment:` (page 319)

Declared In
`UISegmentedControl.h`

removeAllSegments

Removes all segments of the receiver

```
- (void)removeAllSegments
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeSegment:animated:` (page 318)

Declared In
UISegmentedControl.h

removeSegmentAtIndex:animated:

Removes the specified segment from the receiver, optionally animating the transition.

```
- (void)removeSegmentAtIndex:(NSUInteger)segment animated:(BOOL)animated
```

Parameters

segment
An index number identifying a segment in the control. It must be a number between 0 and the number of segments (numberOfSegments (page 313)) minus 1; values exceeding this upper range are pinned to it.

animated
YES if the removal of the new segment should be animated, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeAllSegments (page 317)
- insertSegment:withImage:animated: (page 316)
- insertSegment:withTitle:animated: (page 316)

Declared In
UISegmentedControl.h

setContentOffset:forSegmentAtIndex:

Adjusts the offset for drawing the content (image or text) of the specified segment.

```
- (void)setContentOffset:(CGSize)offset forSegmentAtIndex:(NSUInteger)segment
```

Parameters

offset
The offset (specified as a CGSize type) from the origin of the segment at which to draw the segment's content. The default offset is (0,0).

segment
An index number identifying a segment in the control. It must be a number between 0 and the number of segments (numberOfSegments (page 313)) minus 1; values exceeding this upper range are pinned to it.

Availability
Available in iPhone OS 2.0 and later.

See Also
- contentOffsetForSegment: (page 314)

Declared In
UISegmentedControl.h

setEnabled:forSegmentAtIndex:

Enables the specified segment.

```
- (void)setEnabled:(BOOL)enabled forSegmentAtIndex:(NSUInteger)segment
```

Parameters

enabled

YES to enable the specified segment or NO to disable the segment. By default, segments are enabled.

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (numberOfSegments (page 313)) minus 1; values exceeding this upper range are pinned to it.

Availability

Available in iPhone OS 2.0 and later.

See Also

- isEnabledForSegment: (page 317)

Declared In

UISegmentedControl.h

setImage:forSegmentAtIndex:

Sets the content of a segment to a given image.

```
- (void)setImage:(UIImage *)image forSegmentAtIndex:(NSUInteger)segment
```

Parameters

image

An image object to display in the segment. .

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (numberOfSegments (page 313)) minus 1; values exceeding this upper range are pinned to it.

Discussion

A segment can only have an image or a title; it can't have both. There is no default image.

Availability

Available in iPhone OS 2.0 and later.

See Also

- imageForSegment: (page 315)

Declared In

UISegmentedControl.h

setTitle:forSegmentAtIndex:

Sets the title of a segment.

```
- (void)setTitle:(NSString *)title forSegmentAtIndex:(NSUInteger)segment
```

Parameters

title

An string to display in the segment as its title. .

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Discussion

A segment can only have an image or a title; it can't have both. There is no default title.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- titleForSegment:` (page 320)

Declared In

`UISegmentedControl.h`

setWidth:forSegmentAtIndex:

Sets the width of the specified segment of the receiver.

```
- (void)setWidth:(CGFloat)width forSegmentAtIndex:(NSUInteger)segment
```

Parameters

width

A float value specifying the width of the segment. The default value is {0.0}, which tells `UISegmentedControl` to automatically size the segment.

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- widthForSegment:` (page 321)

Declared In

`UISegmentedControl.h`

titleForSegmentAtIndex:

Returns the title of the specified segment.

```
- (NSString *)titleForSegmentAtIndex:(NSUInteger)segment
```

Parameters

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Return Value

Returns the string (title) assigned to the receiver as content. If no title has been set, it returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setTitle:forSegment:` (page 319)

Declared In

`UISegmentedControl.h`

widthForSegmentAtIndex:

Returns the width of the indicated segment of the receiver.

```
- (CGFloat)widthForSegmentAtIndex:(NSUInteger)segment
```

Parameters

segment

An index number identifying a segment in the control. It must be a number between 0 and the number of segments (`numberOfSegments` (page 313)) minus 1; values exceeding this upper range are pinned to it.

Return Value

A float value specifying the width of the segment. If the value is {0.0}, `UISegmentedControl` automatically sizes the segment.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setWidth:forSegment:` (page 320)

Declared In

`UISegmentedControl.h`

Constants

Segmented Control Style

The styles of the segmented control.

```
typedef enum {
    UISegmentedControlStylePlain,
    UISegmentedControlStyleBordered,
    UISegmentedControlStyleBar,
} UISegmentedControlStyle;
```

Constants

`UISegmentedControlStylePlain`

The large plain style for segmented controls. This style is the default.

Available in iPhone OS 2.0 and later.

Declared in `UISegmentedControl.h`

`UISegmentedControlStyleBordered`

The large bordered style for segmented controls.

Available in iPhone OS 2.0 and later.

Declared in `UISegmentedControl.h`

`UISegmentedControlStyleBar`

The small toolbar style for segmented controls. Segmented controls in this style can have a tint color (see `tintColor` (page 314)).

Available in iPhone OS 2.0 and later.

Declared in `UISegmentedControl.h`

Discussion

You use these constants as values for the `segmentedControlStyle` (page 313) property.

Declared In

`UISegmentedControl.h`

Segment Selection

A constant for indicating that no segment is selected.

```
enum {
    UISegmentedControlNoSegment = -1
};
```

Constants

`UISegmentedControlNoSegment`

A segment index value indicating that there is no selected segment. See `selectedSegment` (page 314) for further information.

Available in iPhone OS 2.0 and later.

Declared in `UISegmentedControl.h`

Declared In

`UISegmentedControl.h`

UISlider Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UISlider.h

Overview

A `UISlider` object is a visual control used to select a single value from a continuous range of values. Sliders are always displayed as horizontal bars. An indicator, or **thumb**, notes the current value of the slider and can be moved by the user to change the setting.

Customizing the Slider's Appearance

The most common way to customize the slider's appearance is to provide custom minimum and maximum value images. These images sit at either end of the slider control and indicate which value that end of the slider represents. For example, a slider used to control volume might display a small speaker with no sound waves emanating from it for the minimum value and display a large speaker with many sound waves emanating from it for the maximum value.

The bar on which the thumb rides is referred to as the slider's **track**. Slider controls draw the track using two distinct images, which are customizable. The region between the thumb and the end of the track associated with the slider's minimum value is drawn using the **minimum track image**. The region between the thumb and the end of the track associated with the slider's maximum value is drawn using the **maximum track image**. Different track images are used in order to provide context as to which end contains the minimum value. For example, the minimum track image typically contains a blue highlight while the maximum track image contains a white highlight. You can assign different pairs of track images to each of control states of the slder. Assigning different images to each state lets you customize the appearance of the slider when it is enabled, disabled, highlighted, and so on.

In addition to customizing the track images, you can also customize the appearance of the thumb itself. Like the track images, you can assign different thumb images to each control state of the slider.

Note: The slider control provides a set of default images for both the track and thumb. If you do not specify any custom images, those images are used automatically.

Tasks

Accessing the Slider's Value

`value` (page 328) *property*
Contains the receiver's current value.

`- setValue:animated:` (page 332)
Sets the receiver's current value, allowing you to animate the change visually.

Accessing the Slider's Value Limits

`minimumValue` (page 328) *property*
Contains the minimum value of the receiver.

`maximumValue` (page 327) *property*
Contains the maximum value of the receiver.

Modifying the Slider's Behavior

`continuous` (page 325) *property*
Contains a Boolean value indicating whether changes in the sliders value generate continuous update events.

Changing the Slider's Appearance

`minimumValueImage` (page 328) *property*
Contains the image that is drawn on the side of the slider representing the minimum value.

`maximumValueImage` (page 327) *property*
Contains the image that is drawn on the side of the slider representing the maximum value.

`currentMinimumTrackImage` (page 326) *property*
Contains the minimum track image currently being used to render the receiver. (read-only)

`- minimumTrackImageForState:` (page 330)
Returns the minimum track image associated with the specified control state.

`- setMinimumTrackImage:forState:` (page 331)
Assigns a minimum track image to the specified control states.

`currentMaximumTrackImage` (page 326) *property*
Contains the maximum track image currently being used to render the receiver. (read-only)

- `maximumTrackImageForState:` (page 329)
  Returns the maximum track image associated with the specified control state.
- `setMaximumTrackImage:forState:` (page 331)
  Assigns a maximum track image to the specified control states.

`currentThumbImage` (page 326) *property*
Contains the thumb image currently being used to render the receiver. (read-only)

- `thumbImageForState:` (page 333)
  Returns the thumb image associated with the specified control state.
- `setThumbImage:forState:` (page 332)
  Assigns a thumb image to the specified control states.

Overrides for Subclasses

- `maximumValueImageRectForBounds:` (page 329)
  Returns the drawing rectangle for the maximum value image.
- `minimumValueImageRectForBounds:` (page 330)
  Returns the drawing rectangle for the minimum value image.
- `trackRectForBounds:` (page 334)
  Returns the drawing rectangle for the slider's track.
- `thumbRectForBounds:trackRect:value:` (page 333)
  Returns the drawing rectangle for the slider's thumb image.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

continuous

Contains a Boolean value indicating whether changes in the sliders value generate continuous update events.

```
@property(nonatomic, getter=isContinuous) BOOL continuous
```

Discussion

If `YES`, the slider sends update events continuously to the associated target's action method. If `NO`, the slider only sends an action event when the user releases the slider's thumb control to set the final value.

The default value of this property is `YES`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UISlider.h`

currentMaximumTrackImage

Contains the maximum track image currently being used to render the receiver. (read-only)

```
@property(nonatomic, readonly) UIImage *currentMaximumTrackImage
```

Discussion

Sliders can have different track images for different control states. The image associated with this property reflects the maximum track image associated with the currently active control state. To get the maximum track image for a different control state, use the `maximumTrackImageForState:` method.

If no custom track images have been set using the `setMaximumTrackImage:forState:` method, this property contains the value `nil`. In that situation, the receiver uses the default maximum track image for drawing.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `maximumTrackImageForState:` (page 329)
- `setMaximumTrackImage:forState:` (page 331)

Declared In

`UISlider.h`

currentMinimumTrackImage

Contains the minimum track image currently being used to render the receiver. (read-only)

```
@property(nonatomic, readonly) UIImage *currentMinimumTrackImage
```

Discussion

Sliders can have different track images for different control states. The image associated with this property reflects the minimum track image associated with the currently active control state. To get the minimum track image for a different control state, use the `minimumTrackImageForState:` method.

If no custom track images have been set using the `setMinimumTrackImage:forState:` method, this property contains the value `nil`. In that situation, the receiver uses the default minimum track image for drawing.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `minimumTrackImageForState:` (page 330)
- `setMinimumTrackImage:forState:` (page 331)

Declared In

`UISlider.h`

currentThumbImage

Contains the thumb image currently being used to render the receiver. (read-only)

```
@property(nonatomic, readonly) UIImage *currentThumbImage
```

Discussion
Sliders can have different thumb images for different control states. The image associated with this property reflects the thumb image associated with the currently active control state. To get the thumb image for a different control state, use the `thumbImageForState:` method.

If no custom thumb images have been set using the `setThumbImage:forState:` method, this property contains the value `nil`. In that situation, the receiver uses the default thumb image for drawing.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `thumbImageForState:` (page 333)
- `setThumbImage:forState:` (page 332)

Declared In
`UISlider.h`

maximumValue

Contains the maximum value of the receiver.

```
@property(nonatomic) float maximumValue
```

Discussion
If you change the value of this property, and the current value of the receiver is above the new maximum, the current value is adjusted to match the new maximum value automatically.

The default value of this property is 1.0.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISlider.h`

maximumValueImage

Contains the image that is drawn on the side of the slider representing the maximum value.

```
@property(nonatomic, retain) UIImage *maximumValueImage
```

Discussion
The image you specify should fit within the bounding rectangle returned by the `maximumValueImageRectForBounds:` method. If it does not, the image is scaled to fit. In addition, the receiver's track is lengthened or shortened as needed to accommodate the image in its bounding rectangle.

This default value of this property is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

minimumValue

Contains the minimum value of the receiver.

```
@property(nonatomic) float minimumValue
```

Discussion
If you change the value of this property, and the current value of the receiver is below the new minimum, the current value is adjusted to match the new minimum value automatically.

The default value of this property is 0.0.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

minimumValueImage

Contains the image that is drawn on the side of the slider representing the minimum value.

```
@property(nonatomic, retain) UIImage *minimumValueImage
```

Discussion
The image you specify should fit within the bounding rectangle returned by the `minimumValueImageRectForBounds:` method. If it does not, the image is scaled to fit. In addition, the receiver's track is lengthened or shortened as needed to accommodate the image in its bounding rectangle.

This default value of this property is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

value

Contains the receiver's current value.

```
@property(nonatomic) float value
```

Discussion
Setting this property causes the receiver to redraw itself using the new value. To render an animated transition from the current value to the new value, you should use the `setValue:animated:` method instead.

If you try to set a value that is below the minimum or above the maximum value, the minimum or maximum value is set instead. The default value of this property is 0.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setValue:animated:` (page 332)

Declared In
`UISlider.h`

Instance Methods

maximumTrackImageForState:

Returns the maximum track image associated with the specified control state.

```
- (UIImage *)maximumTrackImageForState:(UIControlState)state
```

Parameters

state

The control state whose maximum track image you want. You should specify only one control state value for this parameter.

Return Value
The maximum track image associated with the specified state, or `nil` if an appropriate image could not be retrieved. This method might return `nil` if you specify multiple control states in the *state* parameter. For a description of track images, see "Customizing the Slider's Appearance" (page 323).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setMaximumTrackImage:forState:` (page 331)

Declared In
`UISlider.h`

maximumValueImageRectForBounds:

Returns the drawing rectangle for the maximum value image.

```
- (CGRect)maximumValueImageRectForBounds:(CGRect)bounds
```

Parameters

bounds

The bounding rectangle of the receiver.

Return Value
The computed drawing rectangle for the image.

Discussion
You should not call this method directly. If you want to customize the rectangle in which the maximum value image is drawn, you can override this method and return a different rectangle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

minimumTrackImageForState:
Returns the minimum track image associated with the specified control state.

```
- (UIImage *)minimumTrackImageForState:(UIControlState)state
```

Parameters
state

The control state whose minimum track image you want. You should specify only one control state value for this parameter.

Return Value
The minimum track image associated with the specified state, or `nil` if no image has been set. This method might also return `nil` if you specify multiple control states in the *state* parameter. For a description of track images, see "Customizing the Slider's Appearance" (page 323).

Availability
Available in iPhone OS 2.0 and later.

See Also
- setMinimumTrackImage:forState: (page 331)

Declared In
UISlider.h

minimumValueImageRectForBounds:
Returns the drawing rectangle for the minimum value image.

```
- (CGRect)minimumValueImageRectForBounds:(CGRect)bounds
```

Parameters
bounds

The bounding rectangle of the receiver.

Return Value
The computed drawing rectangle for the image.

Discussion
You should not call this method directly. If you want to customize the rectangle in which the minimum value image is drawn, you can override this method and return a different rectangle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

setMaximumTrackImage:forState:

Assigns a maximum track image to the specified control states.

```
- (void)setMaximumTrackImage:(UIImage *)image forState:(UIControlState)state
```

Parameters

image

The maximum track image to associate with the specified states.

state

The control state with which to associate the image.

Discussion
The orientation of the track image must match the orientation of the slider control. To facilitate the stretching of the image to fill the space between the thumb and end point, track images are usually defined in three regions. A stretchable region sits between two end cap regions. The end caps define the portions of the image that remain as is and are not stretched. The stretchable region is a 1-point wide area between the end caps that can be replicated to make the image appear longer.

To define the end cap sizes for a horizontally-oriented slider, assign an appropriate value to the image's `leftCapWidth` (page 208) property. For more information about how this value defines the regions of the slider, see the `UIImage` class.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- maximumTrackImageForState:` (page 329)

Declared In
UISlider.h

setMinimumTrackImage:forState:

Assigns a minimum track image to the specified control states.

```
- (void)setMinimumTrackImage:(UIImage *)image forState:(UIControlState)state
```

Parameters

image

The minimum track image to associate with the specified states.

state

The control state with which to associate the image.

Discussion
The orientation of the track image must match the orientation of the slider control. To facilitate the stretching of the image to fill the space between the thumb and end point, track images are usually defined in three regions. A stretchable region sits between two end cap regions. The end caps define the portions of the image that remain as is and are not stretched. The stretchable region is a 1-point wide area between the end caps that can be replicated to make the image appear longer.

To define the end cap sizes for a horizontally-oriented slider, assign an appropriate value to the image's `leftCapWidth` (page 208) property. For more information about how this value defines the regions of the slider, see the `UIImage` class.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISlider.h`

setThumbImage:forState:

Assigns a thumb image to the specified control states.

```
- (void)setThumbImage:(UIImage *)image forState:(UIControlState)state
```

Parameters

image
The thumb image to associate with the specified states.

state
The control state with which to associate the image.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- thumbImageForState:` (page 333)

Declared In
`UISlider.h`

setValue:animated:

Sets the receiver's current value, allowing you to animate the change visually.

```
- (void)setValue:(float)value animated:(BOOL)animated
```

Parameters

value
The new value to assign to the `value` property

animated
Specify `YES` to animate the change in value when the receiver is redrawn; otherwise, specify `NO` to draw the receiver with the new value only. Animations are performed asynchronously and do not block the calling thread.

Discussion
If you try to set a value that is below the minimum or above the maximum value, the minimum or maximum value is set instead. The default value of this property is 0.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property value` (page 328)

Declared In
`UISlider.h`

thumbImageForState:

Returns the thumb image associated with the specified control state.

```
- (UIImage *)thumbImageForState:(UIControlState)state
```

Parameters

state

The control state whose thumb image you want. You should specify only one control state value for this parameter.

Return Value
The thumb image associated with the specified state, or `nil` if an appropriate image could not be retrieved. This method might return `nil` if you specify multiple control states in the *state* parameter. For a description of track and thumb images, see "Customizing the Slider's Appearance" (page 323).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setThumbImage:forState:` (page 332)

Declared In
`UISlider.h`

thumbRectForBounds:trackRect:value:

Returns the drawing rectangle for the slider's thumb image.

```
- (CGRect)thumbRectForBounds:(CGRect)bounds trackRect:(CGRect)rect value:(float)value
```

Parameters

bounds

The bounding rectangle of the receiver.

rect

The drawing rectangle for the receiver's track, as returned by the `trackRectForBounds:` (page 334) method.

value

The current value of the slider.

Return Value
The computed drawing rectangle for the thumb image.

Discussion
You should not call this method directly. If you want to customize the thumb image's drawing rectangle, you can override this method and return a different rectangle. The rectangle you return should reflect the size of your thumb image and its current position on the slider's track.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

trackRectForBounds:

Returns the drawing rectangle for the slider's track.

```
- (CGRect)trackRectForBounds:(CGRect)bounds
```

Parameters
bounds

The bounding rectangle of the receiver.

Return Value
The computed drawing rectangle for the track. This rectangle corresponds to the entire length of the track between the minimum and maximum value images.

Discussion
You should not call this method directly. If you want to customize the track rectangle, you can override this method and return a different rectangle. The returned rectangle is used to scale the track and thumb images during drawing.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISlider.h

UISwitch Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UISwitch.h

Overview

You use the `UISwitch` class to create and manage the On/Off buttons you see, for example, in the preferences (Settings) for such services as Airplane Mode. These objects are known as switches.

The `UISwitch` class declares a property and a method to control its on/off state. You can also use methods of the superclass, `UISlider`. As with `UISlider`, when the user manipulates the switch control ("flips" it) a `UIControlEventValueChanged` (page 170) event is generated, which results in the control (if properly configured) sending an action message.

The `UISwitch` class is not customizable.

Tasks

Initializing the Switch Object

`- initWithFrame:` (page 336)
Returns an initialized switch object.

Setting the Off/On State

`on` (page 336) *property*
A Boolean value that determines the off/on state of the switch.

`- setOn:animated:` (page 337)
Set the state of the switch to On or Off, optionally animating the transition.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

on

A Boolean value that determines the off/on state of the switch.

```
@property(nonatomic, getter=isOn) BOOL on
```

Discussion

This property allows you to retrieve and set (without animation) a value determining whether the `UISwitch` object is on or off.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UISwitch.h`

Instance Methods

initWithFrame:

Returns an initialized switch object.

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

Parameters

frame
A rectangle defining the frame of the `UISwitch` object. The size components of this rectangle are ignored.

Return Value

An initialized `UISwitch` object or `nil` if the object could not be initialized.

Discussion

`UISwitch` overrides `initWithFrame:` (page 469) and enforces a size appropriate for the control.

Availability

Available in iPhone OS 2.0 and later.

Declared In
UISwitch.h

setOn:animated:

Set the state of the switch to On or Off, optionally animating the transition.

```
- (void)setOn:(BOOL)on animated:(BOOL)animated
```

Parameters

on

YES if the switch should be turn to the On position; NO if it should be turned to the Off position. If the switch is already in the designated position, nothing happens.

animated

YES to animate the "flipping" of the switch; otherwise NO.

Discussion
Setting the switch to either position does not result in an action message being sent.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISwitch.h

UITabBar Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITabBar.h

Overview

The `UITabBar` class implements a control for selecting one of two or more buttons, called items. The most common use of a tab bar is to implement a modal interface where tapping an item changes the selection. Use a `UIToolbar` object if you want to momentarily highlight or not change the appearance of an item when tapped. The `UITabBar` class provides the ability for the user to customize the tab bar by reordering, removing, and adding items to the bar. You can use a tab bar delegate to augment this behavior.

Use the `UITabBarItem` class to create items and the `setItems:animated:` (page 343) method to add them to a tab bar. All methods with an `animated:` argument allow you to optionally animate changes to the display. Use the `selectedItem` (page 341) property to access the current item.

Tasks

Getting and Setting Properties

`delegate` (page 340) *property*
The tab bar's delegate object.

Configuring Items

`items` (page 340) *property*
The items displayed on the tab bar.

`selectedItem` (page 341) *property*
The currently selected item on the tab bar.

- `setItems:animated:` (page 343)
Sets the items on the tab bar, with or without animation.

Customizing Tab Bars

- `beginCustomizingItems:` (page 341)
Presents a modal view allowing the user to customize the tab bar by adding, removing, and rearranging items on the tab bar.

- `endCustomizingAnimated:` (page 342)
Dismisses the modal view used to modify items on the tab bar.

- `isCustomizing` (page 342)
Returns a Boolean value indicating whether the user is customizing the tab bar.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

delegate

The tab bar's delegate object.

```
@property(assign) id<UITabBarDelegate> delegate
```

Discussion

The delegate should conform to the `UITabBarDelegate` protocol. Set this property to further modify the customizing behavior. The default value is `nil`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITabBar.h`

items

The items displayed on the tab bar.

```
@property(copy) NSArray *items
```

Discussion
The items, instances of `UITabBarItem`, that are visible on the tab bar in the order they appear in this array. Any changes to this property are not animated. Use the `setItems:animated:` (page 343) method to animate changes.

The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property selectedItem` (page 341)
`- setItems:animated:` (page 343)

Declared In
`UITabBar.h`

selectedItem

The currently selected item on the tab bar.

```
@property(assign) UITabBarItem *selectedItem
```

Discussion
Changes to this property show visual feedback in the user interface. The selected and unselected images displayed by an item are automatically created based on the alpha values in its original image property that you set. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property items` (page 340)
`- setItems:animated:` (page 343)

Declared In
`UITabBar.h`

Instance Methods

beginCustomizingItems:

Presents a modal view allowing the user to customize the tab bar by adding, removing, and rearranging items on the tab bar.

```
- (void)beginCustomizingItems:(NSArray *)items
```

Parameters

items

The items to display on the modal view that can be rearranged.

The *items* parameter should contain all items that can be added to the tab bar. Visible items not in *items* are fixed in place—they can not be removed or replaced by the user.

Discussion

Use this method to start customizing a tab bar. For example, create an Edit button that invokes this method when tapped. A modal view appears displaying all the items in *items* with a Done button at the top. Tapping the Done button dismisses the modal view. If the selected item is removed from the tab bar, the `selectedItem` (page 341) property is set to `nil`. Set the `delegate` (page 340) property to an object conforming to the `UITabBarDelegate` protocol to further modify this behavior.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `endCustomizingAnimated:` (page 342)
- `isCustomizing` (page 342)

Declared In

`UITabBar.h`

endCustomizingAnimated:

Dismisses the modal view used to modify items on the tab bar.

```
- (BOOL)endCustomizingAnimated:(BOOL)animated
```

Parameters

animated

If `YES`, animates the transition; otherwise, does not.

Return Value

`YES` if items on the tab bar changed; otherwise, `NO`.

Discussion

Typically, you do not need to use this method because the user dismisses the modal view by tapping the Done button.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `beginCustomizingItems:` (page 341)
- `isCustomizing` (page 342)

Declared In

`UITabBar.h`

isCustomizing

Returns a Boolean value indicating whether the user is customizing the tab bar.

```
- (BOOL)isCustomizing
```

Return Value
YES if the user is currently customizing the items on the tab bar; otherwise, NO. For example, by tapping an Edit button, a modal view appears allowing users to change the items on a tab bar. This method returns YES if this modal view is visible.

Availability
Available in iPhone OS 2.0 and later.

See Also
- beginCustomizingItems: (page 341)
- endCustomizingAnimated: (page 342)

Declared In
UITabBar.h

setItems:animated:
Sets the items on the tab bar, with or without animation.

```
- (void)setItems:(NSArray *)items animated:(BOOL)animated
```

Parameters
items

The items to display on the tab bar.

animated

If YES, animates the transition to the items; otherwise, does not.

Discussion
If *animated* is YES, the changes are dissolved or the reordering is animated—for example, removed items fade out and new items fade in. This method also adjusts the spacing between items.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property items (page 340)
@property selectedItem (page 341)

Declared In
UITabBar.h

UITabBarController Class Reference

Inherits from:	UIViewController : NSObject
Conforms to:	UITabBarDelegate NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITabBarController.h
Companion guide:	View Controller Programming Guide for iPhone OS

Overview

The `UITabBarController` class implements a controller that manages all aspects of a radio interface using a tab bar. It uses view controllers, instances of `UIViewController` supplied by the application, to configure the items and display views. Each view controller provides information on how to set up an item and supplies the view to be displayed when that item is selected. A tab bar controller manages a set of view controllers much as a tab bar manages items. Although a tab bar controller uses a tab bar in its implementation, you should never need to, nor should you, access the tab bar directly.

After you create a tab bar controller, you set its view controllers using the `viewControllers` (page 348) property. Also set the initial selection for radio interfaces using the `selectedViewController` (page 348) property. You should then attach a tab bar controller's view to a window to display the tab bar. The tab bar controller displays the selected view above the tab bar.

You configure the appearance of the tab bar by setting the properties of the view controllers you add to the tab bar controller. At a minimum, a view controller added to a tab bar controller should have the `title` (page 490) and `view` (page 491) properties set. Use the `tabBarItem` (page 490) property to specify the appearance of the view controller.

The tab bar controller also resizes and positions your views between the tab bar and a navigation bar (if present); otherwise, between the tab bar and status bar. Therefore, set the `autoresizingMask` (page 446) properties of your views so they are resized appropriately when displayed by the tab bar controller.

If you add more items than can be displayed by a tab bar, a More item automatically appears at the end of the tab bar. When the user taps the More item, a More view appears listing the items that don't fit on the tab bar. The user can select the items by tapping them in the list. If the `customizableViewControllers` (page 346) property is not `nil`, then an Edit button also appears on the More view allowing the user to customize the tab bar.

If you want to augment the behavior of customizing a tab bar (the user interface for adding, replacing, and removing items from a tab bar), set the `delegate` (page 347) property to an object conforming to the `UITabBarControllerDelegate` protocol and implement the delegate methods accordingly.

Read *View Controller Programming Guide for iPhone OS* to learn how to use this class.

This class is not intended to be subclassed.

Tasks

Getting and Setting Properties

`delegate` (page 347) *property*
The tab bar controller's delegate object.

Configuring Items

`viewControllers` (page 348) *property*
An array of view controllers corresponding to the items on the tab bar that the receiver manages.

- `setViewControllers:animated:` (page 349)
Sets the receivers view controllers.

`selectedViewController` (page 348) *property*
The view controller, with the currently selected item on the tab bar.

`moreNavigationController` (page 347) *property*
The view controller that manages the More item if it exists. (read-only)

`customizableViewControllers` (page 346) *property*
The subset of view controllers managed by this tab bar controller that can be customized.

`selectedIndex` (page 348) *property*
The index of the view controller, with the currently selected item on the tab bar.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

customizableViewControllers

The subset of view controllers managed by this tab bar controller that can be customized.

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

Discussion
If you add more items than can be displayed by a tab bar, a More item automatically appears on the tab bar managed by the `moreNavigationController` (page 347) property. When the user taps the More item, a More view appears containing an Edit button that allows items to be rearranged on the tab bar. Only the items belonging to this array can be customized. None of the items on the tab bar can be customized if this array is `nil`.

If you set the `viewControllers` (page 348) property, this property is also set to its default value. By default all view controllers are customizable.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property moreNavigationController` (page 347)

Declared In
`UITabBarController.h`

delegate

The tab bar controller's delegate object.

```
@property(nonatomic, assign) id<UITabBarControllerDelegate> delegate
```

Discussion
The delegate should conform to the `UITabBarControllerDelegate` protocol. The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITabBarController.h`

moreNavigationController

The view controller that manages the More item if it exists. (read-only)

```
@property(nonatomic, readonly) UINavigationController *moreNavigationController
```

Discussion
This property is `nil` if a More item does not exist.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property customizableViewControllers` (page 346)

Declared In
`UITabBarController.h`

selectedIndex

The index of the view controller, with the currently selected item on the tab bar.

```
@property(nonatomic) NSUInteger selectedIndex
```

Discussion

Use this property to change the tab bar's currently selected item. It's the index of a view controller in the `viewControllers` (page 348) array. Setting this property also sets the `selectedViewController` (page 348) property accordingly. The default value is `0`—the index of the first view controller.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property selectedViewController` (page 348)

Declared In

`UITabBarController.h`

selectedViewController

The view controller, with the currently selected item on the tab bar.

```
@property(nonatomic, assign) UIViewController *selectedViewController
```

Discussion

Setting this property also sets the `selectedIndex` (page 348) property accordingly. If the More item is selected, its view controller is the selected view controller. The default value is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property selectedIndex` (page 348)

Declared In

`UITabBarController.h`

viewControllers

An array of view controllers corresponding to the items on the tab bar that the receiver manages.

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

Discussion

If the view controller that manages the More item exists, it is not included in this array. Setting this property also sets the `customizableViewControllers` (page 346) property to its default value. The default value is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `setViewControllers:animated:` (page 349)

Declared In
`UITabBarController.h`

Instance Methods

setViewControllers:animated:

Sets the receivers view controllers.

```
- (void)setViewControllers:(NSArray *)viewControllers animated:(BOOL)animated
```

Parameters

viewControllers
An array of view controllers corresponding to the items on the tab bar that the receiver manages.

animated
If `YES`, the appearance of the selected view controller's view is animated; otherwise, it is not.

Discussion
Setting the `viewControllers` (page 348) property also sets the `customizableViewControllers` (page 346) property to its default value.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property viewControllers` (page 348)

Declared In
`UITabBarController.h`

UITabBarItem Class Reference

Inherits from:	UIBarItem : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITabBar.h

Overview

The `UITabBarItem` class implements an item on a tab bar, instances of the `UITabBar` class. A tab bar operates strictly in radio mode, where one item is selected at a time—tapping a tab bar item toggles the view above the tab bar. You can also specify a badge value on the tab bar item for adding additional visual information—for example, the Phone application uses a badge on the item to show the number of new messages. This class also provides a number of system defaults for creating items.

Use the `initWithTabBarSystemItem:tag:` (page 352) method to create one of the system items. Use the `initWithTitle:image:tag:` (page 353) method to create a custom item with the specified title and image.

Tasks

Initializing a Item

- `initWithTabBarSystemItem:tag:` (page 352)
  Creates and returns a new item containing the specified system item.
- `initWithTitle:image:tag:` (page 353)
  Creates and returns a new item using the specified properties.

Getting and Setting Properties

`badgeValue` (page 352) *property*
Text that is displayed in the upper-right corner of the item with a surrounding red oval.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

badgeValue

Text that is displayed in the upper-right corner of the item with a surrounding red oval.

```
@property(nonatomic, copy) NSString *badgeValue
```

Discussion
The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITabBarItem.h`

Instance Methods

initWithTabBarSystemItem:tag:

Creates and returns a new item containing the specified system item.

```
- (id)initWithTabBarSystemItem:(UITabBarSystemItem)systemItem tag:(NSInteger)tag
```

Parameters

systemItem
The system item to use as the first item on the tab bar. One of the constants defined in `UITabBarSystemItem` (page 353).

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

Return Value
A newly initialized item containing the specified system item. The item's target is `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithTitle:image:tag:` (page 353)

Declared In
UITabBarItem.h

initWithTitle:image:tag:

Creates and returns a new item using the specified properties.

```
- (id)initWithTitle:(NSString *)title image:(UIImage *)image tag:(NSInteger)tag
```

Parameters

title

The item's title. If `nil`, a title is not displayed.

image

The item's image. If `nil`, an image is not displayed.

The images displayed on the tab bar are derived from this image. If this image is too large to fit on the tab bar, it is scaled to fit. The size of an tab bar image is typically 30 x 30 points. The alpha values in the source image are used to create the unselected and selected images—opaque values are ignored.

tag

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

Return Value
Newly initialized item with the specified properties.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithTabBarSystemItem:tag:` (page 352)

Declared In
UITabBarItem.h

Constants

UITabBarSystemItem

System items that can be used on a tab bar.

```
typedef enum {
    UITabBarSystemItemMore,
    UITabBarSystemItemFavorites,
    UITabBarSystemItemFeatured,
    UITabBarSystemItemTopRated,
    UITabBarSystemItemRecents,
    UITabBarSystemItemContacts,
    UITabBarSystemItemHistory,
    UITabBarSystemItemBookmarks,
    UITabBarSystemItemSearch,
    UITabBarSystemItemDownloads,
    UITabBarSystemItemMostRecent,
    UITabBarSystemItemMostViewed,
} UITabBarSystemItem;
```

Constants

`UITabBarSystemItemMore`

The more system item.

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemFavorites`

The favorites system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemFeatured`

The featured system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemTopRated`

The top rated system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemRecents`

The recents system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemContacts`

The contacts system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemHistory`

The history system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemBookmarks`

The bookmarks system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemSearch`

The search system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemDownloads`

The downloads system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemMostRecent`

The most recent system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

`UITabBarSystemItemMostViewed`

The most viewed system item. 

Available in iPhone OS 2.0 and later.

Declared in `UITabBarItem.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITabBarItem.h`

UITableView Class Reference

Inherits from:	UIScrollView : UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITableView.h
Companion guide:	Table View Programming Guide for iPhone OS

Overview

An instance of `UITableView` (or simply, a table view) is a means for displaying and editing hierarchical lists of information.

A table view in the UIKit framework is limited to a single column because it is designed for a device with a small screen. `UITableView` is a subclass of `UIScrollView`, which allows users to scroll through the table, although `UITableView` allows vertical scrolling only. The cells comprising the individual items of the table are `UITableViewCell` objects; `UITableView` uses these objects to draw the visible rows of the table. Cells have content—titles and images—and can have, near the right edge, accessory views. Standard accessory views are disclosure indicators or detail disclosure buttons; the former leads to the next level in a data hierarchy and the latter leads to a detailed view of a selected item. Accessory views can also be framework controls, such as switches and sliders, or can be custom views. Table views can enter an editing mode where users can insert, delete, and reorder rows of the table.

A table view is made up of one or more sections, each with its own rows. Sections are identified by their index number within the table view, and rows are identified by their index number within a section. Any section can optionally be preceded by a section header, and optionally be followed by a section footer.

Table views can have one of two styles, `UITableViewStylePlain` (page 378) and `UITableViewStyleGrouped` (page 378). When you create a `UITableView` instance you must specify a table style, and this style cannot be changed. In the plain style, section headers and footers float above the content if the part of a complete section is visible. A table view can have an index that appears as a bar on the right hand side of the table (for example, "a" through "z"). You can touch a

particular label to jump to the target section. The grouped style of table view provides a default background color and a default background view for all cells. The background view provides a visual grouping for all cells in a particular section. For example, one group could be a person's name and title, another group for phone numbers that the person uses, and another group for email accounts and so on. See the Settings application for examples of grouped tables. Table views in the grouped style cannot have an index.

Many methods of `UITableView` take `NSIndexPath` objects as parameters and return values. `UITableView` declares a category on `NSIndexPath` that enables you to get the represented row index (`row` (page 36) property) and section index (`section` (page 36) property), and to construct an index path from a given row index and section index (`indexPathForRow:inSection:` (page 36) method). Especially in table views with multiple sections, you must evaluate the section index before identifying a row by its index number.

A `UITableView` object must have an object that acts as a data source and and object that acts as a delegate; typically these objects are either the application delegate or, more frequently, a custom `UITableViewController` object. The data source must adopt the `UITableViewDataSource` protocol and the delegate must adopt the `UITableViewDelegate` protocol. The data source provides information that `UITableView` needs to construct tables and manages the data model when rows of a table are inserted, deleted, or reordered. The delegate provides the cells used by tables and performs other tasks, such as managing accessory views and selections.

When sent a `setEditing:animated:` (page 377) message (with a first parameter of `YES`), the table view enters into editing mode where it shows the editing or reordering controls of each visible row, depending on the `editingStyle` (page 387) of each associated `UITableViewCell`. Clicking on the insertion or deletion control causes the data source to receive a `tableView:commitEditingStyle:forRowAtIndexPath:` (page 581) message. You commit a deletion or insertion by calling `deleteRowsAtIndexPaths:withRowAnimation:` (page 366) or `insertRowsAtIndexPaths:withRowAnimation:` (page 371), as appropriate. Also in editing mode, if a table-view cell has its `showsReorderControl` (page 392) property set to `YES`, the data source receives a `tableView:moveRowAtIndexPath:toIndexPath:` (page 582) message. The data source can selectively remove the reordering control for cells by implementing `tableView:canMoveRowAtIndexPath:` (page 580).

`UITableView` caches table-view cells only for visible rows, but caches row, header, and footer heights for the entire table. You can create custom `UITableViewCell` objects with content or behavioral characteristics that are different than the default cells; “A Closer Look at Table-View Cells" in *Table View Programming Guide for iPhone OS* explains how.

Tasks

Initializing a UITableView Object

`- initWithFrame:style:` (page 371)
Initializes and returns a table view object having the given frame and style.

Configuring a Table View

- `dequeueReusableCellWithIdentifier:` (page 367)
  Returns a reusable table-view cell object located by its identifier.

`style` (page 365) *property*
Returns the style of the receiver. (read-only)

- `numberOfRowsInSection:` (page 372)
  Returns the number of rows (table cells) in a specified section.

- `numberOfSections` (page 373)
  Returns the number of sections for the receiver.

`rowHeight` (page 363) *property*
The height of each row (table cell) in the receiver.

`separatorStyle` (page 364) *property*
The style for table cells used as separators.

`separatorColor` (page 364) *property*
The color of separator rows in the table view.

`tableHeaderView` (page 365) *property*
Returns an accessory view that is displayed above the table.

`tableFooterView` (page 365) *property*
Returns an accessory view that is displayed below the table.

`sectionHeaderHeight` (page 363) *property*
The height of section headers in the receiving table view.

`sectionFooterHeight` (page 363) *property*
The height of section footers in the receiving table view.

`sectionIndexMinimumDisplayRowCount` (page 364) *property*
The number of table rows at which to display the index list on the right edge of the table.

Accessing Cells and Sections

- `cellForRowAtIndexPath:` (page 366)
  Returns the table cell at the specified index path.

- `indexPathForCell:` (page 369)
  Returns an index path representing the row and section of a given table-view cell.

- `indexPathForRowAtPoint:` (page 369)
  Returns an index path identifying the row and section at the given pointt.

- `indexPathsForRowsInRect:` (page 370)
  An array of index paths each representing a row enclosed by a given rectangle.

- `visibleCells` (page 377)
  Returns the table cells that are visible in the receiver.

- `indexPathsForVisibleRows` (page 370)
  Returns an array of index paths each identifying a visible row in the receiver.

Scrolling the Table View

- `scrollToRowAtIndexPath:atScrollPosition:animated:` (page 376)
  Scrolls the receiver until a row identified by index path is at a particular location on the screen.
- `scrollToNearestSelectedRowAtScrollPosition:animated:` (page 375)
  Scrolls the row nearest to a specified position in the table view to that position.

Managing Selections

- `indexPathForSelectedRow` (page 370)
  Returns an index path identifying the row and section of the selected row.
- `selectRowAtIndexPath:animated:scrollPosition:` (page 376)
  Selects a row in the receiver identified by index path, optionally scrolling the row to a location in the receiver.
- `deselectRowAtIndexPath:animated:` (page 368)
  Deselects a given row identified by index path, with an option to animate the deselection.

Inserting and Deleting Cells

- `beginUpdates` (page 366)
  Begin a series of method calls that insert, delete, select, or delete rows and sections of the receiver.
- `endUpdates` (page 368)
  Conclude a series of method calls that insert, delete, select, or reload rows and sections of the receiver.
- `insertRowsAtIndexPaths:withRowAnimation:` (page 371)
  Inserts rows in the receiver at the locations identified by an array of index paths, with an option to animate the insertion
- `insertSections:withRowAnimation:` (page 372)
  Inserts one or more sections in the receiver, with an option to animate the insertion.
- `deleteRowsAtIndexPaths:withRowAnimation:` (page 366)
  Deletes the rows specified by an array of index paths, with an option to animate the deletion.
- `deleteSections:withRowAnimation:` (page 367)
  Deletes one or more sections in the receiver, with an option to animate the deletion.

Managing the Editing of Table Cells

`editing` (page 362) *property*
A Boolean value that determines whether the receiver is in editing mode.

- `setEditing:animated:` (page 377)
  Toggles the receiver into and out of editing mode.

`allowsSelectionDuringEditing` (page 361) *property*
A Boolean value that determines whether users can select cells while the receiver is in editing mode.

Reloading the Table

- `reloadData` (page 375)
  Reloads the rows and sections of the receiver.

Accessing Drawing Areas of the Table View

- `rectForSection:` (page 374)
  Returns the drawing area for a specified section of the receiver.
- `rectForRowAtIndexPath:` (page 374)
  Returns the drawing area for a row identified by index path.
- `rectForFooterInSection:` (page 373)
  Returns the drawing area for the footer of the specified section.
- `rectForHeaderInSection:` (page 374)
  Returns the drawin area for the header of the specified section.

Managing the Delegate and the Data Source

`dataSource` (page 361) *property*
The object that acts as the data source of the receiving table view.

`delegate` (page 362) *property*
The object that acts as the delegate of the receiving table view.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

allowsSelectionDuringEditing

A Boolean value that determines whether users can select cells while the receiver is in editing mode.

```
@property(nonatomic) BOOL allowsSelectionDuringEditing
```

Discussion

If the value of this property is `YES`, users can select rows during editing. The default value is `NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

dataSource

The object that acts as the data source of the receiving table view.

```
@property(nonatomic, assign) id<UITableViewDataSource> dataSource
```

Discussion
The data source must adopt the `UITableViewDataSource` protocol. The data source is not retained.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property delegate` (page 362)

Declared In
`UITableView.h`

delegate

The object that acts as the delegate of the receiving table view.

```
@property(nonatomic, assign) id<UITableViewDelegate> delegate
```

Discussion
The delegate must adopt the `UITableViewDelegate` protocol. The delegate is not retained.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property dataSource` (page 361)

Declared In
`UITableView.h`

editing

A Boolean value that determines whether the receiver is in editing mode.

```
@property(nonatomic, getter=isEditing) BOOL editing
```

Discussion
When the value of this property is `YES`, the table view is in editing mode: the cells of the table might show an insertion or deletion control on the left side of each cell and a reordering control on the right side, depending on how the cell is configured. (See *UITableViewCell Class Reference* for details.) Tapping a control causes the table view to invoke the data source method `tableView:commitEditingStyle:forRowAtIndexPath:` (page 581). The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setEditing:animated:` (page 377)

Declared In
`UITableView.h`

rowHeight

The height of each row (table cell) in the receiver.

```
@property(nonatomic) CGFloat rowHeight
```

Discussion

The row height is in points. You may set the row height for cells if the delegate doesn't implement the `tableView:heightForRowAtIndexPath:` (page 593) method. If you do not explicitly set the row height, `UITableView` sets it to a standard value.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

sectionFooterHeight

The height of section footers in the receiving table view.

```
@property(nonatomic) CGFloat sectionFooterHeight
```

Discussion

This value is used only in section group tables, and only if delegate the doesn't implement the `tableView:heightForFooterInSection:` (page 591) method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property tableFooterView` (page 365)

Declared In

`UITableView.h`

sectionHeaderHeight

The height of section headers in the receiving table view.

```
@property(nonatomic) CGFloat sectionHeaderHeight
```

Discussion

This value is used only in section group tables, and only if delegate the doesn't implement the `tableView:heightForHeaderInSection:` (page 592) method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property tableHeaderView` (page 365)

Declared In

`UITableView.h`

sectionIndexMinimumDisplayRowCount

The number of table rows at which to display the index list on the right edge of the table.

```
@property(nonatomic) NSInteger sectionIndexMinimumDisplayRowCount
```

Discussion

This property is applicable only to table views in the `UITableViewStylePlain` (page 378) style. The default value is `NSIntegerMax`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

separatorColor

The color of separator rows in the table view.

```
@property(nonatomic, retain) UIColor *separatorColor
```

Discussion

The default color is gray.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property separatorStyle` (page 364)

Declared In

`UITableView.h`

separatorStyle

The style for table cells used as separators.

```
@property(nonatomic) UITableViewCellSeparatorStyle separatorStyle
```

Discussion

The value of this property is one of the separator-style constants described in *UITableViewCell Class Reference* class reference. `UITableView` uses this property to set the separator style on the cell returned from the delegate in `tableView:cellForRowAtIndexPath:` (page 581).

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property separatorColor` (page 364)

Declared In

`UITableView.h`

style

Returns the style of the receiver. (read-only)

```
@property(nonatomic, readonly) UITableViewStyle style
```

Discussion
See "Table View Style" (page 378) for descriptions of the constants used to specify table-view style.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

tableFooterView

Returns an accessory view that is displayed below the table.

```
@property(nonatomic, retain) UIView *tableFooterView
```

Discussion
The default value is `nil`. The table footer view is different from a section header.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property sectionFooterHeight` (page 363)

Declared In
`UITableView.h`

tableHeaderView

Returns an accessory view that is displayed above the table.

```
@property(nonatomic, retain) UIView *tableHeaderView
```

Discussion
The default value is `nil`. The table header view is different from a section header.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property sectionHeaderHeight` (page 363)

Declared In
`UITableView.h`

Instance Methods

beginUpdates

Begin a series of method calls that insert, delete, select, or delete rows and sections of the receiver.

```
- (void)beginUpdates
```

Discussion

Call this method if you want subsequent insertions, deletion, and selection operations (for example, `cellForRowAtIndexPath:` (page 366) and `indexPathsForVisibleRows` (page 370)) to be animated simultaneously. This group of methods must conclude with an invocation of `endUpdates` (page 368). These method pairs are not nestable. If you do not make the insertion, deletion, and selection calls inside this block, table attributes such as row count might become invalid. You should not call `reloadData` (page 375) within the group; if you call this method within the group, you will need to perform any animations yourself.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

cellForRowAtIndexPath:

Returns the table cell at the specified index path.

```
- (UITableViewCell *)cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

indexPath

The index path locating the row in the receiver.

Return Value

An object representing a cell of the table or `nil` if the cell is not visible or `indexPath` is out of range.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- indexPathForCell:` (page 369)

Declared In

`UITableView.h`

deleteRowsAtIndexPaths:withRowAnimation:

Deletes the rows specified by an array of index paths, with an option to animate the deletion.

```
- (void)deleteRowsAtIndexPaths:(NSArray *)indexPaths
    withRowAnimation:(UITableViewRowAnimation)animation
```

Parameters

indexPaths

An array of `NSIndexPath` objects identifying the rows to delete.

animation

`YES` to animate the deletion of rows, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- insertRowsAtIndexPaths:withRowAnimation:` (page 371)

Declared In

`UITableView.h`

deleteSections:withRowAnimation:

Deletes one or more sections in the receiver, with an option to animate the deletion.

```
- (void)deleteSections:(NSIndexSet *)sections
    withRowAnimation:(UITableViewRowAnimation)animation
```

Parameters

sections

An index set that specifies the sections to delete from the receiving table view. If a section exists after the specified index location, it is moved up one index location.

animation

`YES` to animate the deletion of sections, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- insertSections:withRowAnimation:` (page 372)

Declared In

`UITableView.h`

dequeueReusableCellWithIdentifier:

Returns a reusable table-view cell object located by its identifier.

```
- (UITableViewCell *)dequeueReusableCellWithIdentifier:(NSString *)identifier
```

Parameters

identifier

A string identifying the cell object to be reused. By default, a reusable cell's identifier is its class name, but you can change it to any arbitrary value.

Return Value

A `UITableViewCell` object with the associated *identifier* or `nil` if no such object exists in the reusable-cell queue.

Discussion
For performance reasons, a table view's delegate should generally reuse `UITableViewCell` objects when it assigns cells to rows in its `tableView:cellForRowAtIndexPath:` (page 581) method. A table view maintains a queue or list of `UITableViewCell` objects that the table view's delegate has marked for reuse. It marks a cell for reuse by assigning it a reuse identifier when it creates it (that is, in the `initWithFrame:reuseIdentifier:` (page 394) method of `UITableViewCell`). The delegate can access specific "template" cell objects in this queue by invoking the `dequeueReusableCellWithIdentifier:` method. You can access a cell's reuse identifier through its `reuseIdentifier` property, which is defined by `UITableViewCell`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

deselectRowAtIndexPath:animated:
Deselects a given row identified by index path, with an option to animate the deselection.

```
- (void)deselectRowAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated
```

Parameters
indexPath
An index path identifying a row in the receiver.

animated
`YES` if you want to animate the deselection and `NO` if the change should be immediate.

Discussion
Calling this method does not cause the delegate to receive a `tableView:willSelectRowAtIndexPath:` (page 597) or `tableView:didSelectRowAtIndexPath:` (page 590) message, nor will it send `UITableViewSelectionDidChangeNotification` (page 380) notifications to observers.

Calling this method does not cause any scrolling to the deselected row.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- indexPathForSelectedRow` (page 370)

Declared In
`UITableView.h`

endUpdates
Conclude a series of method calls that insert, delete, select, or reload rows and sections of the receiver.

```
- (void)endUpdates
```

Discussion
You call this method to bracket a series of method calls that began with `beginUpdates` (page 366) and that consist of operations to insert, delete, select, and reload rows and sections of the table view. When you call `endUpdates`, `UITableView` animates the operations simultaneously. Invocations of beginUpdates and endUpdates cannot be nested. If you do not make the insertion, deletion, and selection calls inside this block, table attributes such as row count might become invalid.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

indexPathForCell:

Returns an index path representing the row and section of a given table-view cell.

```
- (NSIndexPath *)indexPathForCell:(UITableViewCell *)cell
```

Parameters

cell

A cell object of the table view.

Return Value
An index path representing the row and section of the cell or `nil` if the index path is invalid.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- cellForRowAtIndexPath:` (page 366)

Declared In
`UITableView.h`

indexPathForRowAtPoint:

Returns an index path identifying the row and section at the given pointt.

```
- (NSIndexPath *)indexPathForRowAtPoint:(CGPoint)point
```

Parameters

point

A point in the local coordinate system of the receiver (the table view's bounds).

Return Value
An index path representing the row and section associated with *point* or `nil` if the point is out of the bounds of any row.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- indexPathForCell:` (page 369)

Declared In
UITableView.h

indexPathForSelectedRow

Returns an index path identifying the row and section of the selected row.

```
- (NSIndexPath *)indexPathForSelectedRow
```

Return Value
An index path identifying the row and section indexes of the selected row or `nil` if the index path is invalid.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `selectRowAtIndexPath:animated:scrollPosition:` (page 376)

Declared In
UITableView.h

indexPathsForRowsInRect:

An array of index paths each representing a row enclosed by a given rectangle.

```
- (NSArray *)indexPathsForRowsInRect:(CGRect)rect
```

Parameters
rect
A rectangle defining an area of the table view in local coordinates.

Return Value
An array of `NSIndexPath` objects each representing a row and section index identifying a row within *rect*. Returns `nil` if *rect* is not valid.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `indexPathForRowAtPoint:` (page 369)

Declared In
UITableView.h

indexPathsForVisibleRows

Returns an array of index paths each identifying a visible row in the receiver.

```
- (NSArray *)indexPathsForVisibleRows
```

Return Value
An array of `NSIndexPath` objects each representing a row index and section index that together identify a visible row in the table view. Returns `nil` if no rows are visible.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- visibleCells` (page 377)

Declared In
`UITableView.h`

initWithFrame:style:

Initializes and returns a table view object having the given frame and style.

```
- (id)initWithFrame:(CGRect)frame style:(UITableViewStyle)style
```

Parameters
frame

A rectangle specifying the initial location and size of the table view in its superview's coordinates. The frame of the table view changes as table cells are added and deleted.

style

A constant that specifies the style of the table view. See "Table View Style" (page 378) for descriptions of valid constants.

Return Value
Returns an initialized `UITableView` object or `nil` if the object could not be successfully initialized.

Discussion
You must specify the style of a table view when you create it and you cannot thereafter modify the style. If you initialize the table view with the `UIView` method `initWithFrame:` (page 469), the `UITableViewStylePlain` (page 378) style is used as a default.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

insertRowsAtIndexPaths:withRowAnimation:

Inserts rows in the receiver at the locations identified by an array of index paths, with an option to animate the insertion

```
- (void)insertRowsAtIndexPaths:(NSArray *)indexPaths
    withRowAnimation:(UITableViewRowAnimation)animation
```

Parameters
indexPaths

An array of NSIndexPath objects each representing a row index and section index that together identify a row in the table view.

animation

A constant that either specifies the kind of animation to perform when inserting the cell or requests no animation. See "Table Cell Insertion and Deletion Animation" (page 379) for descriptions of the constants.

Discussion

`UITableView` call the relevant delegate and data source methods immediately afterwards to get the cells and other content for visible cells.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `insertSections:withRowAnimation:` (page 372)
- `deleteRowsAtIndexPaths:withRowAnimation:` (page 366)

Declared In

`UITableView.h`

insertSections:withRowAnimation:

Inserts one or more sections in the receiver, with an option to animate the insertion.

```
- (void)insertSections:(NSIndexSet *)sections
    withRowAnimation:(UITableViewRowAnimation)animation
```

Parameters

sections

An index set that specifies the sections to insert in the receiving table view. If a section already exists at the specified index location, it is moved down one index location.

animation

`YES` to animate the insertion of sections, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `insertRowsAtIndexPaths:withRowAnimation:` (page 371)
- `deleteSections:withRowAnimation:` (page 367)

Declared In

`UITableView.h`

numberOfRowsInSection:

Returns the number of rows (table cells) in a specified section.

```
- (NSInteger)numberOfRowsInSection:(NSInteger)section
```

Parameters

section

An index number that identifies a section of the table. Table views in a plain style have a section index of zero.

Return Value
An index number that identifies a row in the given section.

Discussion
`UITableView` gets the value returned by this method from its data source and caches it.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- numberOfSections` (page 373)

Declared In
`UITableView.h`

numberOfSections

Returns the number of sections for the receiver.

```
- (NSInteger)numberOfSections
```

Return Value
The number of sections in the table view.

Discussion
`UITableView` gets the value returned by this method from its data source and caches it.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- numberOfRowsInSection:` (page 372)

Declared In
`UITableView.h`

rectForFooterInSection:

Returns the drawing area for the footer of the specified section.

```
- (CGRect)rectForFooterInSection:(NSInteger)section
```

Parameters

section
An index number identifying a section of the table view. Plain-style table views always have a section index of zero.

Return Value
A rectangle defining the area in which the table view draws the section footer.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

rectForHeaderInSection:

Returns the drawin area for the header of the specified section.

```
- (CGRect)rectForHeaderInSection:(NSInteger)section
```

Parameters

section

An index number identifying a section of the table view. Plain-style table views always have a section index of zero.

Return Value
A rectangle defining the area in which the table view draws the section header.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

rectForRowAtIndexPath:

Returns the drawing area for a row identified by index path.

```
- (CGRect)rectForRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

indexPath

An index path object that identifies a row by its index and its section index.

Return Value
A rectangle defining the area in which the table view draws the row or `CGZeroRect` if *indexPath* is invalid.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

rectForSection:

Returns the drawing area for a specified section of the receiver.

```
- (CGRect)rectForSection:(NSInteger)section
```

Parameters

section

An index number identifying a section of the table view. Plain-style table views always have a section index of zero.

Return Value
A rectangle defining the area in which the table view draws the section.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

reloadData

Reloads the rows and sections of the receiver.

```
- (void)reloadData
```

Discussion
Call this method to reload all the data that is used to construct the table, including cells, section headers and footers, index arrays, and so on. For efficiency, the table view redisplays only those rows that are visible. It adjusts offsets if the table shrinks as a result of the reload. The table view's delegate or data source calls this method when it wants the table view to completely reload its data. It should not be called in the methods that insert or delete rows, especially within an animation block implemented with calls to `beginUpdates` (page 366) and `endUpdates` (page 368)

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

scrollToNearestSelectedRowAtScrollPosition:animated:

Scrolls the row nearest to a specified position in the table view to that position.

```
-
    (void)scrollToNearestSelectedRowAtScrollPosition:(UITableViewScrollPosition)scrollPosition
     animated:(BOOL)animated
```

Parameters
scrollPosition
: A constant that identifies a relative position in the receiving table view (top, middle, bottom) for the row when scrolling concludes. See "Table View Scroll Position" (page 378) a descriptions of valid constants.

animated
: `YES` if you want to animate the change in position, `NO` if it should be immediate.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- scrollToRowAtIndexPath:atScrollPosition:animated:` (page 376)

Declared In
UITableView.h

scrollToRowAtIndexPath:atScrollPosition:animated:

Scrolls the receiver until a row identified by index path is at a particular location on the screen.

```
- (void)scrollToRowAtIndexPath:(NSIndexPath *)indexPath
    atScrollPosition:(UITableViewScrollPosition)scrollPosition
    animated:(BOOL)animated
```

Parameters

indexPath

An index path that identifies a row in the table view by its row index and its section index.

scrollPosition

A constant that identifies a relative position in the receiving table view (top, middle, bottom) for *row* when scrolling concludes. See "Table View Scroll Position" (page 378) a descriptions of valid constants.

animated

`YES` if you want to animate the change in position, `NO` if it should be immediate.

Discussion

Invoking this method does not cause the delegate to receive a

Availability

Available in iPhone OS 2.0 and later.

See Also

`- scrollToNearestSelectedRowAtScrollPosition:animated:` (page 375)

Declared In

`UITableView.h`

selectRowAtIndexPath:animated:scrollPosition:

Selects a row in the receiver identified by index path, optionally scrolling the row to a location in the receiver.

```
- (void)selectRowAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated
    scrollPosition:(UITableViewScrollPosition)scrollPosition
```

Parameters

indexPath

An index path identifying a row in the receiver.

animated

`YES` if you want to animate the selection and any change in position, `NO` if the change should be immediate.

scrollPosition

A constant that identifies a relative position in the receiving table view (top, middle, bottom) for the row when scrolling concludes. See "Table View Scroll Position" (page 378) a descriptions of valid constants.

Discussion
Calling this method does not cause the delegate to receive a `tableView:willSelectRowAtIndexPath:` (page 597) or `tableView:didSelectRowAtIndexPath:` (page 590) message, nor will it send `UITableViewSelectionDidChangeNotification` (page 380) notifications to observers.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- indexPathForSelectedRow` (page 370)

Declared In
`UITableView.h`

setEditing:animated:
Toggles the receiver into and out of editing mode.

```
- (void)setEditing:(BOOL)editing animated:(BOOL)animate
```

Parameters
editing
: `YES` to enter editing mode, `NO` to leave it. The default value is `NO` .

animate
: `YES` to animate the transition to editing mode, `NO` to make the transition immediate.

Discussion
When you call this method with the value of *editing* set to `YES`, the table view goes into editing mode by calling `setEditing:animated:` (page 395) on each visible `UITableViewCell` object. Calling this method with *editing* set to `NO` turns off editing mode. In editing mode, the cells of the table might show an insertion or deletion control on the left side of each cell and a reordering control on the right side, depending on how the cell is configured. (See *UITableViewCell Class Reference* for details.) The data source of the table view can selectively exclude cells from editing mode by implementing `tableView:canEditRowAtIndexPath:` (page 579).

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property editing` (page 362)

Declared In
`UITableView.h`

visibleCells
Returns the table cells that are visible in the receiver.

```
- (NSArray *)visibleCells
```

Return Value
An array containing `UITableViewCell` objects, each representing a visible cell in the receiving table view.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- indexPathsForVisibleRows` (page 370)

Declared In
`UITableView.h`

Constants

Table View Style
The style of the table view.

```
typedef enum {
    UITableViewStylePlain,
    UITableViewStyleGrouped
} UITableViewStyle;
```

Constants
`UITableViewStylePlain`
A plain table view. Any section headers or footers are displayed as inline separators and float when the table view is scrolled.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewStyleGrouped`
A table view whose sections present distinct groups of rows. The section headers and footers do not float.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

Discussion
You set the table style when you initialize the table view (see `initWithFrame:style:` (page 371)). You cannot modify the style thereafter.

Declared In
`UITableView.h`

Table View Scroll Position
The position in the table view (top, middle, bottom) to which a given row is scrolled.

```
typedef enum {
    UITableViewScrollPositionNone,
    UITableViewScrollPositionTop,
    UITableViewScrollPositionMiddle,
    UITableViewScrollPositionBottom
} UITableViewScrollPosition;
```

Constants

`UITableViewScrollPositionNone`

The table view does not scroll a row to any particular position. This is the default.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewScrollPositionTop`

The row of interest is scrolled to the top of the visible table view.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewScrollPositionMiddle`

The row of interest is scrolled to the middle of the visible table view.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewScrollPositionBottom`

The row of interest is scrolled to the bottom of the visible table view.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

Discussion

You set the scroll position through a parameter of the `selectRowAtIndexPath:animated:scrollPosition:` (page 376), `scrollToNearestSelectedRowAtScrollPosition:animated:` (page 375), `cellForRowAtIndexPath:` (page 366), and `indexPathForSelectedRow` (page 370) methods.

Declared In

`UITableView.h`

Table Cell Insertion and Deletion Animation

The type of animation when cells are inserted or deleted.

```
typedef struct {
    UITableViewRowAnimationFade,
    UITableViewRowAnimationRight,
    UITableViewRowAnimationLeft,
    UITableViewRowAnimationTop,
    UITableViewRowAnimationBottom
} UITableViewRowAnimation;
```

Constants

`UITableViewRowAnimationFade`

The inserted or deleted row or rows fades into or out of the table view.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewRowAnimationRight`
The inserted row or rows slides in from the right; the deleted row or rows slides out to the right.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewRowAnimationLeft`
The inserted row or rows slides in from the left; the deleted row or rows slides out to the left.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewRowAnimationTop`
The inserted row or rows slides in from the top; the deleted row or rows slides out toward the top.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

`UITableViewRowAnimationBottom`
he inserted row or rows slides in from the bottom; the deleted row or rows slides out toward the bottow.

Available in iPhone OS 2.0 and later.

Declared in `UITableView.h`

Discussion
You specify one of these constants as a parameter of the `indexPathsForVisibleRows` (page 370), `insertSections:withRowAnimation:` (page 372), and `deleteSections:withRowAnimation:` (page 367) methods.

Declared In
`UITableView.h`

Notifications

UITableViewSelectionDidChangeNotification

Posted when the selected row in the posting table view changes.

There is no `userInfo` dictionary associated with this notification.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

UITableViewCell Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITableViewCell.h
Companion guide:	Table View Programming Guide for iPhone OS

Overview

The `UITableViewCell` class defines the attributes and behavior of the cells that appear in `UITableView` objects.

A `UITableViewCell` object (or table cell) includes properties and methods for managing selected state, editing state and controls, accessory views, reordering controls, cell background, and content indentation. There are also properties for associating target objects and action selectors with editing controls or accessory views. The class additionally includes properties for setting and managing cell content, specifically text and images.

You have two ways of extending the standard `UITableViewCell` object. To create cells with multiple, variously formatted and sized strings and images for content, you can get the cell's content view (through its `contentView` (page 386) property) and add subviews to it. You can also subclass `UITableViewCell` to obtain cell characteristics and behavior specific to your application's needs. See "A Closer Look at Table-View Cells" in *Table View Programming Guide for iPhone OS* for details.

Tasks

Initializing a UITableViewCell Object

- `initWithFrame:reuseIdentifier:` (page 394)
  Initializes and returns a table-view cell object.

Reusing Cells

`reuseIdentifier` (page 389) *property*
A string used to identify a cell that is reusable. (read-only)

- `prepareForReuse` (page 395)
  Prepares a reusable cell for reuse by the table view's delegate.

Managing Text as Cell Content

`text` (page 393) *property*
The text of the cell.

`font` (page 387) *property*
The font of the title.

`textAlignment` (page 393) *property*
A constant that specifies the alignment of text in the cell.

`textColor` (page 393) *property*
The color of the title text.

`selectedTextColor` (page 391) *property*
The color of the title text when the cell is selected.

`lineBreakMode` (page 389) *property*
The mode for for wrapping and truncating text in the cell.

Managing Images as Cell Content

`image` (page 388) *property*
The image to use as content for the cell.

`selectedImage` (page 390) *property*
The image to use a cell content when the cell is selected.

Accessing Views of the Cell Object

`contentView` (page 386) *property*
Returns the content view of of the cell object. (read-only)

`backgroundView` (page 385) *property*
The view used as the background of the cell.

`selectedBackgroundView` (page 390) *property*
The view used as the background of the cell when it is selected.

Managing Cell Selection

`selected` (page 390) *property*
A Boolean value that indicates whether the cell is selected.

`selectionStyle` (page 391) *property*
The style of selection for a cell.

`- setSelected:animated:` (page 396)
Sets the selected state of the cell, optionally animating the transition between states.

Managing Targets and Actions

`target` (page 392) *property*
The target object to receive action messages.

`editAction` (page 386) *property*
The selector defining the action message to invoke when users tap the insert or delete button.

`accessoryAction` (page 384) *property*
The selector defining the action message to invoke when users tap the accessory view.

Editing the Cell

`editing` (page 386) *property*
A Boolean value that indicates whether the cell is in an editable state.

`- setEditing:animated:` (page 395)
Toggles the receiver into and out of editing mode.

`editingStyle` (page 387) *property*
The editing style of the cell. (read-only)

`showingDeleteConfirmation` (page 392) *property*
Returns whether the cell is currently showing the delete-confirmation button. (read-only)

`showsReorderControl` (page 392) *property*
A Boolean value that determines whether the cell shows the reordering control.

Managing Accessory Views

`accessoryView` (page 385) *property*
A view that is used (typically as a control) on the right side of the cell.

`accessoryType` (page 384) *property*
A constant that specifies the type of standard accessory view the cell should use.

`hidesAccessoryWhenEditing` (page 388) *property*
A Boolean value that determines whether the accessory view is hidden when the cell is being edited.

Managing Content Indentation

`indentationLevel` (page 388) *property*
Adjusts the indentation level of a cell whose content is indented.

`indentationWidth` (page 388) *property*
The width for each level of indentation of a cell's content.

`shouldIndentWhileEditing` (page 391) *property*
A Boolean value that controls whether the cell background is indented when the table view is in editing mode.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

accessoryAction

The selector defining the action message to invoke when users tap the accessory view.

```
@property(nonatomic) SEL accessoryAction
```

Discussion

The accessory view is a `UITableViewCell`-defined control, framework control, or custom control on the right side of the cell. It is often used to display a new view related to the selected cell. See `accessoryView` (page 385) for more information. If the value of this property is `NULL`, no action message is sent.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property target` (page 392)
`@property editAction` (page 386)

Declared In

`UITableViewCell.h`

accessoryType

A constant that specifies the type of standard accessory view the cell should use.

```
@property(nonatomic) UITableViewCellAccessoryType accessoryType
```

Discussion

The accessory view appears in the the right side of the cell. The standard accessory views include the disclosure chevron; for a description of valid `accessoryType` constants, see "Cell Accessory Type" (page 397). The default is `UITableViewCellAccessoryNone` (page 398). If a custom accessory view is set through the `accessoryView` (page 385) property, the value of this property is ignored. If the cell is enabled, the accessory view tracks touches and, if tapped, sends the accessory action message set through the `accessoryAction` (page 384) property.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property hidesAccessoryWhenEditing` (page 388)
"Accessing Views of the Cell Object" (page 382)

Declared In
`UITableViewCell.h`

accessoryView
A view that is used (typically as a control) on the right side of the cell.

```
@property(nonatomic, retain) UIView *accessoryView
```

Discussion
If the value of this property is not `nil`, the `UITableViewCell` class uses the given view for the accessory view and ignores the value of the `accessoryType` (page 384) property. The provided accessory view can be a framework-provided control or label or a custom view. The accessory view appears in the the right side of the cell. If the cell is enabled (through the `UIView` property `userInteractionEnabled` (page 453)) , the accessory view tracks touches and, if tapped, sends the accessory action message set through the `accessoryAction` (page 384) property.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property hidesAccessoryWhenEditing` (page 388)
"Accessing Views of the Cell Object" (page 382)

Declared In
`UITableViewCell.h`

backgroundView
The view used as the background of the cell.

```
@property(nonatomic, retain) UIView *backgroundView
```

Discussion
The default is `nil` for cells in plain-style tables (`UITableViewStylePlain` (page 378)) and non-`nil` for grouped-style tables `UITableViewStyleGrouped` (page 378)). `UITableViewCell` adds the background view as a subview behind all other views and uses its current frame location.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property contentView` (page 386)
`@property selectedBackgroundView` (page 390)

Declared In
UITableViewCell.h

contentView

Returns the content view of of the cell object. (read-only)

```
@property(nonatomic, readonly, retain) UIView *contentView
```

Discussion
The content view of a UITableViewCell object is the default superview for content displayed by the cell. If you want to customize cells by simply adding additional views, you should add them to the content view so they will be positioned appropriately as the cell transitions into and out of editing mode.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property backgroundView (page 385)

Declared In
UITableViewCell.h

editAction

The selector defining the action message to invoke when users tap the insert or delete button.

```
@property(nonatomic) SEL editAction
```

Discussion
When the cell's table is in editing mode, the cell displays a green insert control or a red delete control to the left of it. (The selectedBackgroundView (page 390) constant applied to the cell via the editingStyle (page 387) property determines which control is used.) Typically, the associated UITableView object sets the editing action for all cells; you can use this property to alter the editing action for individual cells. If the value of this property is NULL, no action message is sent.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property target (page 392)
@property accessoryAction (page 384)

Declared In
UITableViewCell.h

editing

A Boolean value that indicates whether the cell is in an editable state.

```
@property(nonatomic, getter=isEditing) BOOL editing
```

Discussion
When a cell is in an editable state, it displays the editing controls specified for it: the green insertion control, the red deletion control, or (on the right side) the reordering control. Use `editingStyle` (page 387) and `showsReorderControl` (page 392) to specify these controls for the cell.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

editingStyle

The editing style of the cell. (read-only)

```
@property(nonatomic, readonly) UITableViewCellEditingStyle editingStyle
```

Discussion
One of the constants described in "`Cell Editing Style`" (page 397) is used as the value of this property; it specifies whether the cell is in an editable state and, if it is, whether it shows an insertion or deletion control. The default value is `UITableViewCellEditingStyleNone` (page 397) (not editable). The delegate returns the value this property for a particular cell in its implementation of the `tableView:editingStyleForRowAtIndexPath:` (page 591) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property editing` (page 386)

Declared In
`UITableViewCell.h`

font

The font of the title.

```
@property(nonatomic, retain) UIFont *font
```

Discussion
If the value of this property is `nil` (the default), `UITableViewCell` uses a standard font optimized for the device.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

hidesAccessoryWhenEditing

A Boolean value that determines whether the accessory view is hidden when the cell is being edited.

```
@property(nonatomic) BOOL hidesAccessoryWhenEditing
```

Discussion
The default value is `YES`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

image

The image to use as content for the cell.

```
@property(nonatomic, retain) UIImage *image
```

Discussion
The default value of the property is `nil` (no image). Images are positioned to the left of the title.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property selectedImage` (page 390)

Declared In
`UITableViewCell.h`

indentationLevel

Adjusts the indentation level of a cell whose content is indented.

```
@property(nonatomic) NSInteger indentationLevel
```

Discussion
The default value of the property is zero (no indentation). The width for each level of indentation is determined by the `indentationWidth` (page 388) property.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

indentationWidth

The width for each level of indentation of a cell's content.

```
@property(nonatomic) CGFloat indentationWidth
```

Discussion
The default indentation width is 10.0 points.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property indentationLevel` (page 388)

Declared In
`UITableViewCell.h`

lineBreakMode

The mode for for wrapping and truncating text in the cell.

```
@property(nonatomic) UILineBreakMode lineBreakMode
```

Discussion
For further information, see the `UILineBreakMode` (page 50) constants described in *NSString UIKit Additions Reference*. The default value is `UILineBreakModeTailTruncation` (page 51).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

reuseIdentifier

A string used to identify a cell that is reusable. (read-only)

```
@property(nonatomic, readonly, copy) NSString *reuseIdentifier
```

Discussion
The reuse identifier is associated with a `UITableViewCell` object that the table-view delegate creates with the intent to reuse it as the basis (for performance reasons) for multiple rows of a table view. It is assigned to the cell object in `initWithFrame:reuseIdentifier:` (page 394) and cannot be changed thereafter. A `UITableView` object maintains a queue (or list) of the currently reusable cells, each with its own reuse identifier, and makes them available to the delegate in the `dequeueReusableCellWithIdentifier:` (page 367) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- prepareForReuse` (page 395)

Declared In
`UITableViewCell.h`

selected

A Boolean value that indicates whether the cell is selected.

```
@property(nonatomic, getter=isSelected) BOOL selected
```

Discussion

When the the selected state of a cell to `YES`, it draws the background for selected cells with its title in white. The default value is is `NO`. If you set the selection state to `YES` through this property, the transition to the new state appearance is not animated. For animated selected-state transitions, see the `setSelected:animated:` (page 396) method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property selectionStyle` (page 391)

Declared In

`UITableViewCell.h`

selectedBackgroundView

The view used as the background of the cell when it is selected.

```
@property(nonatomic, retain) UIView *selectedBackgroundView
```

Discussion

The default is `nil` for cells in plain-style tables (`UITableViewStylePlain` (page 378)) and non-`nil` for section-group tables `UITableViewStyleGrouped` (page 378)). `UITableViewCell` adds the value of this property as a subview only when the cell is selected. It adds the selected background view as a subview directly above the background view (`backgroundView` (page 385)) if it is not `nil`, or behind all other views. Calling `setSelected:animated:` (page 396) causes the selected background view to animate in and out with an alpha fade.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property backgroundView` (page 385)

Declared In

`UITableViewCell.h`

selectedImage

The image to use a cell content when the cell is selected.

```
@property(nonatomic, retain) UIImage *selectedImage
```

Discussion

The default value of this property is `nil` (no image).

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property image` (page 388)

Declared In
`UITableViewCell.h`

selectedTextColor

The color of the title text when the cell is selected.

```
@property(nonatomic, retain) UIColor *selectedTextColor
```

Discussion
If the value of property is `nil` (the default), the color of text in a selected cell is white.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

selectionStyle

The style of selection for a cell.

```
@property(nonatomic) UITableViewCellSelectionStyle selectionStyle
```

Discussion
The selection style is a `backgroundView` (page 385) constant that determines the color of a cell when it is selected. The default value is `UITableViewCellSelectionStyleBlue` (page 396). See "Cell Selection Style" (page 396) for a description of valid constants.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property selected` (page 390)
`- setSelected:animated:` (page 396)

Declared In
`UITableViewCell.h`

shouldIndentWhileEditing

A Boolean value that controls whether the cell background is indented when the table view is in editing mode.

```
@property(nonatomic) BOOL shouldIndentWhileEditing
```

Discussion
The default value is `YES`. This property is unrelated to `indentationLevel` (page 388). The delegate can override this value in `tableView:shouldIndentWhileEditingRowAtIndexPath:` (page 594).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

showingDeleteConfirmation

Returns whether the cell is currently showing the delete-confirmation button. (read-only)

```
@property(nonatomic, readonly) BOOL showingDeleteConfirmation
```

Discussion
When users tap the deletion control (the red circle to the left of the cell), the cell displays a "Delete" button on the right side of the cell; this string is localized.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

showsReorderControl

A Boolean value that determines whether the cell shows the reordering control.

```
@property(nonatomic) BOOL showsReorderControl
```

Discussion
The reordering control is gray, multiple horizontal bar control on the right side of the cell. Users can drag this control to reorder the cell within the table. The default value is `NO`. If the value is `YES` , the reordering control temporarily replaces any accessory view.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

target

The target object to receive action messages.

```
@property(nonatomic, assign) id target
```

Discussion
The target object receives action messages when the user taps a cell's insert button, delete button, or accessory view. The default value is `nil`, which tells the application to go up the responder chain to find a target. Note that the target is a weak reference.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property editAction` (page 386)
`@property accessoryAction` (page 384)

Declared In
`UITableViewCell.h`

text

The text of the cell.

```
@property(nonatomic, copy) NSString *text
```

Discussion
The default is `nil` (no text).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

textAlignment

A constant that specifies the alignment of text in the cell.

```
@property(nonatomic) UITextAlignment textAlignment
```

Discussion
If the value of the property is `nil` (the default), the title is left-aligned (`UITextAlignmentLeft` (page 51)). See the descriptions of the `UITextAlignment` (page 51) constants for alternative text alignments.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

textColor

The color of the title text.

```
@property(nonatomic, retain) UIColor *textColor
```

Discussion
If the value of property is `nil` (the default), the color of text is black.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewCell.h`

Instance Methods

initWithFrame:reuseIdentifier:

Initializes and returns a table-view cell object.

```
- (id)initWithFrame:(CGRect)frame reuseIdentifier:(NSString *)reuseIdentifier
```

Parameters

frame
The frame rectangle of the cell. Because the table view automatically positions the cell and makes it the optimal size, you can pass in `CGRectZero` in most cases. However, if you have a custom cell with multiple subviews, each with its own autoresizing mask, you must specify a non-zero frame rectangle; this allows the table view to position the subviews automatically as the cell changes size.

reuseIdentifier
A string used to identify the cell object if it is to be reused for drawing multiple rows of a table view. Pass `nil` if the cell object is not to be reused.

Return Value
An initialized `UITableViewCell` object or `nil` if the object could not be created.

Discussion
This method is the designated initializer for the class. The reuse identifier is associated with those cells (rows) of a table view that have the same general configuration, minus cell content. In its implementation of `tableView:cellForRowAtIndexPath:` (page 581), the table view's delegate calls the `UITableView` method `dequeueReusableCellWithIdentifier:` (page 367), passing in a reuse identifier, to obtain the cell object to use as the basis for the current row.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property reuseIdentifier` (page 389)

Declared In
`UITableViewCell.h`

prepareForReuse

Prepares a reusable cell for reuse by the table view's delegate.

```
- (void)prepareForReuse
```

Discussion

If a `UITableViewCell` object is reusable—that is, it has a reuse identifier—this method is invoked just before the object is returned from the `UITableView` method `dequeueReusableCellWithIdentifier:` (page 367). For performance reasons, you should only reset attributes of the cell that are not related to content, for example, alpha, editing, and selection state. The table view's delegate in `tableView:cellForRowAtIndexPath:` (page 581) should *always* reset all content when reusing a cell. If the cell object does not have an associated reuse identifier, this method is not called. If you override this method, you must be sure to invoke the superclass implementation.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithFrame:reuseIdentifier:` (page 394)
`@property reuseIdentifier` (page 389)

Declared In

`UITableViewCell.h`

setEditing:animated:

Toggles the receiver into and out of editing mode.

```
- (void)setEditing:(BOOL)editing animated:(BOOL)animated
```

Parameters

editing

`YES` to enter editing mode, `NO` to leave it. The default value is `NO` .

animated

`YES` to animate the appearance or disappearance of the insertion/deletion control and the reordering control, `NO` to make the transition immediate.

Discussion

When you call this method with the value of *editing* set to `YES`, and the `UITableViewCell` object is configured to have controls, the cell shows an insertion (green plus) or deletion control (red minus) on the left side of each cell and a reordering control on the right side. This method is called on each visible cell when the `setEditing:animated:` (page 377) method of `UITableView` is invoked. Tapping a control sends an action message to a target object, which carries out the insertion or deletion. Calling this method with *editing* set to `NO` removes the controls from the cell.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property editing` (page 386)

Declared In

`UITableViewCell.h`

setSelected:animated:

Sets the selected state of the cell, optionally animating the transition between states.

```
- (void)setSelected:(BOOL)selected animated:(BOOL)animated
```

Parameters

selected

YES to set the cell as selected, NO to set it as unselected. The default is NO.

animated

YES to animate the transition between selected states, NO to make the transition immediate.

Discussion

When the the selected state of a cell to YES, it draws the background for selected cells ("Reusing Cells" (page 382)) with its title in white.

Availability

Available in iPhone OS 2.0 and later.

See Also

@property selected (page 390)
@property selectionStyle (page 391)

Declared In

UITableViewCell.h

Constants

Cell Selection Style

The style of selected cells.

```
typedef enum {
    UITableViewCellSelectionStyleNone,
    UITableViewCellSelectionStyleBlue,
    UITableViewCellSelectionStyleGray
} UITableViewCellSelectionStyle;
```

Constants

UITableViewCellSelectionStyleNone

The cell has no distinct style for when it is selected.

Available in iPhone OS 2.0 and later.

Declared in UITableViewCell.h

UITableViewCellSelectionStyleBlue

The cell when selected has a blue background. This is the default value.

Available in iPhone OS 2.0 and later.

Declared in UITableViewCell.h

`UITableViewCellSelectionStyleGray`
Then cell when selected has a gray background.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

Discussion
You use these constants to to set the value of the `selectionStyle` (page 391) property.

Declared In
`UITableViewCell.h`

Cell Editing Style

The editing control used by a cell.

```
typedef enum {
    UITableViewCellEditingStyleNone,
    UITableViewCellEditingStyleDelete,
    UITableViewCellEditingStyleInsert
} UITableViewCellEditingStyle;
```

Constants
`UITableViewCellEditingStyleNone`
The cell has no editing control. This is the default value.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

`UITableViewCellEditingStyleDelete`
The cell has the delete editing control; this control is a red circle enclosing a minus sign.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

`UITableViewCellEditingStyleInsert`
The cell has the insert editing control; this control is a green circle enclosing a plus sign.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

Discussion
You use them to to set the value of the `editingStyle` (page 387) property.

Declared In
`UITableViewCell.h`

Cell Accessory Type

The type of standard accessory control used by a cell.

```
typedef enum {
    UITableViewCellAccessoryNone,
    UITableViewCellAccessoryDisclosureIndicator,
    UITableViewCellAccessoryDetailDisclosureButton,
    UITableViewCellAccessoryCheckmark
} UITableViewCellAccessoryType;
```

Constants

`UITableViewCellAccessoryNone`

The cell does not have any accessory view. This is the default value.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

`UITableViewCellAccessoryDisclosureIndicator`

The cell has an accessory control shaped like a regular chevron. It is intended as a disclosure indicator. The control doesn't track touches.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

`UITableViewCellAccessoryDetailDisclosureButton`

The cell has an accessory control that is a blue button with a chevron image as content. It is intended for configuration purposes. The control tracks touches.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

`UITableViewCellAccessoryCheckmark`

The cell has a check mark on its right side. This control doesn't track touches. The delegate of the table view can manage check marks in a section of rows (possibly limiting the check mark to one row of the section) in its `tableView:didSelectRowAtIndexPath:` (page 590) method.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

Discussion

You use these constants when setting the value of the `accessoryType` (page 384) property.

Declared In

`UITableViewCell.h`

Cell Separator Style

The style for cells used as separators.

```
typedef enum {
    UITableViewCellSeparatorStyleNone,
    UITableViewCellSeparatorStyleSingleLine
} UITableViewCellSeparatorStyle;
```

Constants

`UITableViewCellSeparatorStyleNone`

The separator cell has no distinct style.

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

`UITableViewCellSeparatorStyleSingleLine`
The separator cell has a single line running across its width. This is the default value

Available in iPhone OS 2.0 and later.

Declared in `UITableViewCell.h`

Discussion
You use these constants to to set the value of the `separatorStyle` property defined by `UITableView`.

Declared In
`UITableViewCell.h`

UITableViewController Class Reference

Inherits from:	UIViewController : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITableViewController.h
Companion guide:	Table View Programming Guide for iPhone OS

Overview

The `UITableViewController` class creates a controller object that manages a table view. It implements the following behavior:

- If a nib file is specified via the `initWithNibName:bundle:` method (which is declared by the superclass `UIViewController`), `UITableViewController` loads the table view archived in the nib file. Otherwise, it creates an unconfigured `UITableView` object with the correct dimensions and autoresize mask. You can access this view through the `tableView` (page 402) property.
- If a nib file containing the table view is loaded, the data source and delegate become those objects defined in the nib file (if any). If no nib file is specified or if the nib file defines no data source or delegate, `UITableViewController` sets the data source and the delegate of the table view to `self`.
- When the table view is about to appear, it reloads its data and clears its selection (with or without animation, depending on the request). The `UITableViewController` class implements this in the superclass method `viewWillAppear:` (page 499).
- When the table view has appeared, the controller flashes the table view's scroll indicators. The `UITableViewController` class implements this in the superclass method `viewDidAppear:` (page 497).
- It implements the superclass method `setEditing:animated:` (page 496) so that if a user taps an Edit | Done button in the navigation bar, the controller toggles the edit mode of the table.

You create a custom subclass of `UITableViewController` for each table view that you want to manage. When you initialize the controller in `initWithStyle:` (page 402), you must specify the style of the table view (plain or grouped) that the controller is to manage. Because the initially created table view is without table dimensions (that is, number of sections and number of rows per section) or content, the table view's data source and delegate—that is, the `UITableViewController` object itself—must provide the table dimensions, the cell content, and any desired configurations (as usual). You may override `loadView` (page 494) or any other superclass method, but if you do be sure to invoke the superclass implementation of the method, usually as the first method call.

Tasks

Initializing the UITableViewController Object

`- initWithStyle:` (page 402)
Initializes a table-view controller to manage a table view of a given style.

Getting the Table View

`tableView` (page 402) *property*
Returns the table view managed by the controller object.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

tableView

Returns the table view managed by the controller object.

```
@property(nonatomic, retain) UITableView *tableView
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableViewController.h`

Instance Methods

initWithStyle:

Initializes a table-view controller to manage a table view of a given style.

```
- (id)initWithStyle:(UITableViewStyle)style
```

Parameters

style

A constant that specifies the style of table view that the controller object is to manage (`UITableViewStylePlain` (page 378) or `UITableViewStyleGrouped` (page 378)).

Return Value

An initialized `UITableViewController` object or `nil` if the object couldn't be created.

Discussion

If you use the standard `init` method to initialize a `UITableViewController` object, a table view in the plain style is created.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableViewController.h`

UITextField Class Reference

Inherits from:	UIControl : UIView : UIResponder : NSObject
Conforms to:	UITextInputTraits NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITextField.h

Overview

A `UITextField` object is a control that displays editable text and sends an action message to a target object when the user presses the return button. You typically use this class to gather small amounts of text from the user and perform some immediate action, such as a search operation, based on that text.

In addition to its basic text-editing behavior, the `UITextField` class supports the use of overlay views to display additional information (and provide additional command targets) inside the text field boundaries. You can use custom overlay views to display features such as a bookmarks button or search icon. The `UITextField` class also provides a built-in button for clearing the current text.

A text field object supports the use of a delegate object to handle editing-related notifications. You can use this delegate to customize the editing behavior of the control and provide guidance for when certain actions should occur. For more information on the methods supported by the delegate, see the `UITextFieldDelegate` protocol.

Managing the Keyboard

When the user taps in a text field, that text field becomes the first responder and automatically asks the system to display the associated keyboard. Because the appearance of the keyboard has the potential to obscure portions of your user interface, it is up to you to make sure that does not happen by repositioning any views that might be obscured. Some system views, like table views, help you

by scrolling the first responder into view automatically. If the first responder is at the bottom of the scrolling region, however, you may still need to resize or reposition the scroll view itself to ensure the first responder is visible.

It is your application's responsibility to dismiss the keyboard at the time of your choosing. You might dismiss the keyboard in response to a specific user action, such as the user tapping a particular button in your user interface. You might also configure your text field delegate to dismiss the keyboard when the user presses the "return" key on the keyboard itself. To dismiss the keyboard, send the `resignFirstResponder` (page 282) message to the text field that is currently the first responder. Doing so causes the text field object to end the current editing session (with the delegate object's consent) and hide the keyboard.

The appearance of the keyboard itself can be customized using the properties provided by the `UITextInputTraits` protocol. Text field objects implement this protocol and support the properties it defines. You can use these properties to specify the type of keyboard (ASCII, Numbers, URL, Email, and others) to display. You can also configure the basic text entry behavior of the keyboard, such as whether it supports automatic capitalization and correction of the text.

Keyboard Notifications

When the system shows or hides the keyboard, it posts several keyboard notifications. These notifications contain information about the keyboard, including its size, which you can use for calculations that involve moving views. Registering for these notifications is the only way to get some types of information about the keyboard. The system delivers the following notifications for keyboard-related events:

- `UIKeyboardWillShowNotification` (page 521)
- `UIKeyboardDidShowNotification` (page 522)
- `UIKeyboardWillHideNotification` (page 522)
- `UIKeyboardDidHideNotification` (page 522)

For more information about these notifications, see their descriptions in *UIWindow Class Reference*.

Tasks

Accessing the Text Attributes

`text` (page 414) *property*
The text displayed by the text field.

`placeholder` (page 413) *property*
The string that is displayed when there is no other text in the text field.

`font` (page 411) *property*
The font of the text.

`textColor` (page 414) *property*
The color of the text.

`textAlignment` (page 414) *property*
The technique to use for aligning the text.

Sizing the Text Field's Text

`adjustsFontSizeToFitWidth` (page 408) *property*
A Boolean value indicating whether the font size should be reduced in order to fit the text string into the text field's bounding rectangle.

`minimumFontSize` (page 412) *property*
The size of the smallest permissible font with which to draw the text field's text.

Managing the Editing Behavior

`editing` (page 411) *property*
A Boolean value indicating whether the text field is currently in edit mode. (read-only)

`clearsOnBeginEditing` (page 410) *property*
A Boolean value indicating whether the text field removes old text when editing begins.

Setting the View's Background Appearance

`borderStyle` (page 409) *property*
The border style used by the text field.

`background` (page 409) *property*
The image that represents the background appearance of the text field when it is enabled.

`disabledBackground` (page 411) *property*
The image that represents the background appearance of the text field when it is disabled.

Managing Overlay Views

`clearButtonMode` (page 409) *property*
Controls when the standard clear button appears in the text field.

`leftView` (page 412) *property*
The overlay view displayed on the left side of the text field.

`leftViewMode` (page 412) *property*
Controls when the left overlay view appears in the text field.

`rightView` (page 413) *property*
The overlay view displayed on the right side of the text field.

`rightViewMode` (page 414) *property*
Controls when the left overlay view appears in the text field.

Accessing the Delegate

`delegate` (page 410) *property*
The receiver's delegate.

Drawing and Positioning Overrides

- `textRectForBounds:` (page 418)
  Returns the drawing rectangle for the text field's text.
- `drawTextInRect:` (page 416)
  Draws the receiver's text in the specified rectangle.
- `placeholderRectForBounds:` (page 418)
  Returns the drawing rectangle for the text field's placeholder text
- `drawPlaceholderInRect:` (page 416)
  Draws the receiver's placeholder text in the specified rectangle.
- `borderRectForBounds:` (page 415)
  Returns the receiver's border rectangle.
- `editingRectForBounds:` (page 417)
  Returns the rectangle in which editable text can be displayed.
- `clearButtonRectForBounds:` (page 415)
  Returns the drawing rectangle for the built-in clear button.
- `leftViewRectForBounds:` (page 417)
  Returns the drawing rectangle of the receiver's left overlay view.
- `rightViewRectForBounds:` (page 418)
  Returns the drawing location of the receiver's right overlay view.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

adjustsFontSizeToFitWidth

A Boolean value indicating whether the font size should be reduced in order to fit the text string into the text field's bounding rectangle.

```
@property(nonatomic) BOOL adjustsFontSizeToFitWidth
```

Discussion

Normally, the text field's content is drawn with the font you specify in the `font` property. If this property is set to `YES`, however, and the contents in the `text` property exceed the text field's bounding rectangle, the receiver starts reducing the font size until the string fits or the minimum font size is reached. The text is shrunk along the baseline.

The default value for this property is `NO`. If you change it to `YES`, you should also set an appropriate minimum font size by modifying the `minimumFontSize` property.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property minimumFontSize` (page 412)

Declared In
`UITextField.h`

background
The image that represents the background appearance of the text field when it is enabled.

```
@property(nonatomic, retain) UIImage *background
```

Discussion
When set, the image referred to by this property replaces the standard appearance controlled by the `borderStyle` property. Background images are drawn in the border rectangle portion of the image. Images you use for the text field's background should be able to stretch to fit.

This property is set to `nil` by default.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property borderStyle` (page 409)
`@property disabledBackground` (page 411)

Declared In
`UITextField.h`

borderStyle
The border style used by the text field.

```
@property(nonatomic) UITextBorderStyle borderStyle
```

Discussion
The default value for this property is `UITextBorderStyleNone`. If a custom background image is set, this property is ignored.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

clearButtonMode
Controls when the standard clear button appears in the text field.

```
@property(nonatomic) UITextFieldViewMode clearButtonMode
```

Discussion
The standard clear button is displayed at the right side of the text field as a way for the user to remove text quickly. This button appears automatically based on the value set for this property.

The default value for this property is `UITextFieldViewModeNever`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

clearsOnBeginEditing

A Boolean value indicating whether the text field removes old text when editing begins.

```
@property(nonatomic) BOOL clearsOnBeginEditing
```

Discussion
If this property is set to `YES`, the text field's previous text is cleared when the user selects the text field to begin editing. If `NO`, the text field places an insertion point at the place where the user tapped the field.

Note: Even if this property is set to `YES`, the text field delegate can override this behavior by returning `NO` from its `textFieldShouldClear:` (page 602) method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

delegate

The receiver's delegate.

```
@property(nonatomic, assign) id<UITextFieldDelegate> delegate
```

Discussion
A text field delegate responds to editing-related messages from the text field. You can use the delegate to respond to the text entered by the user and to some special commands, such as when the return button is pressed.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

disabledBackground

The image that represents the background appearance of the text field when it is disabled.

```
@property(nonatomic, retain) UIImage *disabledBackground
```

Discussion

Background images are drawn in the border rectangle portion of the image. Images you use for the text field's background should be able to stretch to fit. This property is ignored if the `background` property is not also set.

This property is set to `nil` by default.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property background` (page 409)

Declared In

`UITextField.h`

editing

A Boolean value indicating whether the text field is currently in edit mode. (read-only)

```
@property(nonatomic, readonly, getter=isEditing) BOOL editing
```

Discussion

This property is set to `YES` when the user begins editing text in this text field, and it is set to `NO` again when editing ends. Notifications about when editing begins and ends are sent to the text field delegate.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextField.h`

font

The font of the text.

```
@property(nonatomic, retain) UIFont *font
```

Discussion

This property applies to the entire text of the text field. It also applies to the placeholder text. The default font is a 12-point Helvetica plain font.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextField.h`

leftView

The overlay view displayed on the left side of the text field.

```
@property(nonatomic, retain) UIView *leftView
```

Discussion

You can use the left overlay view to indicate the intended behavior of the text field. For example, you might display a magnifying glass in this location to indicate that the text field is a search field.

The left overlay view is placed in the rectangle returned by the `leftViewRectForBounds:` method of the receiver. The image associated with this property should fit the given rectangle. If it does not fit, it is scaled to fit.

If your overlay view does not overlap any other sibling views, it receives touch events like any other view. If you specify a control for your view, the control tracks and sends actions as usual. If an overlay view overlaps the clear button, however, the clear button always takes precedence in receiving events.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- leftViewRectForBounds:` (page 417)

Declared In

`UITextField.h`

leftViewMode

Controls when the left overlay view appears in the text field.

```
@property(nonatomic) UITextFieldViewMode leftViewMode
```

Discussion

The default value for this property is `UITextFieldViewModeNever`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextField.h`

minimumFontSize

The size of the smallest permissible font with which to draw the text field's text.

```
@property(nonatomic) CGFloat minimumFontSize
```

Discussion

When drawing text that might not fit within the bounding rectangle of the text field, you can use this property to prevent the receiver from reducing the font size to the point where it is no longer legible.

The default value for this property is 0.0. If you enable font adjustment for the text field, you should always increase this value.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property adjustsFontSizeToFitWidth` (page 408)

Declared In
`UITextField.h`

placeholder

The string that is displayed when there is no other text in the text field.

```
@property(nonatomic, copy) NSString *placeholder
```

Discussion
This value is `nil` by default. The placeholder string is drawn using a 70% grey color.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

rightView

The overlay view displayed on the right side of the text field.

```
@property(nonatomic, retain) UIView *rightView
```

Discussion
You can use the right overlay view to provide indicate additional features available for the text field. For example, you might display a bookmarks button in this location to allow the user to select from a set of predefined items.

The left overlay view is placed in the rectangle returned by the `rightViewRectForBounds:` method of the receiver. The image associated with this property should fit the given rectangle. If it does not fit, it is scaled to fit.

If your overlay view does not overlap any other sibling views, it receives touch events like any other view. If you specify a control for your view, that control tracks and sends actions as usual. If an overlay view overlaps the clear button, however, the clear button always takes precedence in receiving events. By default, the right overlay view does overlap the clear button.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- rightViewRectForBounds:` (page 418)

Declared In
`UITextField.h`

rightViewMode

Controls when the left overlay view appears in the text field.

```
@property(nonatomic) UITextFieldViewMode rightViewMode
```

Discussion
The default value for this property is `UITextFieldViewModeNever`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

text

The text displayed by the text field.

```
@property(nonatomic, copy) NSString *text
```

Discussion
This string is `nil` by default.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

textAlignment

The technique to use for aligning the text.

```
@property(nonatomic) UITextAlignment textAlignment
```

Discussion
This property applies to the both the main text string and the placeholder string. The default value of this property is `UITextAlignmentLeft` (page 51).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

textColor

The color of the text.

```
@property(nonatomic, retain) UIColor *textColor
```

Discussion
This property applies to the entire text string. The default value for this property is `nil`, which results in opaque black text.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

Instance Methods

borderRectForBounds:

Returns the receiver's border rectangle.

```
- (CGRect)borderRectForBounds:(CGRect)bounds
```

Parameters
bounds
The bounding rectangle of the receiver.

Return Value
The border rectangle for the receiver.

Discussion
You should not call this method directly. If you want to provide a different border rectangle for drawing, you can override this method and return that rectangle.

The default implementation of this method returns the original *bounds* rectangle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

clearButtonRectForBounds:

Returns the drawing rectangle for the built-in clear button.

```
- (CGRect)clearButtonRectForBounds:(CGRect)bounds
```

Parameters
bounds
The bounding rectangle of the receiver.

Return Value
The rectangle in which to draw the clear button.

Discussion
You should not call this method directly. If you want to place the clear button in a different location, you can override this method and return the new rectangle. Your method should call the `super` implementation and modify the returned rectangle's origin only. Changing the size of the clear button may cause unnecessary distortion of the button image.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

drawPlaceholderInRect:

Draws the receiver's placeholder text in the specified rectangle.

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

Parameters

rect
The rectangle in which to draw the placeholder text.

Discussion
You should not call this method directly. If you want to customize the drawing behavior for the placeholder text, you can override this method to do your drawing.

By the time this method is called, the current graphics context is already configured with the default environment and text color for drawing. In your overridden method, you can configure the current context further and then invoke `super` to do the actual drawing or do the drawing yourself. If you do render the text yourself, you should not invoke `super`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

drawTextInRect:

Draws the receiver's text in the specified rectangle.

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

Parameters

rect
The rectangle in which to draw the text.

Discussion
You should not call this method directly. If you want to customize the drawing behavior for the text, you can override this method to do your drawing.

By the time this method is called, the current graphics context is already configured with the default environment and text color for drawing. In your overridden method, you can configure the current context further and then invoke `super` to do the actual drawing or you can do the drawing yourself. If you do render the text yourself, you should not invoke `super`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

editingRectForBounds:

Returns the rectangle in which editable text can be displayed.

```
- (CGRect)editingRectForBounds:(CGRect)bounds
```

Parameters

bounds

The bounding rectangle of the receiver.

Return Value
The computed editing rectangle for the text.

Discussion
You should not call this method directly. If you want to provide a different editing rectangle for the text, you can override this method and return that rectangle. By default, this method returns a region in the text field that is not occupied by any overlay views.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

leftViewRectForBounds:

Returns the drawing rectangle of the receiver's left overlay view.

```
- (CGRect)leftViewRectForBounds:(CGRect)bounds
```

Parameters

bounds

The bounding rectangle of the receiver.

Return Value
The rectangle in which to draw the left overlay view.

Discussion
You should not call this method directly. If you want to place the left overlay view in a different location, you can override this method and return the new rectangle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

placeholderRectForBounds:

Returns the drawing rectangle for the text field's placeholder text

```
- (CGRect)placeholderRectForBounds:(CGRect)bounds
```

Parameters

bounds

The bounding rectangle of the receiver.

Return Value
The computed drawing rectangle for the placeholder text.

Discussion
You should not call this method directly. If you want to customize the drawing rectangle for the placeholder text, you can override this method and return a different rectangle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

rightViewRectForBounds:

Returns the drawing location of the receiver's right overlay view.

```
- (CGRect)rightViewRectForBounds:(CGRect)bounds
```

Parameters

bounds

The bounding rectangle of the receiver.

Return Value
The rectangle in which to draw the right overlay view.

Discussion
You should not call this method directly. If you want to place the right overlay view in a different location, you can override this method and return the new rectangle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

textRectForBounds:

Returns the drawing rectangle for the text field's text.

```
- (CGRect)textRectForBounds:(CGRect)bounds
```

Parameters

bounds

The bounding rectangle of the receiver.

Return Value

The computed drawing rectangle for the label's text.

Discussion

You should not call this method directly. If you want to customize the drawing rectangle for the text, you can override this method and return a different rectangle.

The default implementation of this method returns a rectangle that is derived from the control's original bounds, but which does not include the area occupied by the receiver's border or overlay views.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextField.h`

Constants

UITextFieldBorderStyle

The type of border drawn around the text field.

```
typedef enum {
    UITextBorderStyleNone,
    UITextBorderStyleLine,
    UITextBorderStyleBezel,
    UITextBorderStyleRoundedRect
} UITextBorderStyle;
```

Constants

`UITextBorderStyleNone`

The text field does not display a border.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

`UITextBorderStyleLine`

Displays a thin rectangle around the text field.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

`UITextBorderStyleBezel`

Displays a bezel-style border for the text field. This style is typically used for standard data-entry fields.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

`UITextBorderStyleRoundedRect`
Displays a rounded-style border for the text field. This style is typically used for search buttons.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

UITextFieldViewMode

Defines the times at which overlay views appear in a text field.

```
typedef enum {
    UITextFieldViewModeNever,
    UITextFieldViewModeWhileEditing,
    UITextFieldViewModeUnlessEditing,
    UITextFieldViewModeAlways
} UITextFieldViewMode;
```

Constants

`UITextFieldViewModeNever`
The overlay view never appears.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

`UITextFieldViewModeWhileEditing`
The overlay view is displayed only while text is being edited in the text field.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

`UITextFieldViewModeUnlessEditing`
The overlay view is displayed only when text is not being edited.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

`UITextFieldViewModeAlways`
The overlay view is always displayed.

Available in iPhone OS 2.0 and later.

Declared in `UITextField.h`

Notifications

UITextFieldTextDidBeginEditingNotification

Notifies observers that an editing session began in a text field. The affected text field is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextField.h`

UITextFieldTextDidChangeNotification

Notifies observers that the text in a text field changed. The affected text field is stored in the `object` parameter of the notification.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

UITextFieldTextDidEndEditingNotification

Notifies observers that the editing session ended for a text field. The affected text field is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

UITextView Class Reference

Inherits from:	UIScrollView : UIView : UIResponder : NSObject
Conforms to:	UITextInputTraits NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITextView.h

Overview

The `UITextView` class implements the behavior for a scrollable, multiline text region. The class supports the display of text using a custom font, color, and alignment and also supports text editing. You typically use a text view to display multiple lines of text, such as when displaying the body of a large text document.

This class does not support multiple styles for text. The font, color, and text alignment attributes you specify always apply to the entire contents of the text view. To display more complex styling in your application, you need to use a `UIWebView` object and render your content using HTML.

Managing the Keyboard

When the user taps in an editable text view, that text view becomes the first responder and automatically asks the system to display the associated keyboard. Because the appearance of the keyboard has the potential to obscure portions of your user interface, it is up to you to make sure that does not happen by repositioning any views that might be obscured. Some system views, like table views, help you by scrolling the first responder into view automatically. If the first responder is at the bottom of the scrolling region, however, you may still need to resize or reposition the scroll view itself to ensure the first responder is visible.

It is your application's responsibility to dismiss the keyboard at the time of your choosing. You might dismiss the keyboard in response to a specific user action, such as the user tapping a particular button in your user interface. To dismiss the keyboard, send the `resignFirstResponder` (page 282) message to the text view that is currently the first responder. Doing so causes the text view object to end the current editing session (with the delegate object's consent) and hide the keyboard.

The appearance of the keyboard itself can be customized using the properties provided by the `UITextInputTraits` protocol. Text view objects implement this protocol and support the properties it defines. You can use these properties to specify the type of keyboard (ASCII, Numbers, URL, Email, and others) to display. You can also configure the basic text entry behavior of the keyboard, such as whether it supports automatic capitalization and correction of the text.

Keyboard Notifications

When the system shows or hides the keyboard, it posts several keyboard notifications. These notifications contain information about the keyboard, including its size, which you can use for calculations that involve repositioning or resizing views. Registering for these notifications is the only way to get some types of information about the keyboard. The system delivers the following notifications for keyboard-related events:

- `UIKeyboardWillShowNotification` (page 521)
- `UIKeyboardDidShowNotification` (page 522)
- `UIKeyboardWillHideNotification` (page 522)
- `UIKeyboardDidHideNotification` (page 522)

For more information about these notifications, see their descriptions in *UIWindow Class Reference*.

Tasks

Configuring the Text Attributes

`text` (page 426) *property*
The text displayed by the text view.

`font` (page 426) *property*
The font of the text.

`textColor` (page 427) *property*
The color of the text.

`editable` (page 425) *property*
A Boolean value indicating whether the receiver is editable.

`textAlignment` (page 426) *property*
The technique to use for aligning the text.

- `hasText` (page 427)
Returns a Boolean value indicating whether the text view currently contains any text.

Working with the Selection

`selectedRange` (page 426) *property*
The current selection range of the receiver.

`- scrollRangeToVisible:` (page 427)
Scrolls the receiver until the text in the specified range is visible.

Accessing the Delegate

`delegate` (page 425) *property*
The receiver's delegate.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

delegate

The receiver's delegate.

```
@property(nonatomic, assign) id<UITextViewDelegate> delegate
```

Discussion
A text view delegate responds to editing-related messages from the text view. You can use the delegate to track changes to the text itself and to the current selection.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextView.h`

editable

A Boolean value indicating whether the receiver is editable.

```
@property(nonatomic, getter=isEditable) BOOL editable
```

Discussion
The default value of this property is `YES`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextView.h`

font

The font of the text.

```
@property(nonatomic, retain) UIFont *font
```

Discussion
This property applies to the entire text string. The default font is a 17-point Helvetica plain font.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

selectedRange

The current selection range of the receiver.

```
@property(nonatomic) NSRange selectedRange
```

Discussion
The length of the selection range is always 0, indicating that the selection is actually an insertion point.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

text

The text displayed by the text view.

```
@property(nonatomic, copy) NSString *text
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

textAlignment

The technique to use for aligning the text.

```
@property(nonatomic) UITextAlignment textAlignment
```

Discussion
This property applies to the entire text string. The default value of this property is `UITextAlignmentLeft` (page 51).

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

textColor

The color of the text.

```
@property(nonatomic, retain) UIColor *textColor
```

Discussion
This property applies to the entire text string. The default text color is black.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property backgroundColor (page 446) (UIView)

Declared In
UITextView.h

Instance Methods

hasText

Returns a Boolean value indicating whether the text view currently contains any text.

```
- (BOOL)hasText
```

Return Value
YES if the receiver contains text or NO if it does not.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

scrollRangeToVisible:

Scrolls the receiver until the text in the specified range is visible.

```
- (void)scrollRangeToVisible:(NSRange)range
```

Parameters
range
The range to scroll into view. The length of the range is ignored.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

Notifications

UITextViewTextDidBeginEditingNotification

Notifies observers that an editing session began in a text view. The affected view is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

UITextViewTextDidChangeNotification

Notifies observers that the text in a text view changed. The affected view is stored in the `object` parameter of the notification.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

UITextViewTextDidEndEditingNotification

Notifies observers that the editing session ended for a text view. The affected view is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

UIToolbar Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIToolbar.h

Overview

The `UIToolbar` class implements a control for selecting one of many buttons, called toolbar items. A toolbar momentarily highlights or does not change the appearance of an item when tapped. Use the `UITabBar` class if you need a radio button style control.

Use the `UIBarButtonItem` class to create items and the `setItems:animated:` (page 431) method to add them to a toolbar. All methods with an `animated:` argument allow you to optionally animate changes to the display.

Note that the images used on the toolbar to represent the normal and highlighted states of an item are derived from the image you set using the inherited `image` (page 118) property in `UIBarItem`. For example, the image is converted to white and then bevelled by adding a shadow for the normal state.

Tasks

Getting and Setting Properties

`barStyle` (page 430) *property*
The toolbar style that specifies its appearance.

`tintColor` (page 431) *property*
The color used to tint the bar.

Configuring Items

`items` (page 430) *property*
The items displayed on the toolbar.

`- setItems:animated:` (page 431)
Sets the items on the toolbar by animating the changes.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

barStyle

The toolbar style that specifies its appearance.

```
@property(nonatomic) UIBarStyle barStyle
```

Discussion

See `UIBarStyle` (page 625) for possible values. The default value is `UIBarStyleDefault`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIToolbar.h`

items

The items displayed on the toolbar.

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

Discussion

The items, instances of `UIBarButtonItem`, that are visible on the toolbar in the order they appear in this array. Any changes to this property are not animated. Use the `setItems:animated:` (page 431) method to animate changes.

The default value is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setItems:animated:` (page 431)

Declared In

`UIToolbar.h`

tintColor

The color used to tint the bar.

```
@property(nonatomic, retain) UIColor *tintColor
```

Discussion
The default value is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIToolbar.h`

Instance Methods

setItems:animated:

Sets the items on the toolbar by animating the changes.

```
- (void)setItems:(NSArray *)items animated:(BOOL)animated
```

Parameters

items
The items to display on the toolbar.

animated
A Boolean value if set to `YES` animates the transition to the items; otherwise, does not.

Discussion
If *animated* is `YES`, the changes are dissolved or the reordering is animated—for example, removed items fade out and new items fade in. This method also adjusts the spacing between items.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property items` (page 430)

Declared In
`UIToolbar.h`

UITouch Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITouch.h

Overview

A `UITouch` object represents the presence or movement of a finger on the screen for a particular event. You access `UITouch` objects through `UIEvent` objects passed into responder objects for event handling.

A `UITouch` object includes methods for accessing the view or window in which the touch occurred and for obtaining the location of the touch in a specific view or window. it also lets you find out when the touch occurred, whether the user tapped more than once, whether the finger is swiped (and if so, in which direction), and the phase of a touch—that is, whether it began, moved, or ended the gesture, or whether it was canceled.

A `UITouch` object is persistent throughout a multi-touch sequence. You should never retain an `UITouch` object when handling an event. If you need to keep information about a touch from one phase to another, you should copy that information from the `UITouch` object.

See Event Handling in *iPhone OS Programming Guide* for further information on event handling.

Tasks

Getting the Location of Touches

`- locationInView:` (page 436)
Returns the current location of the receiver in the coordinate system of the given view.

- `previousLocationInView:` (page 436)
  Returns the previous location of the receiver in the coordinate system of the given view.

`view` (page 435) *property*
The view in which the touch initially occurred. (read-only)

`window` (page 435) *property*
The window in which the touch initially occurred. (read-only)

Getting Touch Attributes

`tapCount` (page 434) *property*
The number of times the finger was tapped for this given touch. (read-only)

`timestamp` (page 435) *property*
The time when the touch occurred or when it was last mutated. (read-only)

`phase` (page 434) *property*
The type of touch. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

phase

The type of touch. (read-only)

```
@property(nonatomic, readonly) UITouchPhase phase
```

Discussion

The property value is a constant that indicates whether the touch began, moved, ended, or was canceled. For descriptions of possible `UITouchPhase` values, see "Touch Phase" (page 437).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITouch.h`

tapCount

The number of times the finger was tapped for this given touch. (read-only)

```
@property(nonatomic, readonly) NSUInteger tapCount
```

Discussion

The value of this property is an integer indicating the number of times the user tapped their fingers on a certain point within a predefined period. If want to determine whether the user single-tapped, double-tapped, or event triple-tapped a particular view or window, you should evaluate the value returned by this method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITouch.h

timestamp

The time when the touch occurred or when it was last mutated. (read-only)

```
@property(nonatomic, readonly) NSTimeInterval timestamp
```

Discussion
The value of this property is the time, in seconds, since system startup the touch either originated or was last changed. You can store and compare the initial value of this attribute to subsequent timestamp values of the UITouch instance to determine the duration of the touch and, if it is being swiped, the speed of movement.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITouch.h

view

The view in which the touch initially occurred. (read-only)

```
@property(nonatomic, readonly, retain) UIView *view
```

Discussion
The value of the property is the view object in which the touch originally occurred. This object might not be the view the touch is currently in.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property window (page 435)

Declared In
UITouch.h

window

The window in which the touch initially occurred. (read-only)

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

Discussion
The value of the property is the window object in which the touch originally occurred. This object might not be the window the touch is currently in.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property view` (page 435)

Declared In
`UITouch.h`

Instance Methods

locationInView:

Returns the current location of the receiver in the coordinate system of the given view.

```
- (CGPoint)locationInView:(UIView *)view
```

Parameters

view

The view object in whose coordinate system you want the touch located. A custom view that is handling the touch may specify `self` to get the touch location in its own coordinate system. Pass `nil` to get the touch location in the window's coordinates.

Return Value
A point specifying the location of the receiver in *view*.

Discussion
This method returns the current location of a `UITouch` object in the coordinate system of the specified view. Because the touch object might have been forwarded to a view from another view, this method performs any necessary conversion of the touch location to the coordinate system of the specified view.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- previousLocationInView:` (page 436)

Declared In
`UITouch.h`

previousLocationInView:

Returns the previous location of the receiver in the coordinate system of the given view.

```
- (CGPoint)previousLocationInView:(UIView *)view
```

Parameters

view

The view object in whose coordinate system you want the touch located. A custom view that is handling the touch may specify `self` to get the touch location in its own coordinate system. Pass `nil` to get the touch location in the window's coordinates.

Return Value

This method returns the previous location of a `UITouch` object in the coordinate system of the specified view. Because the touch object might have been forwarded to a view from another view, this method performs any necessary conversion of the touch location to the coordinate system of the specified view.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- locationInView:` (page 436)

Declared In

`UITouch.h`

Constants

Touch Phase

The phase of a finger touch.

```
typedef enum {
    UITouchPhaseBegan,
    UITouchPhaseMoved,
    UITouchPhaseStationary,
    UITouchPhaseEnded,
    UITouchPhaseCancelled,
} UITouchPhase;
```

Constants

`UITouchPhaseBegan`

A finger for a given event touched the screen.

Available in iPhone OS 2.0 and later.

Declared in `UITouch.h`

`UITouchPhaseMoved`

A finger for a given event moved on the screen.

Available in iPhone OS 2.0 and later.

Declared in `UITouch.h`

`UITouchPhaseStationary`

A finger is touching the surface but hasn't moved since the previous event.

Available in iPhone OS 2.0 and later.

Declared in `UITouch.h`

`UITouchPhaseEnded`

A finger for a given event was lifted from the screen.

Available in iPhone OS 2.0 and later.

Declared in `UITouch.h`

`UITouchPhaseCancelled`

The system cancelled tracking for the touch, as when (for example) the user puts the device to his or her face.

Available in iPhone OS 2.0 and later.

Declared in `UITouch.h`

Discussion

The phase of a `UITouch` instance changes in a certain order during the course of an event. You access this value through the `phase` (page 434) property.

Declared In

`UITouch.h`

UIView Class Reference

Inherits from:	UIResponder : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIView.h

Overview

The `UIView` class is primarily an abstract superclass that provides concrete subclasses with a structure for drawing and handling events. You can also create instances of `UIView` to contain other views.

`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.

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 469) method to create a view. The `frame` (page 449) 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 447) and `bounds` (page 446) 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 445) property, especially if you subclass `UIView`, to specify whether the

view should automatically resize its subviews. Use the `autoresizingMask` (page 446) property with the constants described in `UIViewAutoresizing` (page 480) 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 452) property to get the parent view and the `subviews` (page 451) property to get the child views in the hierarchy. There are also a number of methods, listed in "Managing the View Hierarchy" (page 442), 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 467) 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 473) or `setNeedsDisplayInRect:` (page 473) to the view. You can also set the `contentMode` (page 448) to `UIViewContentModeRedraw` (page 479) to invoke the `drawRect:` (page 467) 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 469), to create a view hierarchy. If you want to programmatically force the layout of subviews before drawing, send `setNeedsLayout` (page 474) to the view. Then when `layoutIfNeeded` (page 471) is invoked, the `layoutSubviews` (page 471) method is invoked just before displaying. Subclasses should override `layoutSubviews` (page 471) to perform any custom arrangement of subviews.

Some of the property changes to view objects can be animated—for example, setting the `frame` (page 449), `bounds` (page 446), `center` (page 447), and `transform` (page 452) 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 454) class method to begin an animation block, set the properties you want animated, and then invoke the `commitAnimations` (page 454) 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 468) and `pointInside:withEvent:` (page 472) 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*.

Read Windows and Views in *iPhone OS Programming Guide* to learn how to use this class.

Tasks

Creating Instances

- `initWithFrame:` (page 469)
  Initializes and returns a newly allocated view object with the specified frame rectangle.

Setting and Getting Attributes

`userInteractionEnabled` (page 453) *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 449) *property*
The receiver's frame rectangle.

`bounds` (page 446) *property*
The receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

`center` (page 447) *property*
The center of the frame.

`transform` (page 452) *property*
Specifies the transform applied to the receiver, relative to the center of its bounds.

Converting Coordinates

- `convertPoint:toView:` (page 464)
Converts a point from the receiver's coordinate system to that of a given view.

- `convertPoint:fromView:` (page 464)
Converts a point from the coordinate system of a given view to that of the receiver.

- `convertRect:toView:` (page 465)
Converts a rectangle from the receiver's coordinate system to that of another view.

- `convertRect:fromView:` (page 465)
Converts a rectangle from the coordinate system of another view to that of the receiver.

Resizing Subviews

`autoresizesSubviews` (page 445) *property*
A Boolean value that determines whether the receiver automatically resizes its subviews when its frame size changes.

`autoresizingMask` (page 446) *property*
An integer bit mask that determines how the receiver resizes its subviews when its bounds change.

- `sizeThatFits:` (page 474)
Calculates and returns a size that best fits the receiver's subviews.

- `sizeToFit` (page 475)
Resizes and moves the receiver view so it just encloses its subviews.

`contentMode` (page 448) *property*
A flag used to determine how a view lays out its content when its bounds rectangle changes.

Managing the View Hierarchy

`superview` (page 452) *property*
The receiver's superview, or `nil` if it has none. (read-only)

`subviews` (page 451) *property*
The receiver's immediate subviews. (read-only)

`window` (page 453) *property*
The receiver's window object, or `nil` if it has none. (read-only)

- `addSubview:` (page 463)
Adds a view to the receiver's subviews so it's displayed above its siblings.

- `bringSubviewToFront:` (page 463)
Moves the specified subview to the front of its siblings.

- `sendSubviewToBack:` (page 473)
Moves the specified subview to the back of its siblings.

- `removeFromSuperview` (page 472)
Unlinks the receiver from its superview and its window, and removes it from the responder chain.

- `insertSubview:atIndex:` (page 470)
Inserts a subview at the specified index.

- `insertSubview:aboveSubview:` (page 469)
Inserts a view above another view in the view hierarchy.

- `insertSubview:belowSubview:` (page 470)
Inserts a view below another view in the view hierarchy.

- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 468)
Exchanges the subviews in the receiver at the given indices.

- `isDescendantOfView:` (page 471)
Returns a Boolean value indicating whether the receiver is a subview of a given view or whether it is identical to that view.

Searching for Views

`tag` (page 452) *property*
The receiver's tag, an integer that you can use to identify view objects in your application.

- `viewWithTag:` (page 475)
Returns the view with the specified tag.

Laying out Views

- `setNeedsLayout` (page 474)
Sets whether subviews need to be rearranged before displaying.

- `layoutIfNeeded` (page 471)
Lays out the subviews if needed.

- `layoutSubviews` (page 471)
Lays out subviews.

Displaying

`clipsToBounds` (page 448) *property*
A Boolean value that determines whether subviews can be drawn outside the bounds of the receiver.

`backgroundColor` (page 446) *property*
The receiver's background color.

`alpha` (page 445) *property*
The receiver's alpha value.

`opaque` (page 451) *property*
A Boolean value that determines whether the receiver is opaque.

`clearsContextBeforeDrawing` (page 448) *property*
A Boolean value that determines whether the receiver's bounds should be automatically cleared before drawing.

- `drawRect:` (page 467)
Draws the receiver's image within the passed-in rectangle.

- `setNeedsDisplay` (page 473)
Controls whether the receiver's entire bounds rectangle is marked as needing display.

- `setNeedsDisplayInRect:` (page 473)
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 455)
Returns the class used to create the layer for instances of this class.

`layer` (page 450) *property*
The view's Core Animation layer used for rendering. (read-only)

`hidden` (page 450) *property*
A Boolean value that determines whether the receiver is hidden.

Animating Views

+ `beginAnimations:context:` (page 454)
Begins an animation block.

+ `commitAnimations` (page 454)
Ends an animation block and starts animations when this is the outer animation block.

+ `setAnimationStartDate:` (page 461)
Sets the start time of animating property changes within an animation block.

+ `setAnimationsEnabled:` (page 460)
Sets whether animations are enabled.

+ `setAnimationDelegate:` (page 457)
Sets the delegate for animation messages.

+ `setAnimationWillStartSelector:` (page 462)
Sets the message to send to the animation delegate when animation starts.

+ `setAnimationDidStopSelector:` (page 458)
Sets the message to send to the animation delegate when animation stops.

+ setAnimationDuration: (page 458)
Sets the duration (in seconds) of animating property changes within an animation block.

+ setAnimationDelay: (page 457)
Sets the delay (in seconds) of animating property changes within an animation block.

+ setAnimationCurve: (page 456)
Sets the curve of animating property changes within an animation block.

+ setAnimationRepeatCount: (page 460)
Sets the number of times animations within an animation block repeat.

+ setAnimationRepeatAutoreverses: (page 459)
Sets whether the animation of property changes within an animation block automatically reverses repeatedly.

+ setAnimationBeginsFromCurrentState: (page 455)
Sets whether the animation should begin playing from the current state.

+ setAnimationTransition:forView:cache: (page 461)
Sets a transition to apply to a view during an animation block.

+ areAnimationsEnabled (page 453)
Returns a Boolean value indicating whether animations are enabled.

Handling Events

- hitTest:withEvent: (page 468)
Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains a specified point.

- pointInside:withEvent: (page 472)
Returns a Boolean value indicating whether the receiver contains the specified point.

multipleTouchEnabled (page 450) *property*
A Boolean value indicating whether the receiver handles multi-touch events.

exclusiveTouch (page 449) *property*
A Boolean value indicating whether the receiver handles touch events exclusively.

Observing Changes

- didAddSubview: (page 466)
Tells the view when subviews are added.

- didMoveToSuperview (page 466)
Informs the receiver that its superview has changed (possibly to `nil`).

- didMoveToWindow (page 466)
Informs the receiver that it has been added to a window.

- willMoveToSuperview: (page 476)
Informs the receiver that its superview is about to change to the specified superview (which may be `nil`).

- willMoveToWindow: (page 476)
Informs the receiver that it's being added to the view hierarchy of the specified window object (which may be `nil`).

- `willRemoveSubview:` (page 477)
  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 2.0 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 454) class method to begin and the `commitAnimations` (page 454) class method to end an animation block.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property backgroundColor` (page 446)
`@property opaque` (page 451)

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 iPhone OS 2.0 and later.

See Also

`@property autoresizingMask` (page 446)

Declared In

`UIView.h`

autoresizingMask

An integer bit mask that determines how the receiver resizes its subviews 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 480).

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 449) and `autoresizingMask` (page 446) 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 471) method to explicitly adjust the position of subviews.

If the autoresizing mask is equal to `UIViewAutoresizingNone` (page 480), then the receiver doesn't resize at all when its bounds changes. The default value is `UIViewAutoresizingNone`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property autoresizesSubviews` (page 445)

Declared In

`UIView.h`

backgroundColor

The receiver's background color.

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

Discussion

Use the `beginAnimations:context:` (page 454) class method to begin and the `commitAnimations` (page 454) class method to end an animation block.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property alpha` (page 445)
`@property opaque` (page 451)

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. Setting this property changes the value of the `frame` (page 449) property accordingly.

Changing the frame rectangle automatically redisplay the receiver without invoking the `drawRect:` (page 467) method. If you want the `drawRect:` (page 467) method invoked when the frame rectangle changes, set the `contentMode` (page 448) property to `UIViewContentModeRedraw` (page 479).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 454) class method to begin and the `commitAnimations` (page 454) 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 iPhone OS 2.0 and later.

See Also
`@property frame` (page 449)
`@property center` (page 447)
`@property transform` (page 452)

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. Setting this property changes the values of the `frame` (page 449) properties accordingly.

Changing the frame rectangle automatically redisplay the receiver without invoking the `drawRect:` (page 467) method. If you want the `drawRect:` (page 467) method invoked when the frame rectangle changes, set the `contentMode` (page 448) property to `UIViewContentModeRedraw` (page 479).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 454) class method to begin and the `commitAnimations` (page 454) class method to end an animation block.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property frame` (page 449)
`@property bounds` (page 446)
`@property transform` (page 452)

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

If `YES`, the current graphics context buffer in the `drawRect:` (page 467) method is automatically cleared to transparent black before `drawRect:` (page 467) is invoked. If `NO`, it's the application's responsibility to completely fill its content. Drawing performance can be improved if this property is `NO`—for example, when scrolling. The default value is `YES`.

Availability

Available in iPhone OS 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 iPhone OS 2.0 and later.

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 478). The default value is `UIViewContentModeScaleToFill` (page 478).

Availability

Available in iPhone OS 2.0 and later.

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 iPhone OS 2.0 and later.

See Also

`@property multipleTouchEnabled` (page 450)

Declared In

`UIView.h`

frame

The receiver's frame rectangle.

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

Discussion

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 447) and `bounds` (page 446) properties accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 467) method. If you want the `drawRect:` (page 467) method invoked when the frame rectangle changes, set the `contentMode` (page 448) property to `UIViewContentModeRedraw` (page 479).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 454) class method to begin and the `commitAnimations` (page 454) class method to end an animation block. If the `transform` (page 452) property is also set, use the `bounds` (page 446) and `center` (page 447) properties instead; otherwise, animating changes to the `frame` property does not correctly reflect the actual location of the view.

Warning: If the `transform` (page 452) property is not the identity transform, the value of this property is undefined and therefore should be ignored.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property bounds` (page 446)
`@property center` (page 447)
`@property transform` (page 452)

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 iPhone OS 2.0 and later.

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 iPhone OS 2.0 and later.

See Also

`+ layerClass` (page 455)

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 449) 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 iPhone OS 2.0 and later.

See Also
@property exclusiveTouch (page 449)

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 iPhone OS 2.0 and later.

See Also
@property backgroundColor (page 446)
@property alpha (page 445)

Declared In
UIView.h

subviews

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

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

Availability
Available in iPhone OS 2.0 and later.

See Also
@property superview (page 452)
- removeFromSuperview (page 472)

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 iPhone OS 2.0 and later.

See Also
`@property subviews` (page 451)
`- removeFromSuperview` (page 472)

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 iPhone OS 2.0 and later.

See Also
`- viewWithTag:` (page 475)

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 447) property, or the layer's `anchorPoint` property if it was changed. (Use the `layer` (page 450) 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 454) class method to begin and the `commitAnimations` (page 454) 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 449) property is undefined and therefore should be ignored.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property frame` (page 449)
`@property bounds` (page 446)
`@property center` (page 447)

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 iPhone OS 2.0 and later.

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 iPhone OS 2.0 and later.

Declared In
`UIView.h`

Class Methods

areAnimationsEnabled

Returns a Boolean value indicating whether animations are enabled.

```
+ (BOOL)areAnimationsEnabled
```

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

Availability
Available in iPhone OS 2.0 and later.

See Also
+ setAnimationsEnabled: (page 460)

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 462) and setAnimationDidStopSelector: (page 458) methods.

context

Additional application-supplied information that is passed to the animation delegate messages—the selectors set using the setAnimationWillStartSelector: (page 462) and setAnimationDidStopSelector: (page 458) 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 454) to begin and the commitAnimations (page 454) class method to end an animation block.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ commitAnimations (page 454)
+ setAnimationWillStartSelector: (page 462)
+ setAnimationDidStopSelector: (page 458)
+ setAnimationDelegate: (page 457)

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 455) for how to start animations while others are in progress.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 454)

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 iPhone OS 2.0 and later.

See Also
`@property layer` (page 450)

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 454) class method to start and the `commitAnimations` (page 454) class method to end an animation block. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 454)
`+ commitAnimations` (page 454)
`+ setAnimationStartDate:` (page 461)
`+ setAnimationDuration:` (page 458)
`+ setAnimationDelay:` (page 457)
`+ setAnimationCurve:` (page 456)
`+ setAnimationRepeatCount:` (page 460)
`+ setAnimationRepeatAutoreverses:` (page 459)

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 454) class method to start and the `commitAnimations` (page 454) class method to end an animation block. The default value of the animation curve is `UIViewAnimationCurveEaseInOut` (page 477).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 454)
`+ commitAnimations` (page 454)
`+ setAnimationStartDate:` (page 461)
`+ setAnimationDuration:` (page 458)
`+ setAnimationDelay:` (page 457)
`+ setAnimationRepeatCount:` (page 460)
`+ setAnimationRepeatAutoreverses:` (page 459)
`+ setAnimationBeginsFromCurrentState:` (page 455)

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 454) class method to start and the `commitAnimations` (page 454) class method to end an animation block. The default value of the animation delay is `0.0`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 454)
`+ commitAnimations` (page 454)
`+ setAnimationStartDate:` (page 461)
`+ setAnimationDuration:` (page 458)
`+ setAnimationCurve:` (page 456)
`+ setAnimationRepeatCount:` (page 460)
`+ setAnimationRepeatAutoreverses:` (page 459)
`+ setAnimationBeginsFromCurrentState:` (page 455)

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 462) and `setAnimationDidStopSelector:` (page 458) methods.

Discussion

This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 454) class method to start and the `commitAnimations` (page 454) class method to end an animation block. The default value is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 454)
`+ commitAnimations` (page 454)
`+ setAnimationWillStartSelector:` (page 462)
`+ setAnimationDidStopSelector:` (page 458)

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 ends. The selector should have the following arguments:

animationID
An optional application-supplied identifier. The same argument passed to the beginAnimations:context: (page 454) method. This argument can be nil.

finished
YES if the animation completed before it stopped; otherwise, NO.

context
An optional application-supplied context. The same argument passed to the beginAnimations:context: (page 454) method. This argument can be nil.

Discussion
This method does nothing if invoked outside of an animation block. Use the beginAnimations:context: (page 454) class method to start and the commitAnimations (page 454) class method to end an animation block. The default value is NULL.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ beginAnimations:context: (page 454)
+ commitAnimations (page 454)
+ setAnimationDelegate: (page 457)
+ setAnimationWillStartSelector: (page 462)

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 454) class method to start and the `commitAnimations` (page 454) class method to end an animation block. The default value is `0.2`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 454)
`+ commitAnimations` (page 454)
`+ setAnimationStartDate:` (page 461)
`+ setAnimationDelay:` (page 457)
`+ setAnimationCurve:` (page 456)
`+ setAnimationRepeatCount:` (page 460)
`+ setAnimationRepeatAutoreverses:` (page 459)
`+ setAnimationBeginsFromCurrentState:` (page 455)

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 460) 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 454) class method to start and the `commitAnimations` (page 454) class method to end an animation block. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 454)
`+ commitAnimations` (page 454)
`+ setAnimationStartDate:` (page 461)
`+ setAnimationDuration:` (page 458)
`+ setAnimationDelay:` (page 457)
`+ setAnimationCurve:` (page 456)
`+ setAnimationRepeatCount:` (page 460)

+ setAnimationBeginsFromCurrentState: (page 455)

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 454) class method to start and the commitAnimations (page 454) class method to end an animation block. By default, animations don't repeat.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ beginAnimations:context: (page 454)
+ commitAnimations (page 454)
+ setAnimationStartDate: (page 461)
+ setAnimationDuration: (page 458)
+ setAnimationDelay: (page 457)
+ setAnimationCurve: (page 456)
+ setAnimationRepeatAutoreverses: (page 459)
+ setAnimationBeginsFromCurrentState: (page 455)

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.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ areAnimationsEnabled (page 453)

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 454) class method to start and the commitAnimations (page 454) class method to end an animation block.

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

Availability
Available in iPhone OS 2.0 and later.

See Also
+ beginAnimations:context: (page 454)
+ commitAnimations (page 454)
+ setAnimationDuration: (page 458)
+ setAnimationDelay: (page 457)
+ setAnimationCurve: (page 456)
+ setAnimationRepeatCount: (page 460)
+ setAnimationRepeatAutoreverses: (page 459)
+ setAnimationBeginsFromCurrentState: (page 455)

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 481).

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; otherwise, the view is rendered to create each frame. Typically, caching improves performance but you can no longer update the view during the transition—you need to wait until the transition ends to update the view.

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.

Availability

Available in iPhone OS 2.0 and later.

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 454) 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 454) class method to start and the commitAnimations (page 454) class method to end an animation block.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ beginAnimations:context: (page 454)
+ commitAnimations (page 454)
+ setAnimationDelegate: (page 457)
+ setAnimationDidStopSelector: (page 458)

Declared In
UIView.h

Instance Methods

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 472) 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.

Availability
Available in iPhone OS 2.0 and later.

See Also
- insertSubview:atIndex: (page 470)
- insertSubview:aboveSubview: (page 469)
- insertSubview:belowSubview: (page 470)
- exchangeSubviewAtIndex:withSubviewAtIndex: (page 468)

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 iPhone OS 2.0 and later.

See Also
- sendSubviewToBack: (page 473)

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 iPhone OS 2.0 and later.

See Also

- `convertPoint:toView:` (page 464)
- `convertRect:toView:` (page 465)
- `convertRect:fromView:` (page 465)

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 iPhone OS 2.0 and later.

See Also

- `convertPoint:fromView:` (page 464)
- `convertRect:toView:` (page 465)
- `convertRect:fromView:` (page 465)

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 iPhone OS 2.0 and later.

See Also
- `convertPoint:toView:` (page 464)
- `convertPoint:fromView:` (page 464)
- `convertRect:toView:` (page 465)

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 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 iPhone OS 2.0 and later.

See Also
- `convertPoint:toView:` (page 464)

- convertPoint:fromView: (page 464)
- convertRect:fromView: (page 465)

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 463).

Availability
Available in iPhone OS 2.0 and later.

See Also
- willRemoveSubview: (page 477)
- addSubview: (page 463)

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 iPhone OS 2.0 and later.

See Also
- willMoveToSuperview: (page 476)

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 453) 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 iPhone OS 2.0 and later.

See Also
`- willMoveToWindow:` (page 476)

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 451) 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` (page 644) 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 iPhone OS 2.0 and later.

See Also
`- setNeedsDisplay` (page 473)
`- setNeedsDisplayInRect:` (page 473)
`@property contentMode` (page 448)

Declared In
`UIView.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 iPhone OS 2.0 and later.

See Also

- `addSubview:` (page 463)
- `insertSubview:atIndex:` (page 470)
- `insertSubview:aboveSubview:` (page 469)
- `insertSubview:belowSubview:` (page 470)

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 472) message to each subview to determine which subview should receive a touch event. If `pointInside:withEvent:` (page 472) 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 views that are hidden, that have disabled user interaction, or have an alpha level less than 0.1.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `pointInside:withEvent:` (page 472)

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 created view object. The origin of the frame is in superview coordinates. Setting this property changes the values of the `center` (page 447) and `bounds` (page 446) 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. This method is the designated initializer for the `UIView` class.

Availability
Available in iPhone OS 2.0 and later.

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.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addSubview:` (page 463)
- `insertSubview:atIndex:` (page 470)
- `insertSubview:belowSubview:` (page 470)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 468)

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 nil.

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

Availability
Available in iPhone OS 2.0 and later.

See Also
- addSubview: (page 463)
- insertSubview:aboveSubview: (page 469)
- insertSubview:belowSubview: (page 470)
- exchangeSubviewAtIndex:withSubviewAtIndex: (page 468)

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.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addSubview: (page 463)
- insertSubview:atIndex: (page 470)
- insertSubview:aboveSubview: (page 469)
- exchangeSubviewAtIndex:withSubviewAtIndex: (page 468)

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 iPhone OS 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.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setNeedsLayout (page 474)
- layoutSubviews (page 471)

Declared In

UIView.h

layoutSubviews

Lays out subviews.

```
- (void)layoutSubviews
```

Discussion

Overridden by subclasses to layout subviews when layoutIfNeeded (page 471) is invoked. The default implementation of this method does nothing.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `setNeedsLayout` (page 474)
- `layoutIfNeeded` (page 471)

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 iPhone OS 2.0 and later.

See Also
- `hitTest:withEvent:` (page 468)

Declared In
`UIView.h`

removeFromSuperview

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

```
- (void)removeFromSuperview
```

Discussion
The receiver is also released; if you plan to reuse it, be sure to retain it before sending this message and to release it as appropriate when adding it as a subview of another `UIView` object.

Never invoke this method while displaying.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `@property superview` (page 452)
- `@property subviews` (page 451)

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 iPhone OS 2.0 and later.

See Also

- bringSubviewToFront: (page 463)

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 467) 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 473) message. Any UIView objects marked as needing display are automatically redisplayed when the application returns to the run loop.

Availability

Available in iPhone OS 2.0 and later.

See Also

- drawRect: (page 467)
- setNeedsDisplayInRect: (page 473)

@property contentMode (page 448)

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 467) 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 473) method to mark a view as needing display.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `drawRect:` (page 467)
- `setNeedsDisplay` (page 473)

`@property contentMode` (page 448)

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 471) lays out the subviews; otherwise, it does not.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `layoutIfNeeded` (page 471)
- `layoutSubviews` (page 471)

Declared In

`UIView.h`

sizeThatFits:

Calculates and returns a size that best fits the receiver's subviews.

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

Parameters

size

The preferred size of the receiver.

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

Discussion
The default implementation returns the *size* argument.

Subclasses override this method to return a view-specific size. For example, `UISwitch` returns a fixed size and `UIImageView` returns the size of the image.

This method does not resize the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sizeToFit` (page 475)
- `@property frame` (page 449)
- `@property bounds` (page 446)

Declared In
`UIView.h`

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

```
- (void)sizeToFit
```

Discussion
This method uses the `sizeThatFits:` (page 474) method to determine the size. Subclasses should override `sizeThatFits:` to compute the appropriate size for the receiver. The default implementation does nothing.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sizeThatFits:` (page 474)

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 iPhone OS 2.0 and later.

See Also
`@property tag` (page 452)

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 iPhone OS 2.0 and later.

See Also
`- didMoveToSuperview` (page 466)

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 iPhone OS 2.0 and later.

See Also
`- didMoveToWindow` (page 466)

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 472) message or *subview* is removed from the receiver because it is being added to another view.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `didAddSubview:` (page 466)
- `addSubview:` (page 463)

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 begins slowly, accelerate through the middle of its duration, and then slow again before completing.

Available in iPhone OS 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 iPhone OS 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 iPhone OS 2.0 and later.

Declared in `UIView.h`

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

Available in iPhone OS 2.0 and later.

Declared in `UIView.h`

Availability
Available in iPhone OS 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 iPhone OS 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 iPhone OS 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 iPhone OS 2.0 and later.

Declared in `UIView.h`

`UIViewContentModeRedraw`
: Redisplays the view when the bounds change by invoking the `setNeedsDisplay` (page 473) method.

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

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

  Available in iPhone OS 2.0 and later.

  Declared in `UIView.h`

Availability
Available in iPhone OS 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 iPhone OS 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 iPhone OS 2.0 and later.
Declared in `UIView.h`

`UIViewAutoresizingFlexibleWidth`
The view resizes by expanding or shrinking its width.
Available in iPhone OS 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 iPhone OS 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 iPhone OS 2.0 and later.
Declared in `UIView.h`

`UIViewAutoresizingFlexibleHeight`
The view resizes by expanding or shrinking its height.
Available in iPhone OS 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 iPhone OS 2.0 and later.
Declared in `UIView.h`

Availability
Available in iPhone OS 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 iPhone OS 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 iPhone OS 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 iPhone OS 2.0 and later.

Declared in `UIView.h`

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

Available in iPhone OS 2.0 and later.

Declared in `UIView.h`

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

Available in iPhone OS 2.0 and later.

Declared in `UIView.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIView.h

UIViewController Class Reference

Inherits from:	UIResponder
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIViewController.h UINavigationController.h UITabBarController.h
Companion guide:	View Controller Programming Guide for iPhone OS

Overview

The `UIViewController` class provides the fundamental view-management model for iPhone. You use instances of `UIViewController`, and its subclasses, to manage tab bars, navigation bars, and the application views you want displayed when items are selected. The `UIViewController` class also supports modal views and rotating views when the orientation changes.

The `UIViewController` class is a superclass for all controllers that manage a full-screen view. For example, a tab bar controller manages a tab bar and the view above it used to implement radio interfaces—clicking on tab bar buttons toggles the view above the tab bar. A navigation controller manages a navigation bar and the view below it used to navigate hierarchically. These specialized view controllers hide the complexity of using the `UITabBar` and `UINavigationBar` classes directly.

You primarily subclass the `UIViewController` class to create your own custom view controllers that manage your full-screen views. You can create a single view controller and add its view to a window just for the benefit of autorotation. Or add the view controller to a tab bar controller or navigation controller and let those objects manage your full-screen views. What methods you use to configure your view controller depends on how you plan to use it.

All subclasses of `UIViewController` should implement the `loadView` (page 494) method to set the `view` (page 491) property unless you load your view from a nib file. The view you create should be a normal view that can be added to any view hierarchy and resized by the controller. When a view

controller is added to a tab bar or navigation controller, its view is resized to fit the available space between the tab bar and navigation bar if they are present. Use the `autoresizesSubviews` (page 445) and `autoresizingMask` (page 446) properties so that your view and its subviews autoresize gracefully. You'll need these properties set to autoresize when the orientation changes too.

A view controller pushed onto the stack of a navigation controller should also set some of the properties in "Configuring Navigation Items" (page 486) to configure a navigation bar when the view controller is the top view controller. Specifically, use the `navigationItem` (page 488) property to configure the navigation bar. Read *UINavigationController Class Reference* for details.

Similarly, a view controller that is added to a tab bar controller should set the `tabBarItem` (page 490) property used to update the tab bar when the view controller is selected. Read *UITabBarController Class Reference* before using the "Configuring Tab Bar Items" (page 486) methods.

View controllers also support autorotation—they slide the tab bar and navigation bar out before the interface rotates, and slide them in after the interface rotates. You can optionally augment the animation and even replace the view for different orientations—for example, use a different view for portrait than for landscape orientation. If your view controller supports orientations other than portrait, override the `shouldAutorotateToInterfaceOrientation:` (page 496) method to return `YES` for the orientations it supports.

View controllers are inserted after their views in the responder chain. That is, a view controller can optionally handle an event that its view does not handle. A view controller's next responder is its view's superview.

Read *View Controller Programming Guide for iPhone OS* to learn how to use this class.

Tasks

Creating a View Controller Using Nib Files

- `initWithNibName:bundle:` (page 493)
  Returns a newly initialized view controller with the nib file in the specified bundle.

`nibName` (page 489) *property*
Return the name of the receiver's nib file if it exists. (read-only)

`nibBundle` (page 489) *property*
Return the name of the receiver's nib bundle if it exists. (read-only)

Configuring Views

`view` (page 491) *property*
The view that this controller manages.

- `loadView` (page 494)
  Creates the view that the controller manages.
- `viewDidLoad` (page 498)
  Invoked when the view is finished loading.

`title` (page 490) *property*
A localized string that represents the view that this controller manages.

Observing Views

- `viewWillAppear:` (page 499)
  Sent to the controller before the view appears and any animations begin.
- `viewDidAppear:` (page 497)
  Sent to the controller after the view fully appears and animations end.
- `viewWillDisappear:` (page 499)
  Sent to the controller before the view is dismissed, covered, or otherwise hidden from view.
- `viewDidDisappear:` (page 498)
  Sent to the controller after the view is dismissed, covered, or otherwise hidden from view.

Handling Rotations

`interfaceOrientation` (page 487) *property*
The current orientation of the interface. (read-only)

- `shouldAutorotateToInterfaceOrientation:` (page 496)
  Returns a Boolean value indicating whether the view controller autorotates its view.
- `rotatingFooterView` (page 495)
  Returns the footer view that slides in and out before and after the user interface rotates.
- `rotatingHeaderView` (page 495)
  Returns the header view that slides in and out before and after the user interface rotates.
- `willRotateToInterfaceOrientation:duration:` (page 501)
  Sent to the view controller when rotating the user interface begins.
- `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 500)
  Sent to the view controller before the first half of the user interface rotates.
- `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 500)
  Sent to the view controller before the second half of the user interface rotates.
- `didRotateFromInterfaceOrientation:` (page 492)
  Sent to the view controller after the user interface rotates.

Handling Memory Warnings

- `didReceiveMemoryWarning` (page 491)
  Sent to the view controller when the application receives a memory warning.

Presenting Modal Views

`modalViewController` (page 488) *property*
The controller for a modal view—a view that is temporarily displayed on top of the view managed by the receiver. (read-only)

`parentViewController` (page 489) *property*
The underlying view controller if this view controller is a modal view controller; otherwise, the enclosing navigation or tab bar controller. `nil` if none of these are present. (read-only)

- `presentModalViewController:animated:` (page 494)
Presents a modal view managed by the given view controller to the user.

- `dismissModalViewControllerAnimated:` (page 492)
Dismisses the receiver's modal view controller.

Configuring Navigation Items

`navigationController` (page 488) *property*
A parent or ancestor that is a navigation controller. (read-only)

`navigationItem` (page 488) *property*
The navigation item used to represent the view controller. (read-only)

`editing` (page 486) *property*
A Boolean value indicating whether the view controller currently allows the user to edit the view contents.

- `setEditing:animated:` (page 496)
Sets whether the view controller shows an editable view.

- `editButtonItem` (page 493)
Returns a bar button item that toggles its title and associated state between Edit and Done.

`hidesBottomBarWhenPushed` (page 487) *property*
A Boolean value indicating whether the bar at the bottom of the screen is hidden when the view controller is pushed on to a navigation controller.

Configuring Tab Bar Items

`tabBarController` (page 490) *property*
A parent or ancestor that is a tab bar controller. (read-only)

`tabBarItem` (page 490) *property*
The tab bar item that represents the view controller when added to a tab bar controller.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

editing

A Boolean value indicating whether the view controller currently allows the user to edit the view contents.

```
@property(nonatomic, getter=isEditing) BOOL editing
```

Discussion
If `YES`, the view controller currently allows editing; otherwise, `NO`.

If the view is editable and the associated navigation controller contains an edit-done button, then a Done button is displayed; otherwise, an Edit button is displayed. Clicking either button toggles the state of this property. Add an edit-done button by setting the custom left or right view of the navigation item to the value returned by the `editButtonItem` (page 493) method. Set the `editing` property to the initial state of your view. Use the `setEditing:animated:` (page 496) method as an action method to animate the transition of this state if the view is already displayed.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setEditing:animated:` (page 496)
- `editButtonItem` (page 493)

Declared In
`UIViewController.h`

hidesBottomBarWhenPushed

A Boolean value indicating whether the bar at the bottom of the screen is hidden when the view controller is pushed on to a navigation controller.

```
@property(nonatomic) BOOL hidesBottomBarWhenPushed
```

Discussion
If `YES`, the bar at the bottom of the screen is hidden; otherwise, `NO`. If `YES`, the bottom bar remains hidden until the view controller is popped from the stack.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationController.h`

interfaceOrientation

The current orientation of the interface. (read-only)

```
@property(nonatomic, readonly) UIInterfaceOrientation interfaceOrientation
```

Discussion
The possible values are described in `Interface Orientation Constants` (page 101).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 501)

- willAnimateFirstHalfOfRotationToInterfaceOrientation:duration: (page 500)
- willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration: (page 500)
- didRotateFromInterfaceOrientation: (page 492)

Declared In
UIViewController.h

modalViewController

The controller for a modal view—a view that is temporarily displayed on top of the view managed by the receiver. (read-only)

```
@property(nonatomic, readonly) UIViewController *modalViewController
```

Discussion
Typically, a modal view is used to present an edit page or additional details of a model object. The modal view is optionally displayed using a vertical sheet transition.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property parentViewController (page 489)

Declared In
UIViewController.h

navigationController

A parent or ancestor that is a navigation controller. (read-only)

```
@property(nonatomic, readonly, retain) UINavigationController *navigationController
```

Discussion
Only returns a navigation controller if the view controller is in its stack. This property is nil if a navigation controller cannot be found.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property tabBarController (page 490)

Declared In
UINavigationController.h

navigationItem

The navigation item used to represent the view controller. (read-only)

```
@property(nonatomic, readonly, retain) UINavigationItem *navigationItem
```

Discussion
This is a unique instance of `UINavigationItem` created to represent the view controller when it is pushed onto a navigation bar. The first time you access this property, the `UINavigationItem` is created. Therefore, you shouldn't access this property if you are not using a navigation controller.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UINavigationController.h`

nibBundle

Return the name of the receiver's nib bundle if it exists. (read-only)

```
@property(readonly, retain) NSBundle *nibBundle
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithNibName:bundle:` (page 493)
`@property nibName` (page 489)

Declared In
`UIViewController.h`

nibName

Return the name of the receiver's nib file if it exists. (read-only)

```
@property(readonly, copy) NSString *nibName
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithNibName:bundle:` (page 493)
`@property nibBundle` (page 489)

Declared In
`UIViewController.h`

parentViewController

The underlying view controller if this view controller is a modal view controller; otherwise, the enclosing navigation or tab bar controller. `nil` if none of these are present. (read-only)

```
@property(nonatomic, readonly) UIViewController *parentViewController
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property modalViewController` (page 488)

Declared In
`UIViewController.h`

tabBarController

A parent or ancestor that is a tab bar controller. (read-only)

```
@property(nonatomic, readonly, retain) UITabBarController *tabBarController
```

Discussion
If the receiver is added to a tab bar controller, this property is the tab bar controller. If the receiver's navigation controller is added to a tab bar controller, this property is the navigation controller's tab bar controller. If no tab bar is present or the receiver is a modal view, this property is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITabBarController.h`

tabBarItem

The tab bar item that represents the view controller when added to a tab bar controller.

```
@property(nonatomic, retain) UITabBarItem *tabBarItem
```

Discussion
The default value is a tab bar item that displays the view controller's title. The first time you access this property, the `UITabBarItem` is created. Therefore, you shouldn't access this property if you are not using a tab bar controller.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITabBarController.h`

title

A localized string that represents the view that this controller manages.

```
@property(nonatomic, copy) NSString *title
```

Discussion
Subclasses should set the title to a human-readable string that represents the view to the user. If the receiver is a navigation controller, the default value is the top view controller's title.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIViewController.h

view

The view that this controller manages.

```
@property(nonatomic, retain) UIView *view
```

Discussion
If this property is nil, the controller sends loadView (page 494) to itself to create the view that it manages. Subclasses should override the loadView method to create any custom views. The default value is nil.

Availability
Available in iPhone OS 2.0 and later.

See Also
- loadView (page 494)
- viewDidLoad (page 498)

Declared In
UIViewController.h

Instance Methods

didReceiveMemoryWarning

Sent to the view controller when the application receives a memory warning.

```
- (void)didReceiveMemoryWarning
```

Discussion
The default implementation of this method determines whether the view controller implements loadView (page 494). If the view controller implements this method and its view is not displayed, its view is released. The next time the view is needed, the loadView method is invoked to recreate the view. Subclasses should override the loadView method to create the view.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `loadView` (page 494)

Declared In
`UIViewController.h`

didRotateFromInterfaceOrientation:

Sent to the view controller after the user interface rotates.

```
- (void)didRotateFromInterfaceOrientation:(UIInterfaceOrientation)fromInterfaceOrientation
```

Parameters

fromInterfaceOrientation

The state of the application's user interface orientation before the rotation. The possible values are described in `Interface Orientation Constants` (page 101).

Discussion

Subclasses optionally override this method to perform additional animations. When this method is invoked the `interfaceOrientation` (page 487) property is set to the new orientation. For example, you might implement this method to resume any activities you turned off in the `willRotateToInterfaceOrientation:duration:` (page 501) method.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 501)
- `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 500)
- `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 500)

Declared In
`UIViewController.h`

dismissModalViewControllerAnimated:

Dismisses the receiver's modal view controller.

```
- (void)dismissModalViewControllerAnimated:(BOOL)animated
```

Parameters

animated

If `YES`, animates the view as it's dismissed; otherwise, does not.

Discussion

If you dismiss a modal view controller in the middle or bottom of the stack, all the modal view controllers on top of that view controller are also dismissed. Use the `modalViewController` (page 488) property before invoking this method if you need to retain the modal view controller.

Availability

Available in iPhone OS 2.0 and later.

See Also
- presentModalViewController:animated: (page 494)

Declared In
UIViewController.h

editButtonItem

Returns a bar button item that toggles its title and associated state between Edit and Done.

```
- (UIBarButtonItem *)editButtonItem
```

Discussion
If one of the custom views of the navigationItem (page 488) property is set to the returned object, the associated navigation bar displays an Edit button if editing (page 486) is NO and a Done button if editing (page 486) is YES. The default button action invokes the setEditing:animated: (page 496) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- @property editing (page 486)
- setEditing:animated: (page 496)

Declared In
UIViewController.h

initWithNibName:bundle:

Returns a newly initialized view controller with the nib file in the specified bundle.

```
- (id)initWithNibName:(NSString *)nibName bundle:(NSBundle *)nibBundle
```

Parameters
nibName

The name of the nib file, without any leading path information. If you specify a nib name and need to set values after the nib file is loaded, then you should override the viewDidLoad (page 498) method to do so. If this argument is nil, the nibName (page 489) property is set to nil. In this case, you should override the loadView (page 494) method to set the view (page 491) property.

nibBundle

The bundle in which to search for the nib file. This method looks for the nib file in the bundle's language-specific project directories first, followed by the Resources directory. If nil, this method looks for the nib file in the main bundle.

Return Value
A newly initialized UIViewController object.

Discussion
This is the designated initializer for this class.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property nibName (page 489)
@property nibBundle (page 489)

Declared In
UIViewController.h

loadView

Creates the view that the controller manages.

```
- (void)loadView
```

Discussion
This method is only invoked when the view (page 491) property is nil and it is needed for display. You should not invoke this method directly.

If you create the view that this view controller manages programmatically, then you should override this method to create your view. The default implementation creates a UIView object with no subviews.

However, if you initialize the view using a nib file—that is, you set thenibName (page 489) and nibBundle (page 489) properties—then you should not override this method because the default implementation already reloads the nib file. Instead override the viewDidLoad (page 498) method to set any properties after the nib file is loaded.

Availability
Available in iPhone OS 2.0 and later.

See Also
@property view (page 491)
- viewDidLoad (page 498)

Declared In
UIViewController.h

presentModalViewController:animated:

Presents a modal view managed by the given view controller to the user.

```
- (void)presentModalViewController:(UIViewController *)modalViewController
    animated:(BOOL)animated
```

Parameters
modalViewController
The view controller that manages the modal view.

animated
If YES, animates the view as it's presented; otherwise, does not.

Discussion
Sets the `modalViewController` (page 488) property to the specified view controller. Resizes its view and attaches it to the view hierarchy. The view is animated from below and placed on top of any tab bars or navigation bars.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- dismissModalViewControllerAnimated:` (page 492)

Declared In
`UIViewController.h`

rotatingFooterView
Returns the footer view that slides in and out before and after the user interface rotates.

```
- (UIView *)rotatingFooterView
```

Return Value
The footer view.

If the view controller is a tab bar controller, returns a view containing the tab bar. If the view controller is a navigation controller, returns the top view controller's footer view. If the keyboard is active, returns the keyboard; otherwise, it returns `nil`.

Discussion
In most cases, the header view is the tab bar and the footer view is the navigation bar. If this default behavior is not desired, subclasses should override this method to return an alternate footer view.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- shouldAutorotateToInterfaceOrientation:` (page 496)
`- rotatingHeaderView` (page 495)

Declared In
`UIViewController.h`

rotatingHeaderView
Returns the header view that slides in and out before and after the user interface rotates.

```
- (UIView *)rotatingHeaderView
```

Return Value
The header view.

If the view controller is a tab bar controller, returns the selected view controller's header view. If the view controller is a navigation controller, returns the view containing the navigation bar. Otherwise, returns `nil`. In most cases, the header view is the tab bar and the footer view is the navigation bar.

Discussion
In most cases, the header view is the tab bar and the footer view is the navigation bar. If the default behavior is not desired, subclasses should override this method to return an alternate header view.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `shouldAutorotateToInterfaceOrientation:` (page 496)
- `rotatingFooterView` (page 495)

Declared In
`UIViewController.h`

setEditing:animated:
Sets whether the view controller shows an editable view.

```
- (void)setEditing:(BOOL)editing animated:(BOOL)animated
```

Parameters
editing

If `YES`, the view controller should display an editable view; otherwise, `NO`.

If `YES` and one of the custom views of the `navigationItem` (page 488) property is set to the value returned by the `editButtonItem` (page 493) method, the associated navigation controller displays a Done button; otherwise, an Edit button.

animated

If `YES`, animates the transition; otherwise, does not.

Discussion
Subclasses that use an edit-done button must override this method to change their view to an editable state if `editing` (page 486) is `YES` and a non-editable state if it is `NO`. This method should invoke super's implementation before updating its view.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `@property editing` (page 486)
- `editButtonItem` (page 493)

Declared In
`UIViewController.h`

shouldAutorotateToInterfaceOrientation:
Returns a Boolean value indicating whether the view controller autorotates its view.

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
```

Parameters

interfaceOrientation

The orientation of the application's user interface after the rotation. The possible values are described in `Interface Orientation Constants` (page 101).

Return Value

`YES` if the view controller autorotates its view to the specified orientation; otherwise, `NO`.

Discussion

By default views display in portrait orientation only. Override this method to change the default behavior.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `rotatingFooterView` (page 495)
- `rotatingHeaderView` (page 495)

Declared In

`UIViewController.h`

viewDidAppear:

Sent to the controller after the view fully appears and animations end.

```
- (void)viewDidAppear:(BOOL)animated
```

Parameters

animated

If `YES`, the appearance of the view is animated; otherwise, it is not.

Discussion

Subclasses may override this method to take an appropriate action. The default implementation of this method does nothing.

Warning: If the view belonging to a view controller is added to a view hierarchy directly, the view controller will not receive this message. If you insert or add a view to the view hierarchy, and it has a view controller, you should send the associated view controller this message directly. Failing to send the view controller this message will prevent any associated animation from being displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `viewWillAppear:` (page 499)
- `viewWillDisappear:` (page 499)
- `viewDidDisappear:` (page 498)

Declared In

`UIViewController.h`

viewDidDisappear:

Sent to the controller after the view is dismissed, covered, or otherwise hidden from view.

```
- (void)viewDidDisappear:(BOOL)animated
```

Parameters

animated

If YES, the disappearance of the view is animated; otherwise, it is not.

Discussion

Subclasses may override this method to take an appropriate action. The default implementation of this method does nothing.

Warning: If the view belonging to a view controller is added to a view hierarchy directly, the view controller will not receive this message. If you insert or add a view to the view hierarchy, and it has a view controller, you should send the associated view controller this message directly. Failing to send the view controller this message will prevent any associated animation from being displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- viewWillAppear: (page 499)
- viewDidAppear: (page 497)
- viewWillDisappear: (page 499)

Declared In

UIViewController.h

viewDidLoad

Invoked when the view is finished loading.

```
- (void)viewDidLoad
```

Discussion

If a view controller is unarchived from a nib file, this method is invoked after its view is set. Therefore, subclasses should override this method, not the loadView (page 494) method, to initialize objects loaded from a nib.

Availability

Available in iPhone OS 2.0 and later.

See Also

@property view (page 491)
- loadView (page 494)

Declared In

UIViewController.h

viewWillAppear:

Sent to the controller before the view appears and any animations begin.

```
- (void)viewWillAppear:(BOOL)animated
```

Parameters

animated

If YES, the appearance of the view is animated; otherwise, it is not.

Discussion

This method is invoked before any animations begin. Subclasses may override this method to take an appropriate action. The default implementation of this method does nothing.

Warning: If the view belonging to a view controller is added to a view hierarchy directly, the view controller will not receive this message. If you insert or add a view to the view hierarchy, and it has a view controller, you should send the associated view controller this message directly. Failing to send the view controller this message will prevent any associated animation from being displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- viewDidAppear: (page 497)
- viewWillDisappear: (page 499)
- viewDidDisappear: (page 498)

Declared In

UIViewController.h

viewWillDisappear:

Sent to the controller before the view is dismissed, covered, or otherwise hidden from view.

```
- (void)viewWillDisappear:(BOOL)animated
```

Parameters

animated

If YES, the disappearance of the view is animated; otherwise, it is not.

Discussion

Subclasses may override this method to take an appropriate action, for example, commit editing or resign the view as the first responder. The default implementation of this method does nothing.

Warning: If the view belonging to a view controller is added to a view hierarchy directly, the view controller will not receive this message. If you insert or add a view to the view hierarchy, and it has a view controller, you should send the associated view controller this message directly. Failing to send the view controller this message will prevent any associated animation from being displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `viewWillAppear:` (page 499)
- `viewDidAppear:` (page 497)
- `viewDidDisappear:` (page 498)

Declared In
`UIViewController.h`

willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:

Sent to the view controller before the first half of the user interface rotates.

```
- (void)willAnimateFirstHalfOfRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
    duration:(NSTimeInterval)duration
```

Parameters

toInterfaceOrientation

The state of the application's user interface orientation after the rotation. The possible values are described in `Interface Orientation Constants` (page 101).

duration

The duration of the first half of the rotation.

Discussion

Subclasses optionally override this method to perform additional animations—that is, animations that appear during the first half of the rotation. When this method is invoked the `interfaceOrientation` (page 487) property is still set to the old orientation. This method is invoked within an animation block that begins sliding the header and footer views out. For example, you might implement this method to adjust the zoom level, scroller position, or other attribute of your view.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 501)
- `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 500)
- `didRotateFromInterfaceOrientation:` (page 492)

Declared In
`UIViewController.h`

willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:

Sent to the view controller before the second half of the user interface rotates.

```
- (void)willAnimateSecondHalfOfRotationFromInterfaceOrientation:(UIInterfaceOrientation)fromInterfaceOrientation
    duration:(NSTimeInterval)duration
```

Parameters

fromInterfaceOrientation

The state of the application's user interface orientation before the rotation. The possible values are described in `Interface Orientation Constants` (page 101).

duration

The duration of the second half of the rotation.

Discussion

Subclasses optionally override this method to perform additional animations. When this method is invoked the `interfaceOrientation` (page 487) property is set to the new orientation. This method is invoked within an animation block that begins sliding the header and footer views in. For example, you might implement this method to adjust the zoom level, scroller position, or other attribute of your view.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- willRotateToInterfaceOrientation:duration:` (page 501)
- `- willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 500)
- `- didRotateFromInterfaceOrientation:` (page 492)

Declared In

`UIViewController.h`

willRotateToInterfaceOrientation:duration:

Sent to the view controller when rotating the user interface begins.

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
    duration:(NSTimeInterval)duration
```

Parameters

toInterfaceOrientation

The state of the application's user interface orientation after the rotation. The possible values are described in `Interface Orientation Constants` (page 101).

duration

The duration of the rotation.

Discussion

Subclasses optionally override this method to perform additional animations—animations that appear at the beginning of the rotation. For example, you might implement this method to disable views, stop media playback, or temporarily turn off expensive drawing or live updates. You may also implement this method to swap the view if it should be different for landscape as compared with portrait orientation.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 500)
- `- willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 500)

- didRotateFromInterfaceOrientation: (page 492)

Declared In
UIViewController.h

UIWebView Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIWebView.h

Overview

You use the `UIWebView` class to embed web content in your application. To do so, you simply create a `UIWebView` object, attach it to a window, and send it a request to load web content. You can also use this class to move back and forward in the history of webpages, and you can even set some web content properties programmatically.

Use the `loadRequest:` (page 509) method to begin loading web content, the `stopLoading` (page 510) method to stop loading, and the `loading` (page 506) property to find out if a web view is in the process of loading.

If you allow the user to move back and forward through the webpage history, then you can use the `goBack` (page 507) and `goForward` (page 508) methods as actions for buttons. Use the `canGoBack` (page 505) and `canGoForward` (page 505) properties to disable the buttons when the user can't move in a direction.

By default, a web view automatically converts telephone numbers that appear in web content to Phone links. When a Phone link is tapped, the Phone application launches and dials the number. Set the `detectsPhoneNumbers` (page 506) property to `NO` to turn off this default behavior.

You can also use the `request` (page 507) property to programmatically set the scale of web content the first time it is displayed in a web view. Thereafter, the user can change the scale using gestures.

Set the `delegate` (page 506) property to an object conforming to the `UIWebViewDelegate` protocol if you want to track loading web content or be notified when the scale changes.

Read *Safari Web Content Guide for iPhone OS* for how to create web content that is compatible with and optimized for displaying in Safari on iPhone and your web views.

Tasks

Setting the Delegate

`delegate` (page 506) *property*
The receiver's delegate.

Loading Content

- `loadData:MIMEType:textEncodingName:baseURL:` (page 508)
Sets the main page contents, MIME type, content encoding, and base URL.
- `loadHTMLString:baseURL:` (page 508)
Sets the main page content and base URL.
- `loadRequest:` (page 509)
Connects to a given URL by initiating an asynchronous client request.

`request` (page 507) *property*
The URL request identifying the location of the content to load. (read-only)

`loading` (page 506) *property*
A Boolean value indicating whether the receiver is done loading content. (read-only)

- `stopLoading` (page 510)
Stops the loading of any web content managed by the receiver.
- `reload` (page 509)
Reloads the current page.

Moving Back and Forward

`canGoBack` (page 505) *property*
A Boolean value indicating whether the receiver can move backward. (read-only)

`canGoForward` (page 505) *property*
A Boolean value indicating whether the receiver can move forward. (read-only)

- `goBack` (page 507)
Loads the previous location in the back-forward list.
- `goForward` (page 508)
Loads the next location in the back-forward list.

Setting Web Content Properties

`detectsPhoneNumbers` (page 506) *property*
A Boolean value indicating whether telephone number detection is on.

`scalesPageToFit` (page 507) *property*
A Boolean value determining whether the webpage scales to fit the view and the user can change the scale.

Running JavaScript

`- stringByEvaluatingJavaScriptFromString:` (page 510)
Returns the result of running a script.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

canGoBack

A Boolean value indicating whether the receiver can move backward. (read-only)

```
@property(nonatomic, readonly, getter=canGoBack) BOOL canGoBack
```

Discussion
If `YES`, able to move backward; otherwise, `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property canGoForward` (page 505)

Declared In
`UIWebView.h`

canGoForward

A Boolean value indicating whether the receiver can move forward. (read-only)

```
@property(nonatomic, readonly, getter=canGoForward) BOOL canGoForward
```

Discussion
If `YES`, able to move forward; otherwise, `NO` .

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property canGoBack` (page 505)

Declared In
`UIWebView.h`

delegate

The receiver's delegate.

```
@property(nonatomic, assign) id<UIWebViewDelegate> delegate
```

Discussion

The delegate is sent messages when content is loading and the scale changes. See *UIWebViewDelegate Protocol Reference* for the optional methods this delegate may implement.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIWebView.h`

detectsPhoneNumbers

A Boolean value indicating whether telephone number detection is on.

```
@property(nonatomic) BOOL detectsPhoneNumbers
```

Discussion

If `YES`, telephone number detection is on; otherwise, `NO`. If a webpage contains numbers that can be interpreted as phone numbers, but are not phone numbers, you can turn off telephone number detection by setting this property to `NO`. The default value is `YES` on devices that have phone capabilities.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIWebView.h`

loading

A Boolean value indicating whether the receiver is done loading content. (read-only)

```
@property(nonatomic, readonly, getter=isLoading) BOOL loading
```

Discussion

If `YES`, the receiver is done loading content; otherwise, `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property request` (page 507)
`- stopLoading` (page 510)
`- loadRequest:` (page 509)
`- reload` (page 509)

Declared In

`UIWebView.h`

request

The URL request identifying the location of the content to load. (read-only)

```
@property(nonatomic, readonly, retain) NSURLRequest *request
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `loadRequest:` (page 509)
- `stopLoading` (page 510)
  `@property loading` (page 506)
- `reload` (page 509)

Declared In
`UIWebView.h`

scalesPageToFit

A Boolean value determining whether the webpage scales to fit the view and the user can change the scale.

```
@property(nonatomic) BOOL scalesPageToFit
```

Discussion
If `YES`, the webpage is scaled to fit and the user can zoom in and zoom out. If `NO`, user zooming is disabled. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIWebView.h`

Instance Methods

goBack

Loads the previous location in the back-forward list.

```
- (void)goBack
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `goBack` (page 507)

Declared In
`UIWebView.h`

goForward

Loads the next location in the back-forward list.

```
- (void)goForward
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- goBack (page 507)

Declared In
UIWebView.h

loadData:MIMEType:textEncodingName:baseURL:

Sets the main page contents, MIME type, content encoding, and base URL.

```
- (void)loadData:(NSData *)data MIMEType:(NSString *)MIMEType
    textEncodingName:(NSString *)encodingName baseURL:(NSURL *)baseURL
```

Parameters
data
The content for the main page.

MIMEType
The MIME type of the content.

encodingName
The IANA encoding name as in `utf-8` or `utf-16`.

baseURL
The base URL for the content.

Availability
Available in iPhone OS 2.0 and later.

See Also
- loadHTMLString:baseURL: (page 508)

Declared In
UIWebView.h

loadHTMLString:baseURL:

Sets the main page content and base URL.

```
- (void)loadHTMLString:(NSString *)string baseURL:(NSURL *)baseURL
```

Parameters
string
The content for the main page.

baseURL
The base URL for the content.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `loadData:MIMEType:textEncodingName:baseURL:` (page 508)

Declared In
`UIWebView.h`

loadRequest:

Connects to a given URL by initiating an asynchronous client request.

```
- (void)loadRequest:(NSURLRequest *)request
```

Parameters

request
A URL request identifying the location of the content to load.

Discussion
To stop this load, use the `stopLoading` (page 510) method. To see whether the receiver is done loading the content, use the `loading` (page 506) property.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property request` (page 507)
- `stopLoading` (page 510)
`@property loading` (page 506)
- `reload` (page 509)

Declared In
`UIWebView.h`

reload

Reloads the current page.

```
- (void)reload
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property request` (page 507)
`@property loading` (page 506)
- `loadRequest:` (page 509)
- `stopLoading` (page 510)

Declared In
`UIWebView.h`

stopLoading

Stops the loading of any web content managed by the receiver.

```
- (void)stopLoading
```

Discussion

Stops any content in the process of being loaded by the main frame or any of its children frames. Does nothing if no content is being loaded.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property request` (page 507)
`@property loading` (page 506)
`- loadRequest:` (page 509)
`- reload` (page 509)

Declared In

`UIWebView.h`

stringByEvaluatingJavaScriptFromString:

Returns the result of running a script.

```
- (NSString *)stringByEvaluatingJavaScriptFromString:(NSString *)script
```

Parameters

script
The script to run.

Return Value

The result of running *script* or `nil` if it fails.

Discussion

JavaScript execution time is limited to 5 seconds for each top-level entry point. If your script executes for more than 5 seconds, Safari stops executing the script. This is likely to occur at a random place in your code, so unintended consequences may result. This limit is imposed because JavaScript execution may cause the main thread to block, so when scripts are running, the user is not able to interact with the webpage.

JavaScript allocations are also limited to 10 MB. Safari raises an exception if you exceed this limit on the total memory allocation for JavaScript.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIWebView.h`

Constants

UIWebViewNavigationType

Constant indicating the user's action.

```
enum {
    UIWebViewNavigationTypeLinkClicked,
    UIWebViewNavigationTypeFormSubmitted,
    UIWebViewNavigationTypeBackForward,
    UIWebViewNavigationTypeReload,
    UIWebViewNavigationTypeFormResubmitted,
    UIWebViewNavigationTypeOther
};
typedef NSUInteger UIWebViewNavigationType;
```

Constants

`UIWebViewNavigationTypeLinkClicked`
User tapped a link.
Available in iPhone OS 2.0 and later.
Declared in `UIWebView.h`

`UIWebViewNavigationTypeFormSubmitted`
User submitted a form.
Available in iPhone OS 2.0 and later.
Declared in `UIWebView.h`

`UIWebViewNavigationTypeBackForward`
User tapped the back or forward button.
Available in iPhone OS 2.0 and later.
Declared in `UIWebView.h`

`UIWebViewNavigationTypeReload`
User tapped the reload button.
Available in iPhone OS 2.0 and later.
Declared in `UIWebView.h`

`UIWebViewNavigationTypeFormResubmitted`
User resubmitted a form.
Available in iPhone OS 2.0 and later.
Declared in `UIWebView.h`

`UIWebViewNavigationTypeOther`
Some other action occurred.
Available in iPhone OS 2.0 and later.
Declared in `UIWebView.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIWebView.h`

UIWindow Class Reference

Inherits from:	UIView : UIResponder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIWindow.h

Overview

The `UIWindow` class defines objects (known as **windows**) that manage and coordinate the windows an application displays on the screen. The two principal functions of a window are to provide an area for displaying its views and to distribute events to the views. The window is the root view in the view hierarchy. A window belongs to a level; the windows in one level appear above another level. For example, alerts appear above normal windows. Typically, there is only one window in an iPhone OS application.

Read Windows and Views in *iPhone OS Programming Guide* to learn how to use this class.

Tasks

Configuring Windows

`windowLevel` (page 515) *property*
The receiver's window level.

Making Windows Key

`keyWindow` (page 514) *property*
A Boolean value that indicates whether the receiver is the key window for the application. (read-only)

- `makeKeyAndVisible` (page 518)
Makes the receiver the key window and makes that window visible.

- `becomeKeyWindow` (page 515)
Invoked automatically to inform the receiver that it has become the key window; never invoke this method directly.

- `makeKeyWindow` (page 518)
Makes the receiver the main window.

- `resignKeyWindow` (page 518)
Invoked automatically when the window resigns key window status; never invoke this method directly.

Converting Coordinates

- `convertPoint:toWindow:` (page 516)
Converts a point from the receiver's coordinate system to that of another window.

- `convertPoint:fromWindow:` (page 516)
Converts a point from the coordinate system of a given window to that of the receiver.

- `convertRect:toWindow:` (page 517)
Converts a rectangle from the receiver's coordinate system to that of another window.

- `convertRect:fromWindow:` (page 517)
Converts a rectangle from the coordinate system of another window to that of the receiver.

Sending Events

- `sendEvent:` (page 519)
Dispatches events sent to the receiver by the `UIApplication` object to its views.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

keyWindow

A Boolean value that indicates whether the receiver is the key window for the application. (read-only)

```
@property(nonatomic, readonly, getter=isKeyWindow) BOOL keyWindow
```

Discussion

If `YES`, the receiver is the key window for the application; otherwise, `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `makeKeyAndVisible` (page 518)
- `becomeKeyWindow` (page 515)
- `makeKeyWindow` (page 518)
- `resignKeyWindow` (page 518)

Declared In
`UIWindow.h`

windowLevel

The receiver's window level.

```
@property(nonatomic) UIWindowLevel windowLevel
```

Discussion
Levels are ordered so that each level groups windows within it in front of those in all preceding groups. For example, alert windows appear in front of all normal-level windows. When a window enters a new level, it's ordered in front of all its peers in that level. See "UIWindowLevel" (page 519) for a list of possible values. The default value is `0.0`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIWindow.h`

Instance Methods

becomeKeyWindow

Invoked automatically to inform the receiver that it has become the key window; never invoke this method directly.

```
- (void)becomeKeyWindow
```

Discussion
This method reestablishes the receiver's first responder, sends the `becomeKeyWindow` (page 515) message to that object if it responds, and posts `UIWindowDidBecomeKeyNotification` (page 521) to the default notification center.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property keyWindow` (page 514)
- `makeKeyAndVisible` (page 518)

- makeKeyWindow (page 518)
- resignKeyWindow (page 518)

Declared In
UIWindow.h

convertPoint:fromWindow:

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

```
- (CGPoint)convertPoint:(CGPoint)point fromWindow:(UIWindow *)window
```

Parameters

point

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

window

The window with *point* in its coordinate system. If *window* is nil, this method instead converts from the screen's base coordinates.

Return Value
The point converted to the coordinate system of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- convertPoint:toWindow: (page 516)

Declared In
UIWindow.h

convertPoint:toWindow:

Converts a point from the receiver's coordinate system to that of another window.

```
- (CGPoint)convertPoint:(CGPoint)point toWindow:(UIWindow *)window
```

Parameters

point

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

window

The window into whose coordinate system *point* is to be converted. If nil, it converts the point to screen coordinates.

Return Value
The point converted to the coordinate system of *window*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- convertPoint:fromWindow: (page 516)

Declared In
UIWindow.h

convertRect:fromWindow:

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

```
- (CGRect)convertRect:(CGRect)rect fromWindow:(UIWindow *)window
```

Parameters

rect

The rectangle in the window's coordinate system.

window

The window with *rect* in its coordinate system. If `nil`, this method instead converts from screen's base coordinates.

Return Value
The converted rectangle.

Availability
Available in iPhone OS 2.0 and later.

See Also
- convertRect:toWindow: (page 517)

Declared In
UIWindow.h

convertRect:toWindow:

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

```
- (CGRect)convertRect:(CGRect)rect toWindow:(UIWindow *)window
```

Parameters

rect

A rectangle in the receiver's coordinate system.

window

The window that is the target of the conversion operation. If `nil`, this method instead converts from screen base coordinates.

Return Value
The converted rectangle.

Availability
Available in iPhone OS 2.0 and later.

See Also
- convertRect:fromWindow: (page 517)

Declared In
UIWindow.h

makeKeyAndVisible

Makes the receiver the key window and makes that window visible.

```
- (void)makeKeyAndVisible
```

Discussion

This is a convenience method to make the receiver the main window and displays it in front of other windows. You can also hide and reveal a window using the inherited `hidden` (page 450) `UIView` property.

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property keyWindow` (page 514)
`- becomeKeyWindow` (page 515)
`- makeKeyWindow` (page 518)
`- resignKeyWindow` (page 518)

Declared In

`UIWindow.h`

makeKeyWindow

Makes the receiver the main window.

```
- (void)makeKeyWindow
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`@property keyWindow` (page 514)
`- makeKeyAndVisible` (page 518)
`- becomeKeyWindow` (page 515)
`- resignKeyWindow` (page 518)

Declared In

`UIWindow.h`

resignKeyWindow

Invoked automatically when the window resigns key window status; never invoke this method directly.

```
- (void)resignKeyWindow
```

Discussion

This method sends `resignKeyWindow` (page 518) to the receiver's first responder and posts `UIWindowDidResignKeyNotification` (page 521) to the default notification center.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property keyWindow` (page 514)
`- makeKeyAndVisible` (page 518)
`- becomeKeyWindow` (page 515)
`- makeKeyWindow` (page 518)

Declared In
`UIWindow.h`

sendEvent:

Dispatches events sent to the receiver by the `UIApplication` object to its views.

```
- (void)sendEvent:(UIEvent *)event
```

Parameters
event
The event to process.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIWindow.h`

Constants

UIWindowLevel

The positioning of windows relative to each other.

```
const UIWindowLevel UIWindowLevelNormal;
const UIWindowLevel UIWindowLevelAlert;
const UIWindowLevel UIWindowLevelStatusBar;
typedef CGFloat UIWindowLevel;
```

Constants
`UIWindowLevelNormal`
The default level.
Available in iPhone OS 2.0 and later.
Declared in `UIWindow.h`

`UIWindowLevelAlert`
The level for an alert view.
Available in iPhone OS 2.0 and later.
Declared in `UIWindow.h`

`UIWindowLevelStatusBar`
The level for a status window.

Available in iPhone OS 2.0 and later.

Declared in `UIWindow.h`

Discussion
The stacking of levels takes precedence over the stacking of windows within each level. That is, even the bottom window in a level obscures the top window of the next level down. Levels are listed in order from lowest to highest.

Keyboard Notification User Info Keys

Keys used to get values from the user information dictionary of keyboard notifications.

```
NSString *const UIKeyboardCenterBeginUserInfoKey;
NSString *const UIKeyboardCenterEndUserInfoKey;
NSString *const UIKeyboardBoundsUserInfoKey;
```

Constants
`UIKeyboardCenterBeginUserInfoKey`
The key for an `NSValue` object containing a `CGPoint` that is the center of the keyboard in screen coordinates before animation.

Available in iPhone OS 2.0 and later.

Declared in `UIWindow.h`

`UIKeyboardCenterEndUserInfoKey`
The key for an `NSValue` object containing a `CGPoint` that is the center of the keyboard in screen coordinates after animation.

Available in iPhone OS 2.0 and later.

Declared in `UIWindow.h`

`UIKeyboardBoundsUserInfoKey`
The key for an `NSValue` object containing a `CGRect` that is the bounds of the keyboard in screen coordinates after animation.

Available in iPhone OS 2.0 and later.

Declared in `UIWindow.h`

Notifications

UIWindowDidBecomeVisibleNotification

Posted when an `UIWindow` object becomes visible.

The notification object is the window object that has become visible. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIWindow.h

UIWindowDidBecomeHiddenNotification

Posted when an UIWindow object becomes hidden.

The notification object is the window object that has become hidden. This notification does not contain a *userInfo* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIWindow.h

UIWindowDidBecomeKeyNotification

Posted whenever a window object becomes the key window.

The notification object is the window object that has become key. This notification does not contain a *userInfo* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIWindow.h

UIWindowDidResignKeyNotification

Posted whenever a window object resigns its status as main window.

The notification object is the window object that has resigned its main window status. This notification does not contain a *userInfo* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIWindow.h

UIKeyboardWillShowNotification

Posted before a window object is displayed.

The notification object is nil. The *userInfo* dictionary contains information about the keyboard. Use the keys described in "Keyboard Notification User Info Keys" (page 520) to get the location and size of the keyboard from the *userInfo* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIWindow.h

UIKeyboardDidShowNotification

Posted after a window object is displayed.

The notification object is `nil`. The *`userInfo`* dictionary contains information about the keyboard. Use the keys described in "Keyboard Notification User Info Keys" (page 520) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIWindow.h`

UIKeyboardWillHideNotification

Posted before a window object is hidden.

The notification object is `nil`. The *`userInfo`* dictionary contains information about the keyboard. Use the keys described in "Keyboard Notification User Info Keys" (page 520) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIWindow.h`

UIKeyboardDidHideNotification

Posted after a window object hide.

The notification object is `nil`. The *`userInfo`* dictionary contains information about the keyboard. Use the keys described in "Keyboard Notification User Info Keys" (page 520) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIWindow.h`

Protocols

UIAccelerometerDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAccelerometer.h

Overview

The `UIAccelerometerDelegate` protocol defines a single method for receiving acceleration-related data from the system. Implementation of this method is optional, but expected.

Tasks

Responding to Acceleration Events

`- accelerometer:didAccelerate:` (page 525) *optional method*
Delivers the latest acceleration data to the delegate. This method is optional.

Instance Methods

accelerometer:didAccelerate:

Delivers the latest acceleration data to the delegate. This method is optional.

```
- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration
    *)acceleration
```

Parameters

accelerometer
The application-wide accelerometer object.

acceleration

The most recent acceleration data.

Discussion

The shared `UIAccelerometer` object invokes this method at the desired interval, providing your delegate with updated acceleration data each time.

This method is always invoked on your application's main thread when it is in the `NSDefaultRunLoopMode` run loop mode.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIAccelerometer.h`

UIActionSheetDelegate Protocol Reference

Framework	/System/Library/Framework/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAlert.h

Overview

The `UIActionSheetDelegate` protocol defines the methods a delegate of a `UIActionSheet` object should implement. The delegate implements the button actions and any other custom behavior. Some of the methods defined in this protocol are optional.

If you add your own buttons or customize the behavior of an action sheet, implement a delegate conforming to this protocol to handle the corresponding delegate messages. Use the `delegate` (page 69) property of the action sheet object to specify one of your application objects as the delegate.

If you add your own buttons to an action sheet, the delegate must implement the `actionSheet:clickedButtonAtIndex:` (page 528) message to respond when those buttons are clicked; otherwise, your custom buttons do nothing. The action sheet is automatically dismissed after the `actionSheet:clickedButtonAtIndex:` (page 528) delegate method is invoked.

Optionally, you can implement the `actionSheetCancel:` (page 530) method to take the appropriate action when the system cancels your action sheet. If the delegate does not implement this method, the default behavior is to simulate the user clicking the cancel button and closing the view.

You can also optionally augment the behavior of presenting and dismissing action sheets using the methods in "Customizing Behavior" (page 528).

Tasks

Responding to Actions

- `actionSheet:clickedButtonAtIndex:` (page 528) *optional method*
  Sent to the delegate when the user clicks a button on an action sheet. This method is optional.

Customizing Behavior

- `willPresentActionSheet:` (page 530) *optional method*
  Sent to the delegate before an action sheet is presented to the user. This method is optional.
- `didPresentActionSheet:` (page 530) *optional method*
  Sent to the delegate after an action sheet is presented to the user. This method is optional.
- `actionSheet:willDismissWithButtonIndex:` (page 529) *optional method*
  Sent to the delegate before an action sheet is dismissed. This method is optional.
- `actionSheet:didDismissWithButtonIndex:` (page 529) *optional method*
  Sent to the delegate after an action sheet is dismissed from the screen. This method is optional.

Canceling

- `actionSheetCancel:` (page 530) *optional method*
  Sent to the delegate before an action sheet is canceled. This method is optional.

Instance Methods

actionSheet:clickedButtonAtIndex:

Sent to the delegate when the user clicks a button on an action sheet. This method is optional.

```
- (void)actionSheet:(UIActionSheet *)actionSheet
    clickedButtonAtIndex:(NSInteger)buttonIndex
```

Parameters

actionSheet
  The action sheet containing the button.

buttonIndex
  The position of the clicked button. The button indices start at `0`.

Discussion

The receiver is automatically dismissed after this method is invoked.

Availability

Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

actionSheet:didDismissWithButtonIndex:

Sent to the delegate after an action sheet is dismissed from the screen. This method is optional.

```
- (void)actionSheet:(UIActionSheet *)actionSheet
    didDismissWithButtonIndex:(NSInteger)buttonIndex
```

Parameters

actionSheet
The action sheet that was dismissed.

buttonIndex
The index of the button that was clicked. The button indices start at 0. If this is the cancel button index, the action sheet is canceling. If -1, the cancel button index is not set.

Discussion
This method is invoked after the animation ends and the view is hidden.

Availability
Available in iPhone OS 2.0 and later.

See Also
- actionSheet:willDismissWithButtonIndex: (page 529)

Declared In
UIAlert.h

actionSheet:willDismissWithButtonIndex:

Sent to the delegate before an action sheet is dismissed. This method is optional.

```
- (void)actionSheet:(UIActionSheet *)actionSheet
    willDismissWithButtonIndex:(NSInteger)buttonIndex
```

Parameters

actionSheet
The action sheet that is about to be dismissed.

buttonIndex
The index of the button that was clicked. If this is the cancel button index, the action sheet is canceling. If -1, the cancel button index is not set.

Discussion
This method is invoked before the animation begins and the view is hidden.

Availability
Available in iPhone OS 2.0 and later.

See Also
- actionSheet:didDismissWithButtonIndex: (page 529)

Declared In
UIAlert.h

actionSheetCancel:

Sent to the delegate before an action sheet is canceled. This method is optional.

```
- (void)actionSheetCancel:(UIActionSheet *)actionSheet
```

Parameters

actionSheet
The action sheet that will be canceled.

Discussion
If the action sheet's delegate does not implement this method, clicking the cancel button is simulated and the action sheet is dismissed. Implement this method if you need to perform some actions before an action sheet is canceled. An action sheet can be canceled at any time by the system—for example, when the user taps the Home button. The `actionSheet:willDismissWithButtonIndex:` (page 529) and `actionSheet:didDismissWithButtonIndex:` (page 529) methods are invoked after this method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

didPresentActionSheet:

Sent to the delegate after an action sheet is presented to the user. This method is optional.

```
- (void)didPresentActionSheet:(UIActionSheet *)actionSheet
```

Parameters

actionSheet
The action sheet that was displayed.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- willPresentActionSheet:` (page 530)

Declared In
UIAlert.h

willPresentActionSheet:

Sent to the delegate before an action sheet is presented to the user. This method is optional.

```
- (void)willPresentActionSheet:(UIActionSheet *)actionSheet
```

Parameters

actionSheet

The action sheet that is about to be displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- didPresentActionSheet: (page 530)

Declared In

UIAlert.h

UIAlertViewDelegate Protocol Reference

Framework	/System/Library/Framework/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIAlert.h

Overview

The `UIAlertViewDelegate` protocol defines the methods a delegate of a `UIAlertView` object should implement. The delegate implements the button actions and any other custom behavior. Some of the methods defined in this protocol are optional.

If you add your own buttons or customize the behavior of an alert view, implement a delegate conforming to this protocol to handle the corresponding delegate messages. Use the `delegate` (page 85) property of an alert view to specify one of your application objects as the delegate.

If you add your own buttons to an alert view, the delegate must implement the `alertView:clickedButtonAtIndex:` (page 534) message to respond when those buttons are clicked; otherwise, your custom buttons do nothing. The alert view is automatically dismissed after the `alertView:clickedButtonAtIndex:` (page 534) delegate method is invoked.

Optionally, you can implement the `alertViewCancel:` (page 536) method to take the appropriate action when the system cancels your alert view. If the delegate does not implement this method, the default behavior is to simulate the user clicking the cancel button and closing the view.

You can also optionally augment the behavior of presenting and dismissing alert views using the methods in "Customizing Behavior" (page 534).

Tasks

Responding to Actions

- `alertView:clickedButtonAtIndex:` (page 534) *optional method*
  Sent to the delegate when the user clicks a button on an alert view. This method is optional.

Customizing Behavior

- `willPresentAlertView:` (page 536) *optional method*
  Sent to the delegate before a model view is presented to the user. This method is optional.
- `didPresentAlertView:` (page 536) *optional method*
  Sent to the delegate after an alert view is presented to the user. This method is optional.
- `alertView:willDismissWithButtonIndex:` (page 535) *optional method*
  Sent to the delegate before an alert view is dismissed. This method is optional.
- `alertView:didDismissWithButtonIndex:` (page 535) *optional method*
  Sent to the delegate after an alert view is dismissed from the screen. This method is optional.

Canceling

- `alertViewCancel:` (page 536) *optional method*
  Sent to the delegate before an alert view is canceled. This method is optional.

Instance Methods

alertView:clickedButtonAtIndex:

Sent to the delegate when the user clicks a button on an alert view. This method is optional.

```
- (void)alertView:(UIAlertView *)alertView
    clickedButtonAtIndex:(NSInteger)buttonIndex
```

Parameters

alertView
  The alert view containing the button.

buttonIndex
  The position of the clicked button. The button indices start at `0`.

Discussion

The receiver is automatically dismissed after this method is invoked.

Availability

Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

alertView:didDismissWithButtonIndex:

Sent to the delegate after an alert view is dismissed from the screen. This method is optional.

```
- (void)alertView:(UIAlertView *)alertView
    didDismissWithButtonIndex:(NSInteger)buttonIndex
```

Parameters

alertView
The alert view that was dismissed.

buttonIndex
The index of the button that was clicked. The button indices start at 0. If this is the cancel button index, the alert view is canceling. If -1, the cancel button index is not set.

Discussion
This method is invoked after the animation ends and the view is hidden.

Availability
Available in iPhone OS 2.0 and later.

See Also
- alertView:willDismissWithButtonIndex: (page 535)

Declared In
UIAlert.h

alertView:willDismissWithButtonIndex:

Sent to the delegate before an alert view is dismissed. This method is optional.

```
- (void)alertView:(UIAlertView *)alertView
    willDismissWithButtonIndex:(NSInteger)buttonIndex
```

Parameters

alertView
The alert view that is about to be dismissed.

buttonIndex
The index of the button that was clicked. The button indices start at 0. If this is the cancel button index, the alert view is canceling. If -1, the cancel button index is not set.

Discussion
This method is invoked before the animation begins and the view is hidden.

Availability
Available in iPhone OS 2.0 and later.

See Also
- alertView:didDismissWithButtonIndex: (page 535)

Declared In
UIAlert.h

alertViewCancel:

Sent to the delegate before an alert view is canceled. This method is optional.

```
- (void)alertViewCancel:(UIAlertView *)alertView
```

Parameters

alertView
The alert view that will be canceled.

Discussion
If the alert view's delegate does not implement this method, clicking the cancel button is simulated and the alert view is dismissed. Implement this method if you need to perform some actions before an alert view is canceled. An alert view can be canceled at any time by the system—for example, when the user taps the Home button. The `alertView:willDismissWithButtonIndex:` (page 535) and `alertView:didDismissWithButtonIndex:` (page 535) methods are invoked after this method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIAlert.h

didPresentAlertView:

Sent to the delegate after an alert view is presented to the user. This method is optional.

```
- (void)didPresentAlertView:(UIAlertView *)alertView
```

Parameters

alertView
The alert view that was displayed.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- willPresentAlertView:` (page 536)

Declared In
UIAlert.h

willPresentAlertView:

Sent to the delegate before a model view is presented to the user. This method is optional.

```
- (void)willPresentAlertView:(UIAlertView *)alertView
```

Parameters

alertView

The alert view that is about to be displayed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- didPresentAlertView: (page 536)

Declared In

UIAlert.h

UIApplicationDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIApplication.h

Overview

The `UIApplicationDelegate` protocol declares methods that are implemented by the delegate of the singleton `UIApplication` object.

By implementing these methods, the delegate can respond to application launch and termination, low-memory warnings, the opening of URL resources, changes in status-bar orientation, and other system events.

Tasks

Opening a URL Resource

- `application:handleOpenURL:` (page 541) *optional method*
  Asks the delegate to open a resource identified by URL. This method is optional.

Managing Status Bar Orientation

- `application:willChangeStatusBarOrientation:duration:` (page 542) *optional method*
  Tells the delegate when the interface orientation of the status bar is about to change. This method is optional.
- `application:didChangeStatusBarOrientation:` (page 541) *optional method*
  Tells the delegate when the interface orientation of the status bar has changed. This method is optional.

Responding to a Change in Active Status

- `applicationWillResignActive:` (page 544) *optional method*
  Tells the delegate that the application will become inactive. This method is optional.
- `applicationDidBecomeActive:` (page 543) *optional method*
  Tells the delegate that the application has become active. This method is optional.

Controlling Application Appearance

- `application:willChangeStatusBarFrame:` (page 542) *optional method*
  Tells the delegate when the frame of the status bar is about to change. This method is optional.
- `application:didChangeStatusBarFrame:` (page 540) *optional method*
  Tells the delegate when the frame of the status bar has changed. This method is optional.

Controlling Application Behavior

- `applicationDidFinishLaunching:` (page 543) *optional method*
  Tells the delegate when the application has finished launching. This method is optional.
- `applicationWillTerminate:` (page 545) *optional method*
  Tells the delegate when the application is about to terminate. This method is optional.
- `applicationDidReceiveMemoryWarning:` (page 543) *optional method*
  Tells the delegate when the application receives a memory warning from the system. This method is optional.
- `applicationSignificantTimeChange:` (page 544) *optional method*
  Tells the delegate when there is a significant change in the time. This method is optional.

Instance Methods

application:didChangeStatusBarFrame:

Tells the delegate when the frame of the status bar has changed. This method is optional.

```
- (void)application:(UIApplication *)application
    didChangeStatusBarFrame:(CGRect)oldStatusBarFrame
```

Parameters

application
  The delegating application object.

oldStatusBarFrame
  The previous frame of the status bar, in screen coordinates.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `application:willChangeStatusBarFrame:` (page 542)

Declared In
UIApplication.h

application:didChangeStatusBarOrientation:

Tells the delegate when the interface orientation of the status bar has changed. This method is optional.

```
- (void)application:(UIApplication *)application
    didChangeStatusBarOrientation:(UIInterfaceOrientation)oldStatusBarOrientation
```

Parameters

application
The delegating application object.

oldStatusBarOrientation
A constant that indicates the previous orientation of the application's user interface; see "Controlling Application Behavior" (page 540) for details.

Discussion
The delegate can get the current device orientation from the shared `UIDevice` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIApplication.h

application:handleOpenURL:

Asks the delegate to open a resource identified by URL. This method is optional.

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
```

Parameters

application
The application object.

url
A object representing a URL (Universal Resource Locator). See the appendix of *iPhone OS Programming Guide* for Apple-registered schemes for URLs.

Return Value
`YES` if the delegate successfully handle the request; `NO` if the attempt to handle the URL failed.

Discussion
There is no equivalent notification for this delegation method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- openURL:` (page 98) (`UIApplication`)

Declared In
UIApplication.h

application:willChangeStatusBarFrame:

Tells the delegate when the frame of the status bar is about to change. This method is optional.

```
- (void)application:(UIApplication *)application
    willChangeStatusBarFrame:(CGRect)newStatusBarFrame
```

Parameters

application

The delegating application object.

newStatusBarFrame

The changed frame of the status bar, in screen coordinates.

Discussion

The application calls this method when it receives a `setStatusBarOrientation:animated:` (page 100) message and is about to change the interface orientation.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- application:didChangeStatusBarFrame:` (page 540)

Declared In

`UIApplication.h`

application:willChangeStatusBarOrientation:duration:

Tells the delegate when the interface orientation of the status bar is about to change. This method is optional.

```
- (void)application:(UIApplication *)application
    willChangeStatusBarOrientation:(UIInterfaceOrientation)newStatusBarOrientation
      duration:(NSTimeInterval)duration
```

Parameters

application

The delegating application object.

newStatusBarOrientation

A constant that indicates the new orientation of the application's user interface; see "Controlling Application Behavior" (page 540) for details.

duration

The duration of the animation to the new orientation, in seconds.

Discussion

The delegate typically implements this method to prepare its windows and views for the new orientation. The delegate can get the current device orientation from the shared `UIDevice` object.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIApplication.h`

applicationDidBecomeActive:

Tells the delegate that the application has become active. This method is optional.

```
- (void)applicationDidBecomeActive:(UIApplication *)application
```

Parameters

application
The singleton application instance.

Discussion

The delegate can implement this method to make adjustments when the application transitions from an inactive state to an active state. When an application is inactive, it is executing but is not dispatching incoming events. This occurs when an overlay window pops up or when the device is locked.

Just after it becomes active, the application also posts a `UIApplicationDidBecomeActiveNotification` (page 104).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIApplication.h`

applicationDidFinishLaunching:

Tells the delegate when the application has finished launching. This method is optional.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
```

Parameters

application
The delegating application object.

Discussion

This method is the ideal place for the delegate to perform various initialization and configuration tasks, especially restoring the application to the previous state and setting up the initial windows and views of the application.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- applicationWillTerminate:` (page 545)

Declared In

`UIApplication.h`

applicationDidReceiveMemoryWarning:

Tells the delegate when the application receives a memory warning from the system. This method is optional.

```
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
```

Parameters

application

The delegating application object.

Discussion

Your implementation of this method should free up as much memory as possible by purging cached data objects that can be recreated (or reloaded from disk) later. You use this method in conjunction with the `didReceiveMemoryWarning` of the `UIViewController` class and the `UIApplicationDidReceiveMemoryWarningNotification` notification to release memory throughout your application.

It is strongly recommended that you implement this method. If your application does not release enough memory during low-memory conditions, the system may terminate it outright.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- didReceiveMemoryWarning` (page 491) (UIViewController)

`UIApplicationDidReceiveMemoryWarningNotification` (page 105)

Declared In

`UIApplication.h`

applicationSignificantTimeChange:

Tells the delegate when there is a significant change in the time. This method is optional.

```
- (void)applicationSignificantTimeChange:(UIApplication *)application
```

Parameters

application

The delegating application object.

Discussion

Examples of significant time changes include the arrival of midnight, an update of the time by a carrier, and the change to daylight savings time. The delegate can implement this method to adjust any object of the application that displays time or is sensitive to time changes.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIApplication.h`

applicationWillResignActive:

Tells the delegate that the application will become inactive. This method is optional.

```
- (void)applicationWillResignActive:(UIApplication *)application
```

Parameters

application

The singleton application instance.

Discussion

The delegate can implement this method to make adjustments when the application transitions from an active state to an inactive state. When an application is inactive, it is executing but is not dispatching incoming events. This occurs when an overlay window pops up or when the device is locked.

Just before it becomes inactive, the application also posts a `UIApplicationWillResignActiveNotification` (page 106).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIApplication.h`

applicationWillTerminate:

Tells the delegate when the application is about to terminate. This method is optional.

```
- (void)applicationWillTerminate:(UIApplication *)application
```

Parameters

application

The delegating application object.

Discussion

This method is the ideal place for the delegate to perform clean-up tasks, such as freeing allocated memory, invalidating timers, and storing application state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- applicationDidFinishLaunching:` (page 543)

Declared In

`UIApplication.h`

UIImagePickerControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIImagePickerController.h

Overview

The `UIImagePickerControllerDelegate` protocol defines methods that your delegate must implement to interact with the image picker interface. The methods of this protocol notify your delegate object when the user either picks an image or cancels the picker operation.

Your delegate methods are responsible for dismissing the picker when the operation completes. You do this using the `dismissModalViewControllerAnimated:` method of the parent controller responsible for displaying the `UIImagePickerController` object.

Tasks

Closing the Picker

`- imagePickerController:didFinishPickingImage:editingInfo:` (page 548) *optional method*
Tells the delegate that the user picked an image. This method is optional.

`- imagePickerControllerDidCancel:` (page 548) *optional method*
Tells the delegate that the user cancelled the pick operation. This method is optional.

Instance Methods

imagePickerController:didFinishPickingImage:editingInfo:

Tells the delegate that the user picked an image. This method is optional.

```
- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo
```

Parameters

picker

The controller object managing the image picker interface.

image

The image the user picked. If user editing is enabled, this image represents the final image and may actually be a cropped and adjusted version of the original image. In that case, the original image and editing information are available in the *editingInfo* parameter.

editingInfo

A dictionary containing any relevant editing information. If editing is disabled, this parameter is `nil`. The keys for this dictionary are listed in "Editing information keys" (page 549).

Discussion

Your delegate's implementation of this method should pass the specified image on to any custom code that needs it and then dismiss the picker view.

When user editing is enabled, the picker view presents the user with a preview of the currently selected image along with controls for modifying it. (This behavior is managed by the picker view prior to calling this method.) If the user modifies the image, the editing information is returned in the `editingInfo` parameter in case your code needs it. If not, you can simply use the image in the *image* parameter as is.

Implementation of this method is optional, but expected.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIImagePickerController.h`

imagePickerControllerDidCancel:

Tells the delegate that the user cancelled the pick operation. This method is optional.

```
- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
```

Parameters

picker

The controller object managing the image picker interface.

Discussion

Your delegate's implementation of this method should dismiss the picker view.

Implementation of this method is optional, but expected.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImagePickerController.h`

Constants

Editing information keys

These strings identify the possible keys in the editing information dictionary passed to the delegate.

```
UIKIT_EXTERN NSString *const UIImagePickerControllerOriginalImage;
UIKIT_EXTERN NSString *const UIImagePickerControllerCropRect;
```

Constants

`UIImagePickerControllerOriginalImage`
The key specifying the original uncropped image selected by the user. The value for this key is a `UIImage` object.

Available in iPhone OS 2.0 and later.

Declared in `UIImagePickerController.h`

`UIImagePickerControllerCropRect`
The key specifying the cropping rectangle that was applied to the original image. The value for this key is an `NSValue` object containing a `CGRect` data type.

Available in iPhone OS 2.0 and later.

Declared in `UIImagePickerController.h`

UINavigationBarDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UINavigationBar.h

Overview

The `UINavigationBarDelegate` protocol defines optional methods that a `UINavigationBar` delegate should implement to update its views when items are pushed and popped from the stack. The navigation bar represents only the bar at the top of the screen, not the view below. It's the application's responsibility to implement the behavior when the top item changes.

You can control whether an item should be pushed or popped by implementing the `navigationBar:shouldPushItem:` (page 553) and `navigationBar:shouldPopItem:` (page 553) methods. These methods should return `YES` if the action is allowed; otherwise, `NO`.

The screen should always reflect the top item on the navigation bar. You implement the `navigationBar:didPushItem:` (page 552) method to update the view below the navigation bar to reflect the new item. Similarly, you implement the `navigationBar:didPopItem:` (page 552) method to replace the view below the navigation bar.

Tasks

Pushing Items

- `navigationBar:shouldPushItem:` (page 553) *optional method*
  Returns a Boolean value indicating whether the navigation bar should push an item. This method is optional.
- `navigationBar:didPushItem:` (page 552) *optional method*
  Tells the delegate that an item was pushed onto the navigation bar. This method is optional.

Popping Items

- navigationBar:shouldPopItem: (page 553) *optional method*
  Returns a Boolean value indicating whether the navigation bar should pop an item. This method is optional.
- navigationBar:didPopItem: (page 552) *optional method*
  Tells the delegate that an item was popped from the navigation bar. This method is optional.

Instance Methods

navigationBar:didPopItem:

Tells the delegate that an item was popped from the navigation bar. This method is optional.

```
- (void)navigationBar:(UINavigationBar *)navigationBar didPopItem:(UINavigationItem
    *)item
```

Parameters

navigationBar
The navigation bar that the item is being popped from.

item
The navigation item that is being popped.

Discussion

If animating the pop operation, this method is invoked after the animation ends; otherwise, it is invoked immediately after the pop.

Availability

Available in iPhone OS 2.0 and later.

See Also

- navigationBar:shouldPopItem: (page 553)

Declared In

UINavigationBar.h

navigationBar:didPushItem:

Tells the delegate that an item was pushed onto the navigation bar. This method is optional.

```
- (void)navigationBar:(UINavigationBar *)navigationBar didPushItem:(UINavigationItem
    *)item
```

Parameters

navigationBar
The navigation bar that the item is being pushed onto.

item
The navigation item that is being pushed.

Discussion
If pushing an item onto the navigation bar is animated, this method is invoked after the animation ends; otherwise, it is invoked immediately after the push.

Availability
Available in iPhone OS 2.0 and later.

See Also
- navigationBar:shouldPushItem: (page 553)

Declared In
UINavigationBar.h

navigationBar:shouldPopItem:
Returns a Boolean value indicating whether the navigation bar should pop an item. This method is optional.

```
- (BOOL)navigationBar:(UINavigationBar *)navigationBar
    shouldPopItem:(UINavigationItem *)item
```

Parameters
navigationBar
The navigation bar that the item is being popped from.

item
The navigation item that is being popped.

Return Value
YES if the item should be popped; otherwise, NO.

Discussion
Sent to the delegate before popping an item from the navigation bar.

Availability
Available in iPhone OS 2.0 and later.

See Also
- navigationBar:didPopItem: (page 552)

Declared In
UINavigationBar.h

navigationBar:shouldPushItem:
Returns a Boolean value indicating whether the navigation bar should push an item. This method is optional.

```
- (BOOL)navigationBar:(UINavigationBar *)navigationBar
    shouldPushItem:(UINavigationItem *)item
```

Parameters
navigationBar
The navigation bar that the item is being pushed onto.

item
The navigation item that is being pushed.

Return Value
YES if the item should be pushed; otherwise, NO.

Discussion
Sent to the delegate before pushing an item onto the navigation bar.

Availability
Available in iPhone OS 2.0 and later.

See Also
- navigationBar:didPushItem: (page 552)

Declared In
UINavigationBar.h

UIPickerViewDataSource Protocol Reference

Conforms to: NSObject (NSObject)

Framework /System/Library/Frameworks/UIKit.framework

Availability: Available in iPhone OS 2.0 and later.

Declared in: UIPickerView.h

Overview

The `UIPickerViewDataSource` protocol is adopted by a object that mediates the application's data model for a `UIPickerView` object. The data source provides the `UIPickerView` object with the number of rows and components it needs to display the data in the picker view.

Tasks

Providing Content for the Picker

`- numberOfComponentsInPickerView:` (page 555)
Asks the data source to return the number of components in the given picker view.

`- pickerView:numberOfRowsInComponent:` (page 556)
Asks the data source to return the number of rows in the given component.

Instance Methods

numberOfComponentsInPickerView:

Asks the data source to return the number of components in the given picker view.

```
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
```

Parameters

pickerView

An object representing the picker view requesting the data.

Return Value

The number of components (or "columns") that the picker view should display.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIPickerView.h

pickerView:numberOfRowsInComponent:

Asks the data source to return the number of rows in the given component.

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
    numberOfRowsInComponent:(NSInteger)component
```

Parameters

pickerView

An object representing the picker view requesting the data.

component

An index number identifying a component of *pickerView*.

Return Value

The number of rows displayed by *component*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIPickerView.h

UIPickerViewDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIPickerView.h

Overview

The delegate of a `UIPickerView` object must adopt this protocol and implement at least some of its methods to provide the picker view with the data it needs to construct itself.

The delegate implements the required methods of the protocol to return height, width, row title, and the view content for the rows in each component. It must also provide the content for each component's row, either as a string or a view. Typically the delegate implements other optional methods to respond to new selections or deselections of component rows.

See the `UIPickerView` class reference for a discussion of components, rows, row content, and row selection.

Tasks

Setting the Dimensions of the Picker View

`- pickerView:rowHeightForComponent:` (page 559) *optional method*
Asks the delegate to return the row height to use for drawing row content. This method is optional.

`- pickerView:widthForComponent:` (page 560) *optional method*
Asks the delegate to return the row width to use for drawing row content. This method is optional.

Setting the Content of Component Rows

Although they are marked `@optional`, you must implement either `pickerView:titleForRow:forComponent:` (page 559) or `pickerView:viewForRow:forComponent:reusingView:` (page 559) to provide the content of component rows.

- `pickerView:titleForRow:forComponent:` (page 559) *optional method*
  Asks the delegate to return the title to use for a given row in a given component. This method is optional.
- `pickerView:viewForRow:forComponent:reusingView:` (page 559) *optional method*
  Asks the delegate to return the view to use for a given row in a given component. This method is optional.

Responding to Row Selection

- `pickerView:didSelectRow:inComponent:` (page 558) *optional method*
  Asks the delegate to respond to the user selecting a specific row in a component of a picker view. This method is optional.

Instance Methods

pickerView:didSelectRow:inComponent:

Asks the delegate to respond to the user selecting a specific row in a component of a picker view. This method is optional.

```
- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
    inComponent:(NSInteger)component
```

Parameters

pickerView
  An object representing the picker view requesting the data.

row
  An index number identifying a row of *component*.

component
  An index number identifying a component of *pickerView*.

Discussion

To determine what value the user selected, the delegate uses the *row* index to access the value at the corresponding position in the array used to construct the component.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIPickerView.h`

pickerView:rowHeightForComponent:

Asks the delegate to return the row height to use for drawing row content. This method is optional.

```
- (CGFloat)pickerView:(UIPickerView
    *)pickerViewrowHeightForComponent:(NSInteger)component
```

Parameters

pickerView

The picker view requesting this information.

component

An index number identifying a component of *pickerView*.

Return Value

A float value indicating the height of the row in points.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIPickerView.h

pickerView:titleForRow:forComponent:

Asks the delegate to return the title to use for a given row in a given component. This method is optional.

```
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
    forComponent:(NSInteger)component
```

Parameters

pickerView

An object representing the picker view requesting the data.

row

An index number identifying a row of *component*.

component

An index number identifying a component of *pickerView*.

Return Value

The string to use as the title of the indicated component row.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIPickerView.h

pickerView:viewForRow:forComponent:reusingView:

Asks the delegate to return the view to use for a given row in a given component. This method is optional.

```
- (UIView *)pickerView:(UIPickerView *)pickerView viewForRow:(NSInteger)row
    forComponent:(NSInteger)component reusingView:(UIView *)view
```

Parameters

pickerView

An object representing the picker view requesting the data.

row

An index number identifying a row of *component*.

component

An index number identifying a component of *pickerView*.

view

A view object that was previously used for this row, but is now hidden and cached by the picker view.

Return Value

A view object to use as the content of *row*. The object can be any subclass of `UIView`, such as `UILabel`, `UIImageView`, or even a custom view.

Discussion

If the previously used view (the *view* parameter) is adequate, return that. If you return a different view, the previously used view is released. The picker view centers the returned view in the rectangle for *row*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIPickerView.h`

pickerView:widthForComponent:

Asks the delegate to return the row width to use for drawing row content. This method is optional.

```
- (CGFloat)pickerView:(UIPickerView *)pickerView
    widthForComponent:(NSInteger)component
```

Parameters

pickerView

The picker view requesting this information.

component

An index number identifying a component of the picker view.

Return Value

A float value indicating the width of the row in points.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIPickerView.h`

UIScrollViewDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIScrollView.h

Overview

The methods declared by the `UIScrollViewDelegate` protocol allow the adopting delegate to respond to messages from the `UIScrollView` class and thus respond to, and in some affect, operations such as scrolling, zooming, deceleration of scrolled content, and scrolling animations.

Tasks

Responding to Scrolling and Dragging

- `scrollViewDidScroll:` (page 564) *optional method*
  Tells the delegate when the user scrolls the content view within the receiver. This method is optional.
- `scrollViewWillBeginDragging:` (page 565) *optional method*
  Tells the delegate when the scroll view is about to start scrolling the content. This method is optional.
- `scrollViewDidEndDragging:willDecelerate:` (page 563) *optional method*
  Tells the delegate when dragging ended in the scroll view. This method is optional.
- `scrollViewWillScrollToTop:` (page 566) *optional method*
  Asks the delegate if the scroll view should scroll to the top of the content. This method is optional.
- `scrollViewDidScrollToTop:` (page 564) *optional method*
  Tells the delegate that the scroll view scrolled to the top of the content. This method is optional.

- `scrollViewWillBeginDecelerating:` (page 565) *optional method*
  Tells the delegate that the scroll view is starting to decelerate the scrolling movement. This method is optional.
- `scrollViewDidEndDecelerating:` (page 562) *optional method*
  Tells the delegate that the scroll view has ended decelerating the scrolling movement. This method is optional.

Managing Zooming

- `viewForZoomingInScrollView:` (page 566) *optional method*
  Asks the delegate for the view to scale when zooming is about to occur in the scroll view. This method is optional.
- `scrollViewDidEndZooming:withView:atScale:` (page 563) *optional method*
  Tells the delegate when zooming of the content in the scroll view completed. This method is optional.

Responding to Scrolling Animations

- `scrollViewDidEndScrollingAnimation:` (page 563) *optional method*
  Tells the delegate when a scrolling animation in the scroll view concludes. This method is optional.

Instance Methods

scrollViewDidEndDecelerating:

Tells the delegate that the scroll view has ended decelerating the scrolling movement. This method is optional.

```
- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView
```

Parameters

scrollView
  The scroll-view object that is decelerating the scrolling of the content view.

Discussion

The scroll view calls this method when the scrolling movement comes to a halt. The `decelerating` (page 295) property of `UIScrollView` controls deceleration.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `scrollViewWillBeginDecelerating:` (page 565)

Declared In

`UIScrollView.h`

scrollViewDidEndDragging:willDecelerate:

Tells the delegate when dragging ended in the scroll view. This method is optional.

```
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
    willDecelerate:(BOOL)decelerate
```

Parameters

scrollView

The scroll-view object that finished scrolling the content view.

decelerate

YES if the scrolling movement will continue, but decelerate, after a touch-up gesture during a dragging operation. If the value is NO, scrolling stops immediately upon touch-up.

Discussion

The scroll view sends this message when the user's finger touches up after dragging content. The decelerating (page 295) property of UIScrollView controls deceleration.

Availability

Available in iPhone OS 2.0 and later.

See Also

- scrollViewDidScroll: (page 564)
- scrollViewWillBeginDragging: (page 565)
- scrollViewWillBeginDecelerating: (page 565)

Declared In

UIScrollView.h

scrollViewDidEndScrollingAnimation:

Tells the delegate when a scrolling animation in the scroll view concludes. This method is optional.

```
- (void)scrollViewDidEndScrollingAnimation:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object that is performing the scrolling animation.

Discussion

The scroll view calls this method at the end of its implementations of the UIScrollView and setContentOffset:animated: (page 302) and scrollRectToVisible:animated: (page 301) methods, but only if animations are requested.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UIScrollView.h

scrollViewDidEndZooming:withView:atScale:

Tells the delegate when zooming of the content in the scroll view completed. This method is optional.

```
- (void)scrollViewDidEndZooming:(UIScrollView *)scrollView withView:(UIView *)view
    atScale:(float)scale
```

Parameters

scrollView

The scroll-view object displaying the content view.

view

The view object representing that part of the content view that needs to be scaled.

scale

The scale factor to use for scaling; this value must be between the limits established by the `UIScrollView` properties `maximumZoomScale` (page 297) and `minimumZoomScale` (page 297).

Discussion

The scroll view also calls this method after any "bounce" animations.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- scrollViewWillBeginZooming:` (page 566)

Declared In

`UIScrollView.h`

scrollViewDidScroll:

Tells the delegate when the user scrolls the content view within the receiver. This method is optional.

```
- (void)scrollViewDidScroll:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object in which the scrolling occurred.

Discussion

The delegate typically implements this method to obtain the change in content offset from *scrollView* and draw the affected portion of the content view.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- scrollViewWillBeginDragging:` (page 565)

`- scrollViewDidEndDragging:willDecelerate:` (page 563)

Declared In

`UIScrollView.h`

scrollViewDidScrollToTop:

Tells the delegate that the scroll view scrolled to the top of the content. This method is optional.

```
- (void)scrollViewDidScrollToTop:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object that perform the scrolling operation.

Discussion

The scroll view sends this message when it finishes scrolling to the top of the content. It might call it immediately if the top of the content is already shown. For the scroll-to-top gesture (a tap on the status bar) to be effective, the `scrollsToTop` (page 299) property of the `UIScrollView` must be set to YES.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `scrollViewWillScrollToTop:` (page 566)
- `scrollViewDidScroll:` (page 564)

Declared In

`UIScrollView.h`

scrollViewWillBeginDecelerating:

Tells the delegate that the scroll view is starting to decelerate the scrolling movement. This method is optional.

```
- (void)scrollViewWillBeginDecelerating:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object that is decelerating the scrolling of the content view.

Discussion

The scroll view calls this method as the user's finger touches up as it is moving during a scrolling operation; the scroll view will continue to move a short distance afterwards. The `decelerating` (page 295) property of `UIScrollView` controls deceleration.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `scrollViewDidEndDecelerating:` (page 562)

Declared In

`UIScrollView.h`

scrollViewWillBeginDragging:

Tells the delegate when the scroll view is about to start scrolling the content. This method is optional.

```
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object that is about to scroll the content view.

Discussion

The delegate might not receive this message until dragging has occurred over a small distance.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `scrollViewDidScroll:` (page 564)
- `scrollViewDidEndDragging:willDecelerate:` (page 563)

Declared In

`UIScrollView.h`

scrollViewWillScrollToTop:

Asks the delegate if the scroll view should scroll to the top of the content. This method is optional.

```
- (BOOL)scrollViewWillScrollToTop:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object requesting this information.

Return Value

`YES` to permit scrolling to the top of the content, `NO` to disallow it.

Discussion

If the delegate doesn't implement this method, `YES` is assumed. For the scroll-to-top gesture (a tap on the status bar) to be effective, the `scrollsToTop` (page 299) property of the `UIScrollView` must be set to YES.

See Also

- `scrollViewDidScrollToTop:` (page 564)
- `scrollViewDidScroll:` (page 564)

viewForZoomingInScrollView:

Asks the delegate for the view to scale when zooming is about to occur in the scroll view. This method is optional.

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object displaying the content view.

Return Value

A `UIView` object that will be scaled as a result of the zooming gesture. Return `nil` if you don't want zooming to occur.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- scrollViewDidEndZooming:withView:atScale:` (page 563)

Declared In
`UIScrollView.h`

UISearchBarDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UISearchBar.h

Overview

The `UISearchBarDelegate` protocol defines the optional methods you implement to make a `UISearchBar` control functional. A `UISearchBar` object provides the user interface for a search field on a bar, but it's the application's responsibility to implement the actions when buttons are tapped. At a minimum, the delegate needs to perform the actual search when text is entered in the text field.

Tasks

Editing Text

- `searchBar:textDidChange:` (page 570)
  Sent to the delegate after the user changed the search text.
- `searchBarShouldBeginEditing:` (page 571)
  Asks the delegate if editing should begin in the specified search bar.
- `searchBarTextDidBeginEditing:` (page 572)
  Sent to the delegate when the user begins editing the search text.
- `searchBarShouldEndEditing:` (page 572)
  Asks the delegate if editing should stop in the specified search bar.
- `searchBarTextDidEndEditing:` (page 573)
  Sent to the delegate when the user finishes editing the search text.

Clicking Buttons

- `searchBarBookmarkButtonClicked:` (page 570)
  Sent to the delegate when the bookmark button is tapped.
- `searchBarCancelButtonClicked:` (page 571)
  Sent to the delegate when the cancel button is tapped.
- `searchBarSearchButtonClicked:` (page 571)
  Sent to the delegate when the search button is tapped.

Instance Methods

searchBar:textDidChange:

Sent to the delegate after the user changed the search text.

```
- (void)searchBar:(UISearchBar *)searchBar textDidChange:(NSString *)searchText
```

Parameters

searchBar
The search bar that is being edited.

searchText
The current text in the search text field.

Discussion
This method is also invoked when text is cleared from the search text field.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UISearchBar.h

searchBarBookmarkButtonClicked:

Sent to the delegate when the bookmark button is tapped.

```
- (void)searchBarBookmarkButtonClicked:(UISearchBar *)searchBar
```

Parameters

searchBar
The search bar that was tapped.

Discussion
There is no automatic bookmark support provided by the search bar. It's the application's responsibility to implement this method to perform some action if the bookmark button is tapped by the user.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property showsBookmarkButton` (page 308)

Declared In
`UISearchBar.h`

searchBarCancelButtonClicked:

Sent to the delegate when the cancel button is tapped.

```
- (void)searchBarCancelButtonClicked:(UISearchBar *)searchBar
```

Parameters

searchBar
The search bar that was tapped.

Discussion
Typically, you implement this method to dismiss the search bar.

Availability
Available in iPhone OS 2.0 and later.

See Also
`@property showsCancelButton` (page 309)

Declared In
`UISearchBar.h`

searchBarSearchButtonClicked:

Sent to the delegate when the search button is tapped.

```
- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar
```

Parameters

searchBar
The search bar that was tapped.

Discussion
You should implement this method to begin the search. Use the `text` (page 309) property of the search bar to get the text. You can also send `becomeFirstResponder` (page 280) to the search bar to begin editing programmatically.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UISearchBar.h`

searchBarShouldBeginEditing:

Asks the delegate if editing should begin in the specified search bar.

```
- (BOOL)searchBarShouldBeginEditing:(UISearchBar *)searchBar
```

Parameters

searchBar

The search bar that is being edited.

Return Value

YES if an editing session should be initiated; otherwise, NO to disallow editing.

Availability

Available in iPhone OS 2.0 and later.

See Also

- searchBarTextDidBeginEditing: (page 572)
- searchBarShouldEndEditing: (page 572)
- searchBarTextDidEndEditing: (page 573)

Declared In

UISearchBar.h

searchBarShouldEndEditing:

Asks the delegate if editing should stop in the specified search bar.

```
- (BOOL)searchBarShouldEndEditing:(UISearchBar *)searchBar
```

Parameters

searchBar

The search bar that is being edited.

Return Value

YES if editing should stop; otherwise, NO if the editing session should continue

Availability

Available in iPhone OS 2.0 and later.

See Also

- searchBarShouldBeginEditing: (page 571)
- searchBarTextDidBeginEditing: (page 572)
- searchBarTextDidEndEditing: (page 573)

Declared In

UISearchBar.h

searchBarTextDidBeginEditing:

Sent to the delegate when the user begins editing the search text.

```
- (void)searchBarTextDidBeginEditing:(UISearchBar *)searchBar
```

Parameters

searchBar

The search bar that is being edited.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `searchBarShouldBeginEditing:` (page 571)
- `searchBarShouldEndEditing:` (page 572)
- `searchBarTextDidEndEditing:` (page 573)

Declared In
`UISearchBar.h`

searchBarTextDidEndEditing:

Sent to the delegate when the user finishes editing the search text.

```
- (void)searchBarTextDidEndEditing:(UISearchBar *)searchBar
```

Parameters

searchBar
The search bar that is being edited.

Discussion
Typically, you implement this method to perform the text-based search.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `searchBarShouldBeginEditing:` (page 571)
- `searchBarTextDidBeginEditing:` (page 572)
- `searchBarShouldEndEditing:` (page 572)

Declared In
`UISearchBar.h`

UITabBarControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITabBarController.h
Companion guide:	View Controller Programming Guide for iPhone OS

Overview

You use the `UITabBarControllerDelegate` protocol when you want to augment the behavior of customizing a tab bar—the user interface for adding, replacing, and removing items from a tab bar. For example, if you want to perform some action after the user rearranges items on the tab bar. Use this protocol by setting the delegate of a `UITabBarController` object to an object conforming to this protocol.

Read *View Controller Programming Guide for iPhone OS* to learn how to use this protocol.

Tasks

Customizing Tab Bars

- `tabBarController:didSelectViewController:` (page 576)
  Sent to the delegate after a view controller is selected.
- `tabBarController:didEndCustomizingViewControllers:changed:` (page 576)
  Sent to the delegate after the customizing sheet is dismissed.

Instance Methods

tabBarController:didEndCustomizingViewControllers:changed:

Sent to the delegate after the customizing sheet is dismissed.

```
- (void)tabBarController:(UITabBarController *)tabBarController
    didEndCustomizingViewControllers:(NSArray *)viewControllers changed:(BOOL)changed
```

Parameters

tabBarController

The tab bar controller that is being customized.

viewControllers

The view controllers that changed.

changed

A Boolean value indicating whether items changed on the tab bar. `YES` if items changed; otherwise, `NO`.

Discussion

Use this method to update the view depending on the items the user selected for the tab bar.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITabBarController.h`

tabBarController:didSelectViewController:

Sent to the delegate after a view controller is selected.

```
- (void)tabBarController:(UITabBarController *)tabBarController
    didSelectViewController:(UIViewController *)viewController
```

Parameters

tabBarController

The tab bar controller containing *viewController*.

viewController

The view controller that was selected.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITabBarController.h`

UITableViewDataSource Protocol Reference

Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITableView.h
Companion guide:	Table View Programming Guide for iPhone OS

Overview

The `UITableViewDataSource` protocol is adopted by a object that mediates the application's data model for a `UITableView` object. The data source provides the table-view object with the information it needs to construct and modify a table view.

As a representative of the data model, the data source supplies minimal information about the table view's appearance, The table-view object's delegate—an object adopting the `UITableViewDelegate` protocol—provides that information.

The required methods of the protocol provide the cells display to be displayed by the table-view as well as inform the `UITableView` object about the number of sections and the number of rows in each section.

The data source may implement optional methods to configure various aspects of the table view and to insert, delete, and reorder rows. Many methods take `NSIndexPath` objects as parameters. `UITableView` declares a category on `NSIndexPath` that enables you to get the represented row index (`row` (page 36) property) and section index (`section` (page 36) property), and to construct an index path from a given row index and section index (`indexPathForRow:inSection:` (page 36) class method). (The first index in each index path identifies the section and the next identifies the row.)

Tasks

Configuring a Table View

- `tableView:cellForRowAtIndexPath:` (page 581)
  Asks the data source for a cell to insert in a particular location of the table view.
- `numberOfSectionsInTableView:` (page 579) *optional method*
  Asks the data source to return the number of sections in the table view. This method is optional.
- `tableView:numberOfRowsInSection:` (page 582)
  Tells the data source to return the number of rows in a given section of a table view.
- `sectionIndexTitlesForTableView:` (page 579) *optional method*
  Asks the data source to return the titles for the sections for a table view. This method is optional.
- `tableView:sectionForSectionIndexTitle:atIndex:` (page 583) *optional method*
  Asks the data source to return the index of the section having the given title and section title index. This method is optional.
- `tableView:titleForHeaderInSection:` (page 584) *optional method*
  Asks the data source for the title of the header of the specified section of the table view. This method is optional.
- `tableView:titleForFooterInSection:` (page 584) *optional method*
  Asks the data source for the title of the footer of the specified section of the table view. This method is optional.

Inserting or Deleting Table Rows

- `tableView:commitEditingStyle:forRowAtIndexPath:` (page 581) *optional method*
  Asks the data source to commit the insertion or deletion of a specified row in the receiver. This method is optional.
- `tableView:canEditRowAtIndexPath:` (page 579) *optional method*
  Asks the data source to verify that the given row is editable. This method is optional.

Reordering Table Rows

- `tableView:canMoveRowAtIndexPath:` (page 580) *optional method*
  Asks the data source whether a given row can be moved to another location in the table view. This method is optional.
- `tableView:moveRowAtIndexPath:toIndexPath:` (page 582) *optional method*
  Tells the data source to move a row at a specific location in the table view to another location. This method is optional.

Instance Methods

numberOfSectionsInTableView:

Asks the data source to return the number of sections in the table view. This method is optional.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
```

Parameters

tableView

An object representing the table view requesting this information.

Return Value

The number of sections in *tableView*.The default value is 1. Values less than 1 are not valid and will cause an assertion to fail immediately.

Availability

Available in iPhone OS 2.0 and later.

See Also

- tableView:numberOfRowsInSection: (page 582)

Declared In

UITableView.h

sectionIndexTitlesForTableView:

Asks the data source to return the titles for the sections for a table view. This method is optional.

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView
```

Parameters

tableView

The table-view object requesting this information.

Return Value

An array of strings that serve as the title of sections in the table view and appear in the index list on the right side of the table view. The table view must be in the plain style (`UITableViewStylePlain`). For example, for an alphabetized list, you could return an array containing strings "A" through "Z".

Availability

Available in iPhone OS 2.0 and later.

See Also

- tableView:sectionForSectionIndexTitle:atIndex: (page 583)

Declared In

UITableView.h

tableView:canEditRowAtIndexPath:

Asks the data source to verify that the given row is editable. This method is optional.

```
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath
    *)indexPath
```

Parameters

tableView

The table-view object requesting this information.

indexPath

An index path locating a row in *tableView*.

Return Value

YES if the row indicated by *indexPath* is editable; otherwise, NO.

Discussion

The method permits the delegate to exclude individual rows from being treated as editable. Editable rows display the insertion or deletion control in their cells. If this method is not implemented, all rows are assumed to be editable. Rows that are not editable ignore the editingStyle (page 387) property of a UITableViewCell object and do no indentation for the deletion or insertion control. Rows that are editable, but that do not want to have an insertion or remove control shown, can return UITableViewCellEditingStyleNone (page 397) from the tableView:editingStyleForRowAtIndexPath: (page ?) delegate method.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITableView.h

tableView:canMoveRowAtIndexPath:

Asks the data source whether a given row can be moved to another location in the table view. This method is optional.

```
- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath
    *)indexPath
```

Parameters

tableView

The table-view object requesting this information.

indexPath

An index path locating a row in *tableView*.

Return Value

YES if the row can be moved; otherwise NO.

Discussion

This method allows the delegate to specify that the reordering control for a the specified row not be shown. By default, the reordering control is shown if the data source implements the tableView:moveRowAtIndexPath:toIndexPath: (page 582) method.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITableView.h

tableView:cellForRowAtIndexPath:

Asks the data source for a cell to insert in a particular location of the table view.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView

A table-view object requesting the cell.

indexPath

An index path locating a row in *tableView*.

Return Value

An object inheriting from `UITableViewCell` that the table view can use for the specified row. An assertion is raised if you return `nil`.

Discussion

The returned `UITableViewCell` object is frequently one that the application reuses for performance reasons. You should fetch a previously created cell object that is marked for reuse by sending a `dequeueReusableCellWithIdentifier:` (page 367) message to *tableView*. The identifier for a reusable cell object is assigned when the delegate initializes the cell object by calling the `initWithFrame:reuseIdentifier:` (page 394) method of `UITableViewCell`. Various attributes of a table cell are set automatically based on whether the cell is a separator and on information the data source provides, such as for accessory views and editing controls.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

tableView:commitEditingStyle:forRowAtIndexPath:

Asks the data source to commit the insertion or deletion of a specified row in the receiver. This method is optional.

```
- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView

The table-view object requesting the insertion or deletion.

editingStyle

The cell editing style corresponding to a insertion or deletion requested for the row specified by *indexPath*. Possible editing styles are `UITableViewCellEditingStyleInsert` (page 397) or `UITableViewCellEditingStyleDelete` (page 397).

indexPath

An index path locating the row in *tableView*.

Discussion
When users click the insertion (green plus) or deletion (red minus) control of a `UITableViewCell` object in the table view, the table view sends this message to the data source, asking it to commit the change. The data source commits the insertion or deletion by invoking the `UITableView` methods `insertRowsAtIndexPaths:withRowAnimation:` (page 371) or `deleteRowsAtIndexPaths:withRowAnimation:` (page 366), as appropriate.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

tableView:moveRowAtIndexPath:toIndexPath:

Tells the data source to move a row at a specific location in the table view to another location. This method is optional.

```
- (void)tableView:(UITableView *)tableView  moveRowAtIndexPath:(NSIndexPath
    *)fromIndexPath  toIndexPath:(NSIndexPath *)toIndexPath
```

Parameters
tableView
The table-view object requesting this action.

fromIndexPath
An index path locating the row to be moved in *tableView*.

toIndexPath
An index path locating the row in *tableView* that is the destination of the move.

Discussion
The `UITableView` object sends this message to the data source when the user presses the reorder control in *fromRow*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- tableView:commitEditingStyle:forRowAtIndexPath:` (page 581)

Declared In
`UITableView.h`

tableView:numberOfRowsInSection:

Tells the data source to return the number of rows in a given section of a table view.

```
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
```

Parameters
tableView
The table-view object requesting this information.

section
An index number identifying a section in *tableView*.

Return Value
The number of rows in `section`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- numberOfSectionsInTableView:` (page 579)

Declared In
`UITableView.h`

tableView:sectionForSectionIndexTitle:atIndex:

Asks the data source to return the index of the section having the given title and section title index. This method is optional.

```
- (NSInteger)tableView:(UITableView *)tableView sectionForSectionIndexTitle:(NSString
    *)title atIndex:(NSInteger)index
```

Parameters
tableView
The table-view object requesting this information.

title
The title as displayed in the section index of *tableView*.

index
An index number identifying a section title in the array returned by `sectionIndexTitlesForTableView:` (page 579).

Return Value
An index number identifying a section.

Discussion
This method is passed the index number and title of an entry in the section index list and should return the index of the referenced section. To be clear, there are two index numbers in play here: an index to an section index title in the array returned by `sectionIndexTitlesForTableView:`, and an index to a section of the table view; the former is passed in, and the latter is returned. You implement this method only for table views with a section index list—which can only be table views created in the plain style (`UITableViewStylePlain` (page 378)). Note that the array of section titles returned by `sectionIndexTitlesForTableView:` can have fewer items than the actual number of sections in the table view.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- numberOfSectionsInTableView:` (page 579)

Declared In
`UITableView.h`

tableView:titleForFooterInSection:

Asks the data source for the title of the footer of the specified section of the table view. This method is optional.

```
- (NSString *)tableView:(UITableView *)tableView
    titleForFooterInSection:(NSInteger)section
```

Parameters

tableView

The table-view object asking for the title.

section

An index number identifying a section of *tableView*.

Return Value

A string to use as the title of the section footer. If you return `nil`, the section will have no title.

Discussion

The table view uses a fixed font style for section footer titles. If you want a different font style, return a custom view (for example, a `UILabel` object) in the delegate method `tableView:viewForFooterInSection:` (page ?) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- tableView:titleForHeaderInSection:` (page 584)

Declared In

`UITableView.h`

tableView:titleForHeaderInSection:

Asks the data source for the title of the header of the specified section of the table view. This method is optional.

```
- (NSString *)tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section
```

Parameters

tableView

The table-view object asking for the title.

section

An index number identifying a section of *tableView*.

Return Value

A string to use as the title of the section header. If you return `nil`, the section will have no title.

Discussion

The table view uses a fixed font style for section header titles. If you want a different font style, return a custom view (for example, a `UILabel` object) in the delegate method `tableView:viewForHeaderInSection:` (page ?) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also
- tableView:titleForFooterInSection: (page 584)

Declared In
UITableView.h

UITableViewDelegate Protocol Reference

Conforms to:	NSObject (NSObject) UIScrollViewDelegate
Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITableView.h
Companion guide:	Table View Programming Guide for iPhone OS

Overview

The delegate of a `UITableView` object must adopt the `UITableViewDelegate` protocol. Optional methods of the protocol allow the delegate to manage selections, configure section headings and footers, help to delete and reorder cells, and perform other actions.

Many methods of `UITableViewDelegate` take `NSIndexPath` objects as parameters and return values. `UITableView` declares a category on `NSIndexPath` that enables you to get the represented row index (`row` (page 36) property) and section index (`section` (page 36) property), and to construct an index path from a given row index and section index (`indexPathForRow:inSection:` (page 36) method). Because rows are located within their sections, you usually must evaluate the section index number before you can identify the row by its index number.

Tasks

Providing Table Cells for the Table View

`- tableView:heightForRowAtIndexPath:` (page 593) *optional method*

Asks the delegate for the height to use for a row in a specified location. This method is optional.

- `tableView:indentationLevelForRowAtIndexPath:` (page 593) *optional method*
  Asks the delegate to return the level of indentation for a row in a given section. This method is optional.
- `tableView:willDisplayCell:forRowAtIndexPath:` (page 596) *optional method*
  Tells the delegate the table view is about to draw a cell for a particular row. This method is optional.

Managing Accessory Views

- `tableView:accessoryTypeForRowWithIndexPath:` (page 589) *optional method*
  Asks the delegate for the type of standard accessory view to use as a disclosure control for the specified row. This method is optional.
- `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 589) *optional method*
  Tells the delegate that the user tapped the accessory (disclosure) view associated with a given row. This method is optional.

Managing Selections

- `tableView:willSelectRowAtIndexPath:` (page 597) *optional method*
  Tells the delegate that a specified row is about to be selected. This method is optional.
- `tableView:didSelectRowAtIndexPath:` (page 590) *optional method*
  Tells the delegate that the specified row is now selected. This method is optional.

Modifying the Header and Footer of Sections

- `tableView:viewForHeaderInSection:` (page 595) *optional method*
  Asks the delegate for a view object to display in the header of the specified section of the table view. This method is optional.
- `tableView:viewForFooterInSection:` (page 595) *optional method*
  Asks the delegate for a view object to display in the footer of the specified section of the table view. This method is optional.
- `tableView:heightForHeaderInSection:` (page 592) *optional method*
  Asks the delegate for the height to use for the header of a particular section. This method is optional.
- `tableView:heightForFooterInSection:` (page 591) *optional method*
  Asks the delegate for the height to use for the footer of a particular section. This method is optional.

Editing Table Rows

- `tableView:willBeginEditingRowAtIndexPath:` (page 596) *optional method*
  Tells the delegate that the table view is about to go into editing mode. This method is optional.
- `tableView:didEndEditingRowAtIndexPath:` (page 590) *optional method*
  Tells the delegate that the table view has left editing mode. This method is optional.

- `tableView:editingStyleForRowAtIndexPath:` (page 591) *optional method*
  Asks the delegate for the editing style of a row at a particular location in a table view. This method is optional.
- `tableView:shouldIndentWhileEditingRowAtIndexPath:` (page 594)
  Asks the delegate whether the background of the specified row should be indented while the table view is in editing mode.

Reordering Table Rows

- `tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:` (page 594)
  Asks the delegate to return a new index path to retarget a proposed move of a row.

Instance Methods

tableView:accessoryButtonTappedForRowWithIndexPath:

Tells the delegate that the user tapped the accessory (disclosure) view associated with a given row. This method is optional.

```
- (void)tableView:(UITableView *)tableView
    accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView
The table-view object informing the delegate of this event.

indexPath
An index path locating the row in *tableView*.

Discussion

The delegate usually responds to the the tap on the disclosure button (the accessory view) by displaying a new view related to the selected row.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

tableView:accessoryTypeForRowWithIndexPath:

Asks the delegate for the type of standard accessory view to use as a disclosure control for the specified row. This method is optional.

```
- (UITableViewCellAccessoryType)tableView:(UITableView *)tableView
    accessoryTypeForRowWithIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView

The table-view object requesting the accessory-view type.

indexPath

An index path locating the row in `tableView`.

Return Value

A constant identifying a type of standard accessory view. For details, see the "Constants" section in the class reference for the `UITableViewCell` class.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- tableView:accessoryButtonTappedForRowWithIndexPath:` (page 589)

Declared In

`UITableView.h`

tableView:didEndEditingRowAtIndexPath:

Tells the delegate that the table view has left editing mode. This method is optional.

```
- (void)tableView:(UITableView *)tableView didEndEditingRowAtIndexPath:(NSIndexPath
    *)indexPath
```

Parameters

tableView

The table-view object providing this information.

indexPath

An index path locating the row in *tableView*.

Discussion

This method is called when the table view exits editing mode after having been put into the mode by the user swiping across the row identified by `indexPath`. (As a result, a Delete button appears in the row.) When entering editing mode in that way, the table view sends a `tableView:willBeginEditingRowAtIndexPath:` (page 596) message to the delegate to allow it to adjusts its user interface. In editing mode the table displays the insertion, deletion, and reordering controls that have been associated with rows.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

tableView:didSelectRowAtIndexPath:

Tells the delegate that the specified row is now selected. This method is optional.

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
    *)indexPath
```

Parameters

tableView

A table-view object informing the delegate about the new row selection.

indexPath

An index path locating the new selected row in *tableView*.

Discussion

The delegate handles selections in this method. One of the things it can do is exclusively assign the check-mark image (`UITableViewCellAccessoryCheckmark` (page 398)) to one row in a section (radio-list style). See "Managing Selections" in *Table View Programming Guide for iPhone OS* for further information (and code examples) related to this method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- tableView:willSelectRowAtIndexPath:` (page 597)

Declared In

`UITableView.h`

tableView:editingStyleForRowAtIndexPath:

Asks the delegate for the editing style of a row at a particular location in a table view. This method is optional.

```
- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView
    editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView

The table-view object requesting this information.

indexPath

An index path locating a row in *tableView*.

Return Value

The editing style of the cell for the row identified by *indexPath*.

Discussion

This method allows the delegate to customize the editing style of the cell located at *indexPath*. If the delegate does not implement this method and the `UITableViewCell` object is editable (that is, it has its `editing` (page 386) property set to `YES`), the cell has the `UITableViewCellEditingStyleDelete` (page 397) style set for it.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITableView.h`

tableView:heightForFooterInSection:

Asks the delegate for the height to use for the footer of a particular section. This method is optional.

```
- (CGFloat)tableView:(UITableView *)tableView
    heightForFooterInSection:(NSInteger)section
```

Parameters

tableView

The table-view object requesting this information.

section

An index number identifying a section of *tableView*.

Return Value

A floating-point value that specifies the height (in points) of the footer for *section*.

Discussion

This method allows the delegate to specify section footers with varying heights. The table view does not call this method if it was created in a plain style (`UITableViewStylePlain` (page 378)).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:` (page 594)
- `tableView:viewForFooterInSection:` (page 595)

Declared In

`UITableView.h`

tableView:heightForHeaderInSection:

Asks the delegate for the height to use for the header of a particular section. This method is optional.

```
- (CGFloat)tableView:(UITableView *)tableView
    heightForHeaderInSection:(NSInteger)section
```

Parameters

tableView

The table-view object requesting this information.

section

An index number identifying a section of *tableView*.

Return Value

A floating-point value that specifies the height (in points) of the header for *section*.

Discussion

This method allows the delegate to specify section headers with varying heights. The table view does not call this method if it was created in a plain style (`UITableViewStylePlain` (page 378)).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `tableView:willDisplayCell:forRowAtIndexPath:` (page 596)
- `tableView:viewForHeaderInSection:` (page 595)

Declared In
UITableView.h

tableView:heightForRowAtIndexPath:

Asks the delegate for the height to use for a row in a specified location. This method is optional.

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath
    *)indexPath
```

Parameters

tableView
The table-view object requesting this information.

indexPath
An index path that locates a row in *tableView*.

Return Value
A floating-point value that specifies the height (in points) that *row* should be.

Discussion
The method allows the delegate to specify rows with varying heights.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

tableView:indentationLevelForRowAtIndexPath:

Asks the delegate to return the level of indentation for a row in a given section. This method is optional.

```
- (NSInteger)tableView:(UITableView *)tableView
    indentationLevelForRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView
The table-view object requesting this information.

indexPath
An index path locating the row in *tableView*.

Return Value
Returns the depth of the specified row to show its hierarchical position in the section.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

tableView:shouldIndentWhileEditingRowAtIndexPath:

Asks the delegate whether the background of the specified row should be indented while the table view is in editing mode.

```
- (BOOL)tableView:(UITableView *)tableView
    shouldIndentWhileEditingRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView

The table-view object requesting this information.

indexPath

An index-path object locating the row in its section.

Return Value

YES if the background of the row should be indented, otherwise NO.

Discussion

If the delegate does not implement this method, the default is YES. This method is unrelated to tableView:indentationLevelForRowAtIndexPath: (page 593).

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITableView.h

tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:

Asks the delegate to return a new index path to retarget a proposed move of a row.

```
- (NSIndexPath *)tableView:(UITableView *)tableView
    targetIndexPathForMoveFromRowAtIndexPath:(NSIndexPath *)sourceIndexPath
    toProposedIndexPath:(NSIndexPath *)proposedDestinationIndexPath
```

Parameters

tableView

The table-view object that is requesting this information.

sourceIndexPath

An index-path object identifying the original location of a row (in its section) that is being dragged.

proposedDestinationIndexPath

An index-path object identifying the currently proposed destination of the row being dragged.

Return Value

An index-path object locating the desired row destination for the move operation. Return *proposedDestinationIndexPath* if that location is suitable.

Discussion

This method allows customization of the target row for a particular row as it is being moved up and down a table view. As the dragged row hovers over a another row, the destination row slides downward to visually make room for the relocation; this is the location identified by *proposedDestinationIndexPath*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITableView.h

tableView:viewForFooterInSection:

Asks the delegate for a view object to display in the footer of the specified section of the table view. This method is optional.

```
- (UIView *)tableView:(UITableView *)tableView
    viewForFooterInSection:(NSInteger)section
```

Parameters
tableView
The table-view object asking for the view object.

section
An index number identifying a section of *tableView*.

Return Value
A view object to be displayed in the footer of *section*.

Discussion
The returned object, for example, can be a UILabel or UIImageView object. The table view automatically adjusts the height of the section footer to accommodate the returned view object. The table view does not call this method if it was created in a plain style (UITableViewStylePlain (page 378)).

Availability
Available in iPhone OS 2.0 and later.

See Also
- tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath: (page 594)
- tableView:heightForFooterInSection: (page 591)

Declared In
UITableView.h

tableView:viewForHeaderInSection:

Asks the delegate for a view object to display in the header of the specified section of the table view. This method is optional.

```
- (UIView *)tableView:(UITableView *)tableView
    viewForHeaderInSection:(NSInteger)section
```

Parameters
tableView
The table-view object asking for the view object.

section
An index number identifying a section of *tableView*.

Return Value
A view object to be displayed in the header of *section*.

Discussion
The returned object, for example, can be a `UILabel` or `UIImageView` object. The table view automatically adjusts the height of the section header to accommodate the returned view object. The table view does not call this method if it was created in a plain style (`UITableViewStylePlain` (page 378)).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `tableView:willDisplayCell:forRowAtIndexPath:` (page 596)
- `tableView:heightForHeaderInSection:` (page 592)

Declared In
`UITableView.h`

tableView:willBeginEditingRowAtIndexPath:

Tells the delegate that the table view is about to go into editing mode. This method is optional.

```
- (void)tableView:(UITableView *)tableView
    willBeginEditingRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters
tableView
The table-view object providing this information.

indexPath
An index path locating the row in *tableView*.

Discussion
This method is called when the user swipes horizontally across a row; as a consequence, the table view sets its `editing` (page 362) property to `YES` (thereby entering editing mode) and displays a Delete button in the row identified by `indexPath`. In editing mode the table view displays the insertion, deletion, and reordering controls that have been associated with rows. This method gives the delegate an opportunity to adjust the application's user interface to editing mode. When the table exits editing mode (for example, the user taps the Delete button), the table view calls `tableView:didEndEditingRowAtIndexPath:` (page 590).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITableView.h`

tableView:willDisplayCell:forRowAtIndexPath:

Tells the delegate the table view is about to draw a cell for a particular row. This method is optional.

```
- (void)tableView:(UITableView *)tableView willDisplayCell:(UITableViewCell *)cell
    forRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView
The table-view object informing the delegate of this impending event.

cell
A table-view cell object that *tableView* is going to use when drawing the row.

indexPath
An index path locating the row in *tableView*.

Discussion

A table view sends this message to its delegate just before it uses *cell* to draw a row, thereby permitting the delegate to customize the cell object before it is displayed. This method gives the delegate a chance to override state-based properties set earlier by the table view, such as selection and background color. After the delegate returns, the table view sets only the alpha and frame properties, and then only when animating rows as they slide in or out.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `tableView:cellForRowAtIndexPath:` (page 581) (`UITableViewDataSource`)
- `prepareForReuse` (page 395) (`UITableViewCell`)

Declared In

`UITableView.h`

tableView:willSelectRowAtIndexPath:

Tells the delegate that a specified row is about to be selected. This method is optional.

```
- (NSIndexPath *)tableView:(UITableView *)tableView
    willSelectRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView
A table-view object informing the delegate about the impending selection.

indexPath
An index path locating the row in *tableView*.

Return Value

An index-path object that confirms or alters the selected row. Return an `NSIndexPath` object other than *indexPath* if you want another cell to be selected. Return `nil` if you don't want the row selected.

Discussion

This method is not called until users touch a row and then lift their finger; the row isn't selected until then, although it is highlighted on touch-down. You can use `UITableViewCellSelectionStyleNone` to disable the appearance of the cell highlight on touch-down.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `tableView:didSelectRowAtIndexPath:` (page 590)
- `tableView:shouldIndentWhileEditingRowAtIndexPath:` (page 594)

Declared In
UITableView.h

UITextFieldDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITextField.h

Overview

The `UITextFieldDelegate` protocol defines the messages sent to a text field delegate as part of the sequence of editing its text. All of the methods of this protocol are optional.

Tasks

Managing Editing

- `textFieldShouldBeginEditing:` (page 601) *optional method*
  Asks the delegate if editing should begin in the specified text field. This method is optional.
- `textFieldDidBeginEditing:` (page 600) *optional method*
  Tells the delegate that editing began for the specified text field. This method is optional.
- `textFieldShouldEndEditing:` (page 602) *optional method*
  Asks the delegate if editing should stop in the specified text field. This method is optional.
- `textFieldDidEndEditing:` (page 601) *optional method*
  Tells the delegate that editing stopped for the specified text field. This method is optional.

Editing the Text Field's Text

- `textField:shouldChangeCharactersInRange:replacementString:` (page 600) *optional method*
  Asks the delegate if the specified text should be changed. This method is optional.
- `textFieldShouldClear:` (page 602) *optional method*
  Asks the delegate if the text field's current contents should be removed. This method is optional.

- textFieldShouldReturn: (page 603) *optional method*
  Asks the delegate if the text field should process the pressing of the return button. This method is optional.

Instance Methods

textField:shouldChangeCharactersInRange:replacementString:

Asks the delegate if the specified text should be changed. This method is optional.

```
- (BOOL)textField:(UITextField *)textField
    shouldChangeCharactersInRange:(NSRange)range replacementString:(NSString *)string
```

Parameters

textField
The text field containing the text.

range
The range of characters to be replaced

string
The replacement string.

Return Value

YES if the specified text range should be replaced; otherwise, NO to keep the old text.

Discussion

The text field calls this method whenever the user types a new character in the text field or deletes an existing character.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITextField.h

textFieldDidBeginEditing:

Tells the delegate that editing began for the specified text field. This method is optional.

```
- (void)textFieldDidBeginEditing:(UITextField *)textField
```

Parameters

textField
The text field for which an editing session began.

Discussion

This method notifies the delegate that the specified text field just became the first responder. You can use this method to update your delegate's state information. For example, you might use this method to show overlay views that should be visible while editing.

Implementation of this method by the delegate is optional.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

textFieldDidEndEditing:

Tells the delegate that editing stopped for the specified text field. This method is optional.

```
- (void)textFieldDidEndEditing:(UITextField *)textField
```

Parameters

textField
The text field for which editing ended.

Discussion
This method is called after the text field resigns its first responder status. You can use this method to update your delegate's state information. For example, you might use this method to hide overlay views that should be visible only while editing.

Implementation of this method by the delegate is optional.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

textFieldShouldBeginEditing:

Asks the delegate if editing should begin in the specified text field. This method is optional.

```
- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField
```

Parameters

textField
The text field for which editing is about to begin.

Return Value
YES if an editing session should be initiated; otherwise, NO to disallow editing.

Discussion
When the user performs an action that would normally initiate an editing session, the text field calls this method first to see if editing should actually proceed. In most circumstances, you would simply return YES from this method to allow editing to proceed.

Implementation of this method by the delegate is optional. If it is not present, editing proceeds as if this method had returned YES.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

textFieldShouldClear:

Asks the delegate if the text field's current contents should be removed. This method is optional.

```
- (BOOL)textFieldShouldClear:(UITextField *)textField
```

Parameters

textField
The text field containing the text.

Return Value
YES if the text field's contents should be cleared; otherwise, NO.

Discussion
The text field calls this method in response to the user pressing the built-in clear button. (This button is not shown by default but can be enabled by changing the value in the clearButtonMode (page 409) property of the text field.) This method is also called when editing begins and the clearsOnBeginEditing (page 410) property of the text field is set to YES.

Implementation of this method by the delegate is optional. If it is not present, the text is cleared as if this method had returned YES.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextField.h

textFieldShouldEndEditing:

Asks the delegate if editing should stop in the specified text field. This method is optional.

```
- (BOOL)textFieldShouldEndEditing:(UITextField *)textField
```

Parameters

textField
The text field for which editing is about to end.

Return Value
YES if editing should stop; otherwise, NO if the editing session should continue

Discussion
This method is called when the text field is asked to resign the first responder status. This might occur when your application asks the text field to resign focus or when the user tries to change the editing focus to another control. Before the focus actually changes, however, the text field calls this method to give your delegate a chance to decide whether it should.

Normally, you would return YES from this method to allow the text field to resign the first responder status. You might return NO, however, in cases where your delegate detects invalid contents in the text field. By returning NO, you could prevent the user from switching to another control until the text field contained a valid value.

Note: If you use this method to validate the contents of the text field, you might also want to provide feedback to that effect using an overlay view. For example, you could temporarily display a small icon indicating the text was invalid and needs to be corrected. For more information about adding overlays to text fields, see the methods of `UITextField`.

Be aware that this method provides only a recommendation about whether editing should end. Even if you return `NO` from this method, it is possible that editing might still end. For example, this might happen when the text field is forced to resign the first responder status by being removed from its parent view or window.

Implementation of this method by the delegate is optional. If it is not present, the first responder status is resigned as if this method had returned `YES`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

textFieldShouldReturn:

Asks the delegate if the text field should process the pressing of the return button. This method is optional.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField
```

Parameters
`textField`
The text field whose return button was pressed.

Return Value
`YES` if the text field should implement its default behavior for the return button; otherwise, `NO`.

Discussion
The text field calls this method whenever the user taps the return button. You can use this method to implement any custom behavior when the button is tapped.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextField.h`

UITextInputTraits Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITextInputTraits.h

Overview

The `UITextInputTraits` protocol defines features that are associated with keyboard input. All objects that support keyboard input must adopt this protocol in order to interact properly with the text input management system. The `UITextField` and `UITextView` classes already support this protocol.

Tasks

Managing the Keyboard Behavior

`autocapitalizationType` (page 606) *property*
The auto-capitalization style for the text object.

`autocorrectionType` (page 606) *property*
The auto-correction style for the text object.

`enablesReturnKeyAutomatically` (page 606) *property*
A Boolean value indicating whether the return key is automatically enabled when text is entered by the user.

`keyboardAppearance` (page 607) *property*
The appearance style of the keyboard that is associated with the text object

`keyboardType` (page 607) *property*
The keyboard style associated with the text object.

`returnKeyType` (page 607) *property*
The contents of the “return” key.

`secureTextEntry` (page 608) *property*
Identifies whether the text object should hide the text being entered.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

autocapitalizationType

The auto-capitalization style for the text object.

```
@property(nonatomic) UITextAutocapitalizationType autocapitalizationType
```

Discussion

This property determines at what times the Shift key is automatically pressed, thereby making the typed character a capital letter. The default value for this property is `UITextAutocapitalizationTypeNone`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextInputTraits.h`

autocorrectionType

The auto-correction style for the text object.

```
@property(nonatomic) UITextAutocorrectionType autocorrectionType
```

Discussion

This property determines whether auto-correction is enabled or disabled during typing. With auto-correction enabled, the text object tracks unknown words and suggests a more suitable replacement candidate to the user, replacing the typed text automatically unless the user explicitly overrides the action.

The default value for this property is `UITextAutocorrectionTypeDefault`, which for most input methods results in auto-correction being enabled.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextInputTraits.h`

enablesReturnKeyAutomatically

A Boolean value indicating whether the return key is automatically enabled when text is entered by the user.

```
@property(nonatomic) BOOL enablesReturnKeyAutomatically
```

Discussion
The default value for this property is `NO`. If you set it to `YES`, the keyboard disables the return key when the text entry area contains no text. As soon as the user enters any text, the return key is automatically enabled.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextInputTraits.h`

keyboardAppearance

The appearance style of the keyboard that is associated with the text object

```
@property(nonatomic) UIKeyboardAppearance keyboardAppearance
```

Discussion
This property lets you distinguish between the default text entry inside your application and text entry inside an alert panel. The default value for this property is `UIKeyboardAppearanceDefault`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextInputTraits.h`

keyboardType

The keyboard style associated with the text object.

```
@property(nonatomic) UIKeyboardType keyboardType
```

Discussion
Text objects can be targeted for specific types of input, such as plain text, email, numeric entry, and so on. The keyboard style identifies what keys are available on the keyboard and which ones appear by default. The default value for this property is `UIKeyboardTypeDefault`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextInputTraits.h`

returnKeyType

The contents of the “return” key.

```
@property(nonatomic) UIReturnKeyType returnKeyType
```

Discussion
Setting this property to a different key type changes the title of the key and typically results in the keyboard being dismissed when it is pressed. The default value for this property is `UIReturnKeyDefault`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextInputTraits.h`

secureTextEntry

Identifies whether the text object should hide the text being entered.

```
@property(nonatomic, getter=isSecureTextEntry) BOOL secureTextEntry
```

Discussion
This property is set to `NO` by default. Setting this property to `YES` creates a password-style text object, which hides the text being entered.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UITextInputTraits.h`

Constants

UITextAutocapitalizationType

The auto-capitalization behavior of a text-based view.

```
typedef enum {
    UITextAutocapitalizationTypeNone,
    UITextAutocapitalizationTypeWords,
    UITextAutocapitalizationTypeSentences,
    UITextAutocapitalizationTypeAllCharacters,
} UITextAutocapitalizationType;
```

Constants

`UITextAutocapitalizationTypeNone`
Do not capitalize any text automatically.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UITextAutocapitalizationTypeWords`
Capitalize the first letter of each word automatically.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UITextAutocapitalizationTypeSentences`
Capitalize the first letter of each sentence automatically.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UITextAutocapitalizationTypeAllCharacters`
Capitalize all characters automatically.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

Discussion
If the script system does not support capitalization, the keyboard input method ignores these constants.

UITextAutocorrectionType

The auto-correction behavior of a text-based view.

```
typedef enum {
    UITextAutocorrectionTypeDefault,
    UITextAutocorrectionTypeNo,
    UITextAutocorrectionTypeYes,
} UITextAutocorrectionType;
```

Constants
`UITextAutocorrectionTypeDefault`
Choose an appropriate auto-correction behavior for the current script system.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UITextAutocorrectionTypeNo`
Disable auto-correction behavior.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UITextAutocorrectionTypeYes`
Enable auto-correction behavior.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

Discussion
If the script system does not support inline auto-correction, the keyboard input method ignores these constants.

UIKeyboardType

The type of keyboard to display for a given text-based view.

```
typedef enum {
    UIKeyboardTypeDefault,
    UIKeyboardTypeASCIICapable,
    UIKeyboardTypeNumbersAndPunctuation,
    UIKeyboardTypeURL,
    UIKeyboardTypeNumberPad,
    UIKeyboardTypePhonePad,
    UIKeyboardTypeNamePhonePad,
    UIKeyboardTypeEmailAddress,
} UIKeyboardType;
```

Constants

`UIKeyboardTypeDefault`
Use the default keyboard for the current input method.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypeASCIICapable`
Use a keyboard that displays standard ASCII characters.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypeNumbersAndPunctuation`
Use the numbers and punctuation keyboard.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypeURL`
Use a keyboard optimized for URL entry. This type features ".", "/", and ".com" prominently.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypeNumberPad`
Use a numeric keypad designed for PIN entry. This type features the numbers 0 through 9 prominently.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypePhonePad`
Use a keypad designed for entering telephone numbers. This type features the numbers 0 through 9 and the "*" and "#" characters prominently.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypeNamePhonePad`
Use a keypad designed for entering a person's name or phone number.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardTypeEmailAddress`
Use a keyboard optimized for specifying email addresses. This type features the "@", "." and space characters prominently.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

UIKeyboardAppearance

The appearance of the keyboard used by a text-based view.

```
typedef enum {
    UIKeyboardAppearanceDefault,
    UIKeyboardAppearanceAlert,
} UIKeyboardAppearance;
```

Constants

`UIKeyboardAppearanceDefault`

Use the default keyboard appearance for the current input method.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIKeyboardAppearanceAlert`

Use a keyboard that is suitable for an alert panel.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

UIReturnKeyType

The text string displayed in the “return” key of a keyboard.

```
typedef enum {
    UIReturnKeyDefault,
    UIReturnKeyGo,
    UIReturnKeyGoogle,
    UIReturnKeyJoin,
    UIReturnKeyNext,
    UIReturnKeyRoute,
    UIReturnKeySearch,
    UIReturnKeySend,
    UIReturnKeyYahoo,
    UIReturnKeyDone,
    UIReturnKeyEmergencyCall,
} UIReturnKeyType;
```

Constants

`UIReturnKeyDefault`

Set the text of the return key to “return”.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIReturnKeyGo`

Set the text of the return key to “Go”.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIReturnKeyGoogle`

Set the text of the return key to “Google”.

Available in iPhone OS 2.0 and later.

Declared in `UITextInputTraits.h`

`UIReturnKeyJoin`
Set the text of the return key to “Join”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeyNext`
Set the text of the return key to “Next”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeyRoute`
Set the text of the return key to “Route”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeySearch`
Set the text of the return key to “Search”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeySend`
Set the text of the return key to “Send”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeyYahoo`
Set the text of the return key to “Yahoo”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeyDone`
Set the text of the return key to “Done”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

`UIReturnKeyEmergencyCall`
Set the text of the return key to “Emergency Call”.
Available in iPhone OS 2.0 and later.
Declared in `UITextInputTraits.h`

UITextViewDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UITextView.h

Overview

The `UITextViewDelegate` protocol defines a set of optional methods you can use to receive editing-related messages for `UITextView` objects. All of the methods in this protocol are optional. You can use them in situations where you might want to adjust the text being edited (such as in the case of a spell checker program) or modify the intended insertion point.

Tasks

Responding to Editing Notifications

- `textViewShouldBeginEditing:` (page 616) *optional method*
  Asks the delegate if editing should begin in the specified text view. This method is optional.
- `textViewDidBeginEditing:` (page 615) *optional method*
  Tells the delegate that editing of the specified text view has begun. This method is optional.
- `textViewShouldEndEditing:` (page 617)
  Asks the delegate if editing should stop in the specified text view.
- `textViewDidEndEditing:` (page 616) *optional method*
  Tells the delegate that editing of the specified text view has ended. This method is optional.

Responding to Text Changes

`- textView:shouldChangeTextInRange:replacementText:` (page 614) *optional method*
Asks the delegate whether the specified text should be replaced in the text view. This method is optional.

`- textViewDidChange:` (page 615) *optional method*
Tells the delegate that the text or attributes in the specified text view changed. This method is optional.

Responding to Selection Changes

`- textViewDidChangeSelection:` (page 615) *optional method*
Tells the delegate that the text selection changed in the specified text view. This method is optional.

Instance Methods

textView:shouldChangeTextInRange:replacementText:

Asks the delegate whether the specified text should be replaced in the text view. This method is optional.

```
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range
    replacementText:(NSString *)text
```

Parameters

textView
The text view containing the changes.

range
The current selection range. The length of the range is always 0, so this range reflects the current insertion point.

text
The text to insert.

Return Value

`YES` if the old text should be replaced by the new text; `NO` if the replacement operation should be aborted.

Discussion

The text view calls this method whenever the user types a new character or deletes an existing character. Implementation of this method is optional. You can use this method to replace text before it is committed to the text view storage. For example, a spell checker might use this method to replace a misspelled word with the correct spelling.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UITextView.h`

textViewDidBeginEditing:

Tells the delegate that editing of the specified text view has begun. This method is optional.

```
- (void)textViewDidBeginEditing:(UITextView *)textView
```

Parameters

textView

The text view in which editing began.

Discussion

Implementation of this method is optional. A text view sends this message to its delegate immediately after the user initiates editing in a text view and before any changes are actually made. You can use this method to set up any editing-related data structures and generally prepare your delegate to receive future editing messages.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITextView.h

textViewDidChange:

Tells the delegate that the text or attributes in the specified text view changed. This method is optional.

```
- (void)textViewDidChange:(UITextView *)textView
```

Parameters

textView

The text view containing the changes.

Discussion

Implementation of this method is optional.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITextView.h

textViewDidChangeSelection:

Tells the delegate that the text selection changed in the specified text view. This method is optional.

```
- (void)textViewDidChangeSelection:(UITextView *)textView
```

Parameters

textView

The text view whose selection changed.

Discussion

Implementation of this method is optional. You can use the `selectedRange` (page 426) property of the text view to get the new selection.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

textViewDidEndEditing:

Tells the delegate that editing of the specified text view has ended. This method is optional.

```
- (void)textViewDidEndEditing:(UITextView *)textView
```

Parameters
textView
The text view in which editing ended.

Discussion
Implementation of this method is optional. A text view sends this message to its delegate after it closes out any pending edits and resigns its first responder status. You can use this method to tear down any data structures or change any state information that you set when editing began.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

textViewShouldBeginEditing:

Asks the delegate if editing should begin in the specified text view. This method is optional.

```
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView
```

Parameters
textView
The text view for which editing is about to begin.

Return Value
YES if an editing session should be initiated; otherwise, NO to disallow editing.

Discussion
When the user performs an action that would normally initiate an editing session, the text view calls this method first to see if editing should actually proceed. In most circumstances, you would simply return YES from this method to allow editing to proceed.

Implementation of this method by the delegate is optional. If it is not present, editing proceeds as if this method had returned YES.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UITextView.h

textViewShouldEndEditing:

Asks the delegate if editing should stop in the specified text view.

```
- (BOOL)textViewShouldEndEditing:(UITextView *)textView
```

Parameters

textView

The text view for which editing is about to end.

Return Value

YES if editing should stop; otherwise, NO if the editing session should continue

Discussion

This method is called when the text view is asked to resign the first responder status. This might occur when the user tries to change the editing focus to another control. Before the focus actually changes, however, the text view calls this method to give your delegate a chance to decide whether it should.

Normally, you would return YES from this method to allow the text view to resign the first responder status. You might return NO, however, in cases where your delegate wants to validate the contents of the text view. By returning NO, you could prevent the user from switching to another control until the text view contained a valid value.

Be aware that this method provides only a recommendation about whether editing should end. Even if you return NO from this method, it is possible that editing might still end. For example, this might happen when the text view is forced to resign the first responder status by being removed from its parent view or window.

Implementation of this method by the delegate is optional. If it is not present, the first responder status is resigned as if this method had returned YES.

Availability

Available in iPhone OS 2.0 and later.

Declared In

UITextView.h

UIWebViewDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	UIWebView.h

Overview

The `UIWebViewDelegate` protocol defines methods that a delegate of a `UIWebView` object can optionally implement to intervene when web content is loaded or the scale changes.

Tasks

Loading Content

- `webView:shouldStartLoadWithRequest:navigationType:` (page 620) *optional method*
  Sent before a web view begins loading content. This method is optional.
- `webViewDidStartLoad:` (page 621) *optional method*
  Sent after a web view starts loading content. This method is optional.
- `webViewDidFinishLoad:` (page 621) *optional method*
  Sent after a web view finishes loading content. This method is optional.
- `webView:didFailLoadWithError:` (page 620) *optional method*
  Sent if a web view failed to load content. This method is optional.

Instance Methods

webView:didFailLoadWithError:

Sent if a web view failed to load content. This method is optional.

```
- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error
```

Parameters

webView
The web view that failed to load content.

error
The error that occurred during loading.

Availability
Available in iPhone OS 2.0 and later.

See Also
- webView:shouldStartLoadWithRequest: (page 620)
- webViewDidStartLoad: (page 621)
- webViewDidFinishLoad: (page 621)

Declared In
UIWebView.h

webView:shouldStartLoadWithRequest:navigationType:

Sent before a web view begins loading content. This method is optional.

```
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest
    *)request navigationType:(UIWebViewNavigationType)navigationType
```

Parameters

webView
The web view that is about to load content.

request
The content location.

navigationType
The type of user action that started the load request.

Return Value
YES if the web view should begin loading content; otherwise, NO .

Availability
Available in iPhone OS 2.0 and later.

See Also
- webViewDidStartLoad: (page 621)
- webViewDidFinishLoad: (page 621)
- webView:didFailLoadWithError: (page 620)

Declared In
UIWebView.h

webViewDidFinishLoad:

Sent after a web view finishes loading content. This method is optional.

```
- (void)webViewDidFinishLoad:(UIWebView *)webView
```

Parameters

webView
The web view has finished loading.

Availability
Available in iPhone OS 2.0 and later.

See Also
- webView:shouldStartLoadWithRequest: (page 620)
- webViewDidStartLoad: (page 621)
- webView:didFailLoadWithError: (page 620)

Declared In
UIWebView.h

webViewDidStartLoad:

Sent after a web view starts loading content. This method is optional.

```
- (void)webViewDidStartLoad:(UIWebView *)webView
```

Parameters

webView
The web view that has begun loading content.

Availability
Available in iPhone OS 2.0 and later.

See Also
- webView:shouldStartLoadWithRequest: (page 620)
- webViewDidFinishLoad: (page 621)
- webView:didFailLoadWithError: (page 620)

Declared In
UIWebView.h

Data Types

PART III
Data Types

UIKit Data Types Reference

Framework: UIKit/UIKit.h
Declared in: UIGeometry.h

Overview

The UIKit framework defines data types that are used in multiple places throughout the framework.

Data Types

UIEdgeInsets

Defines inset distances for views.

```
typedef struct {
    CGFloat top, left, bottom, right;
} UIEdgeInsets;
```

Discussion

Edge inset values are applied to a rectangle to shrink or expand the area represented by that rectangle. Typically, edge insets are used during view layout to modify the view's frame. Positive values cause the frame to be inset (or shrunk) by the specified amount. Negative values cause the frame to be outset (or expanded) by the specified amount.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIGeometry.h`

UIBarStyle

Defines the stylistic appearance of different types of views.

```
typedef enum {
    UIBarStyleDefault          = 0,
    UIBarStyleBlackOpaque      = 1,
    UIBarStyleBlackTranslucent = 2,
} UIBarStyle;
```

Constants

`UIBarStyleDefault`

Use the default style normally associated with the given view. For example, search bars and tool bars typically use a blue gradient background.

Available in iPhone OS 2.0 and later.

Declared in `UIInterface.h`

`UIBarStyleBlackOpaque`

Use an opaque black style.

Available in iPhone OS 2.0 and later.

Declared in `UIInterface.h`

`UIBarStyleBlackTranslucent`

Use a partially transparent black style.

Available in iPhone OS 2.0 and later.

Declared in `UIInterface.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIInterface.h`

Constants

UIKit Constants Reference

Framework: UIKit/UIKit.h
Declared in: UIGeometry.h

Overview

This document describes constants that are used throughout the UIKit framework.

Constants

UIEdgeInsetsZero

Defines a set of edge insets where all of the values are 0.

```
extern const UIEdgeInsets UIEdgeInsetsZero;
```

Constants

`UIEdgeInsetsZero`

A `UIEdgeInsets` struct whose `top`, `left`, `bottom`, and `right` fields are all set to the value 0.

Available in iPhone OS 2.0 and later.

Declared in `UIGeometry.h`

Other References

PART V

Other References

UIKit Function Reference

Framework: UIKit/UIKit.h

Declared in: UIGraphics.h
UIGeometry.h
UIApplication.h

Overview

The UIKit framework defines a number of functions, many of them used in graphics and drawing operations.

Functions by Task

Application Launch

`UIApplicationMain` (page 640)
This function is called in the `main` entry point to create the application object and the application delegate and set up the event cycle.

Image Manipulation

`UIImageJPEGRepresentation` (page 646)
Returns the data for the specified image in JPEG format.

`UIImagePNGRepresentation` (page 646)
Returns the data for the specified image in PNG format

`UIImageWriteToSavedPhotosAlbum` (page 647)
Adds the specified image to the user's Saved Photos album.

Graphics

UIGraphicsGetCurrentContext (page 644)
Returns the current graphics context.

UIGraphicsPushContext (page 645)
Makes the specified graphics context the current context.

UIGraphicsPopContext (page 645)
Removes the current graphics context from the top of the stack, restoring the previous context.

UIGraphicsBeginImageContext (page 643)
Creates a bitmap-based graphics context and makes it the current context.

UIGraphicsGetImageFromCurrentImageContext (page 644)
Returns an image based on the contents of the current bitmap-based graphics context.

UIGraphicsEndImageContext (page 643)
Removes the current bitmap-based graphics context from the top of the stack.

UIRectClip (page 647)
Modifies the current clipping path by intersecting it with the specified rectangle.

UIRectFill (page 648)
Fills the specified rectangle with the current color.

UIRectFillUsingBlendMode (page 649)
Fills a rectangle with the current fill color using the specified blend mode.

UIRectFrame (page 649)
Draws a frame around the inside of the specified rectangle.

UIRectFrameUsingBlendMode (page 650)
Draws a frame around the inside of a rectangle using the specified blend mode.

String Conversions

CGPointFromString (page 636)
Returns a Core Graphics point structure corresponding to the data in a given string.

CGRectFromString (page 636)
Returns a Core Graphics rectangle structure corresponding to the data in a given string.

CGSizeFromString (page 637)
Returns a Core Graphics size structure corresponding to the data in a given string.

CGAffineTransformFromString (page 635)
Returns a Core Graphics affine transform structure corresponding to the data in a given string.

UIEdgeInsetsFromString (page 641)
Returns a UIKit edge insets structure corresponding to the data in a given string.

NSStringFromCGPoint (page 638)
Returns a string object formatted to contain the data from a point.

NSStringFromCGRect (page 638)
Returns a string object formatted to contain the data from a rectangle.

NSStringFromCGSize (page 639)
Returns a string object formatted to contain the data from a size data structure.

NSStringFromCGAffineTransform (page 637)
Returns a string object formatted to contain the data from an affine transform.

NSStringFromUIEdgeInsets (page 639)
Returns a string object formatted to contain the data from an edge insets structure.

Setting Edge Insets

UIEdgeInsetsMake (page 642)
Creates an edge inset for a button or view.

UIEdgeInsetsEqualToEdgeInsets (page 640)
Compares two edge insets to determine if they are the same.

UIEdgeInsetsInsetRect (page 641)
Adjusts a rectangle by the given edge insets.

Functions

CGAffineTransformFromString

Returns a Core Graphics affine transform structure corresponding to the data in a given string.

```
CGAffineTransform CGAffineTransformFromString (
   NSString *string
);
```

Parameters

string

A string object whose contents are of the form "{*a*, *b*, *c*, *d*, *tx*, *ty*}", where *a*, *b*, *c*, *d*, *tx*, and *ty* are the floating-point component values of the CGAffineTransform data structure. An example of a valid string is @"{1,0,0,1,2.5,3.0}". The string is not localized, so items are always separate with a comma. For information about the position of each value in the transform array, see *CGAffineTransform Reference*.

Return Value

A Core Graphics affine transform structure. If the string is not well-formed, the function returns the identity transform.

Discussion

In general, you should use this function only to convert strings that were previously created using the NSStringFromCGAffineTransform function.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSStringFromCGAffineTransform (page 637)

Declared In

UIGeometry.h

CGPointFromString

Returns a Core Graphics point structure corresponding to the data in a given string.

```
CGPoint CGPointFromString (
   NSString *string
);
```

Parameters

string

A string object whose contents are of the form "{*x*,*y*}", where *x* is the x coordinate and *y* is the y coordinate. The *x* and *y* values can represent integer or float values. An example of a valid string is @"{3.0,2.5}". The string is not localized, so items are always separate with a comma.

Return Value

A Core Graphics structure that represents a point. If the string is not well-formed, the function returns `CGPointZero`.

Discussion

In general, you should use this function only to convert strings that were previously created using the `NSStringFromCGPoint` function.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSStringFromCGPoint` (page 638)

Declared In

`UIGeometry.h`

CGRectFromString

Returns a Core Graphics rectangle structure corresponding to the data in a given string.

```
CGRect CGRectFromString (
   NSString *string
);
```

Parameters

string

A string object whose contents are of the form "{{*x*,*y*},{*h*,*w*}}", where *x* is the x coordinate, *y* is the y coordinate, *h* is the height, and *w* is the width. These components can represent integer or float values. An example of a valid string is @"{{3,2},{4,5}}". The string is not localized, so items are always separate with a comma.

Return Value

A Core Graphics structure that represents a rectangle. If the string is not well-formed, the function returns `CGRectZero`.

Discussion

In general, you should use this function only to convert strings that were previously created using the `NSStringFromCGRect` function.

Availability

Available in iPhone OS 2.0 and later.

See Also
NSStringFromCGRect (page 638)

Declared In
UIGeometry.h

CGSizeFromString

Returns a Core Graphics size structure corresponding to the data in a given string.

```
CGSize CGSizeFromString (
   NSString *string
);
```

Parameters
string

A string object whose contents are of the form "{*h*,*w*}", where *h* is the height and *w* is the width. The *w* and *w* values can be integer or float values. An example of a valid string is @"{3.0,2.5}". The string is not localized, so items are always separate with a comma.

Return Value
A Core Graphics structure that represents a size. If the string is not well-formed, the function returns CGSizeZero.

Discussion
In general, you should use this function only to convert strings that were previously created using the NSStringFromCGSize function.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSStringFromCGSize (page 639)

Declared In
UIGeometry.h

NSStringFromCGAffineTransform

Returns a string object formatted to contain the data from an affine transform.

```
NSString *NSStringFromCGAffineTransform(CGAffineTransform transform);
```

Parameters
transform

A Core Graphics affine transform structure.

Return Value
A string object that corresponds to *transform*. See CGAffineTransformFromString (page 635) for a discussion of the string format.

Availability
Available in iPhone OS 2.0 and later.

See Also
CGAffineTransformFromString (page 635)

Declared In
UIGeometry.h

NSStringFromCGPoint

Returns a string object formatted to contain the data from a point.

```
NSString * NSStringFromCGPoint (
   CGPoint point
);
```

Parameters

point
A Core Graphics structure representing a point.

Return Value
A string object that corresponds to *point*. See CGPointFromString (page 636) for a discussion of the string format.

Availability
Available in iPhone OS 2.0 and later.

See Also
CGPointFromString (page 636)

Declared In
UIGeometry.h

NSStringFromCGRect

Returns a string object formatted to contain the data from a rectangle.

```
NSString * NSStringFromCGRect (
   CGRect rect
);
```

Parameters

rect
A Core Graphics structure representing a rectangle.

Return Value
A string object that corresponds to *rect*. See CGRectFromString (page 636) for a discussion of the string format.

Availability
Available in iPhone OS 2.0 and later.

See Also
CGRectFromString (page 636)

Declared In
UIGeometry.h

NSStringFromCGSize

Returns a string object formatted to contain the data from a size data structure.

```
NSString * NSStringFromCGSize (
   CGSize size
);
```

Parameters

size

A Core Graphics structure representing a size.

Return Value
A string object that corresponds to *size*. See CGSizeFromString (page 637) for a discussion of the string format.

Availability
Available in iPhone OS 2.0 and later.

See Also
CGSizeFromString (page 637)

Declared In
UIGeometry.h

NSStringFromUIEdgeInsets

Returns a string object formatted to contain the data from an edge insets structure.

```
NSString *NSStringFromUIEdgeInsets(UIEdgeInsets insets);
```

Parameters

insets

A UIKit edge insets data structure.

Return Value
A string object that corresponds to *insets*. See UIEdgeInsetsFromString (page 641) for a discussion of the string format.

Availability
Available in iPhone OS 2.0 and later.

See Also
UIEdgeInsetsFromString (page 641)

Declared In
UIGeometry.h

UIApplicationMain

This function is called in the `main` entry point to create the application object and the application delegate and set up the event cycle.

```
int UIApplicationMain (
   int argc,
   char *argv[],
   NSString *principalClassName,
   NSString *delegateClassName
);
```

Parameters

argc

The count of arguments in *argv*; this usually is the corresponding parameter to `main`.

argv

A variable list of arguments; this usually is the corresponding parameter to `main`.

principalClassName

The name of the `UIApplication` class or subclass. If you specify `nil`, `UIApplication` is assumed.

delegateClassName

The name of the class from which the application delegate is instantiated. If *principalClassName* designates a subclass of `UIApplication`, you may designate the subclass as the delegate; the subclass instance receives the application-delegate messages.

Return Value

The value 0. This value is always returned when the function exits successfully. If there are internal problems, the application code calls the `exit` system function with an appropriate error code, thus killing the application immediately without returning from this function.

Discussion

This function instantiates the application object from the principal class and and instantiates the delegate (if any) from the given class and sets the delegate for the application. It also sets up the main event loop, including the application's run loop, and begins processing events. This function returns only on termination.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIApplication.h`

UIEdgeInsetsEqualToEdgeInsets

Compares two edge insets to determine if they are the same.

```
BOOL UIEdgeInsetsEqualToEdgeInsets (
   UIEdgeInsets insets1,
   UIEdgeInsets insets2
);
```

Parameters

insets1

An edge inset to compare with *insets2*.

insets2
An edge inset to compare with *insets1*.

Return Value
YES if the edge insets are the same; otherwise, NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
UIEdgeInsetsMake (page 642)

Declared In
UIGeometry.h

UIEdgeInsetsFromString

Returns a UIKit edge insets structure corresponding to the data in a given string.

```
UIEdgeInsets UIEdgeInsetsFromString (
   NSString *string
);
```

Parameters
string
A string object whose contents are of the form "{*top*, *left*, *bottom*, *right*}", where *top*, *left*, *bottom*, *right* are the floating-point component values of the UIEdgeInsets structure. An example of a valid string is @"{3.0,8.0,3.0,5.0}". The string is not localized, so items are always separate with a comma.

Return Value
An edge insets data structure. If the string is not well-formed, the function returns UIEdgeInsetsZero.

Discussion
In general, you should use this function only to convert strings that were previously created using the NSStringFromUIEdgeInsets function.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSStringFromUIEdgeInsets (page 639)

Declared In
UIGeometry.h

UIEdgeInsetsInsetRect

Adjusts a rectangle by the given edge insets.

```
CGRect UIEdgeInsetsInsetRect (
   CGRect rect,
   UIEdgeInsets insets
);
```

Parameters

rect

The rectangle to be adjusted.

insets

The edge insets to be applied to the adjustment.

Return Value

A rectangle that is adjusted by the `UIEdgeInsets` structure passed in insets. i

Discussion

This inline function increments the origin of *rect* and decrements the size of *rect* by applying the appropriate member values of the `UIEdgeInsets` structure.

Availability

Available in iPhone OS 2.0 and later.

See Also

`UIEdgeInsetsMake` (page 642)

Declared In

`UIGeometry.h`

UIEdgeInsetsMake

Creates an edge inset for a button or view.

```
UIEdgeInsets UIEdgeInsetsMake (
   CGFloat top,
   CGFloat left,
   CGFloat bottom,
   CGFloat right
);
```

Parameters

top

The inset at the top of an object.

left

The inset on the left of an object

bottom

The inset on the bottom of an object.

right

The inset on the right of an object.

Return Value

An inset for a button or view

Discussion

An inset is a margin around the drawing rectangle where each side (left, right, top, and bottom) can have a different value.

Availability
Available in iPhone OS 2.0 and later.

See Also
UIEdgeInsetsEqualToEdgeInsets (page 640)

Declared In
UIGeometry.h

UIGraphicsBeginImageContext

Creates a bitmap-based graphics context and makes it the current context.

```
void UIGraphicsBeginImageContext(CGSize size);
```

Parameters

size

The size of the new bitmap context. This represents the size of the image returned by the UIGraphicsGetImageFromCurrentImageContext function.

Discussion
You use this function to configure the drawing environment for rendering into a bitmap. This drawing environment uses the premultiplied RGBA format to store the resulting bitmap data. The environment also uses the default coordinate system for UIKit views, where the origin is in the upper-left corner and the positive axes extend down and to the right of the origin. The drawing environment is pushed onto the graphics context stack immediately.

While the context created by this function is the current context, you can call the UIGraphicsGetImageFromCurrentImageContext function to retrieve an image object based on the current contents of the context. When you are done modifying the context, you must call the UIGraphicsEndImageContext function to clean up the bitmap drawing environment and remove the graphics context from the top of the context stack. You should not use the UIGraphicsPopContext function to remove this type of context from the stack.

In most other respects, the graphics context created by this function behaves like any other graphics context. You can change the context by pushing and popping other graphics contexts. You can also get the bitmap context using the UIGraphicsGetCurrentContext function.

Availability
Available in iPhone OS 2.0 and later.

See Also
UIGraphicsGetCurrentContext (page 644)
UIGraphicsGetImageFromCurrentImageContext (page 644)
UIGraphicsEndImageContext (page 643)

Declared In
UIGraphics.h

UIGraphicsEndImageContext

Removes the current bitmap-based graphics context from the top of the stack.

```
void UIGraphicsEndImageContext(void);
```

Discussion
You use this function to clean up the drawing environment put in place by the `UIGraphicsBeginImageContext` function and to remove the corresponding bitmap-based graphics context from the top of the stack. If the current context was not created using the `UIGraphicsBeginImageContext` function, this method does nothing.

Availability
Available in iPhone OS 2.0 and later.

See Also
`UIGraphicsBeginImageContext` (page 643)

Declared In
`UIGraphics.h`

UIGraphicsGetCurrentContext

Returns the current graphics context.

```
CGContextRef UIGraphicsGetCurrentContext (
   void
);
```

Return Value
The current graphics context.

Discussion
The current graphics context is `nil` by default. Prior to calling its `drawRect:` method, view objects push a valid context onto the stack, making it current. If you are not using a `UIView` object to do your drawing, however, you must push a valid context onto the stack manually using the `UIGraphicsPushContext` (page 645) function.

Availability
Available in iPhone OS 2.0 and later.

See Also
`UIGraphicsPushContext` (page 645)
`UIGraphicsPopContext` (page 645)

Declared In
`UIGraphics.h`

UIGraphicsGetImageFromCurrentImageContext

Returns an image based on the contents of the current bitmap-based graphics context.

```
UIImage* UIGraphicsGetImageFromCurrentImageContext(void);
```

Return Value
An autoreleased image object containing the contents of the current bitmap graphics context.

Discussion
You should call this function only when a bitmap-based graphics context is the current graphics context. If the current context is `nil` or was not created by a call to `UIGraphicsBeginImageContext`, this function returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`UIGraphicsBeginImageContext` (page 643)

Declared In
`UIGraphics.h`

UIGraphicsPopContext
Removes the current graphics context from the top of the stack, restoring the previous context.

```
void UIGraphicsPopContext (
   void
);
```

Discussion
Use this function to balance calls to the `UIGraphicsPushContext` (page 645) function.

Availability
Available in iPhone OS 2.0 and later.

See Also
`UIGraphicsPushContext` (page 645)

Declared In
`UIGraphics.h`

UIGraphicsPushContext
Makes the specified graphics context the current context.

```
void UIGraphicsPushContext (
   CGContextRef context
);
```

Parameters
context
The graphics context to make the current context.

Discussion
You can use this function to save the previous graphics state and make the specified context the current context. You must balance calls to this function with matching calls to the `UIGraphicsPopContext` (page 645) function.

Availability
Available in iPhone OS 2.0 and later.

See Also
UIGraphicsPopContext (page 645)

Declared In
UIGraphics.h

UIImageJPEGRepresentation

Returns the data for the specified image in JPEG format.

```
NSData * UIImageJPEGRepresentation (
   UIImage *image,
   CGFloat compressionQuality
);
```

Parameters

image
The original image data.

compressionQuality
The quality of the resulting JPEG image, expressed as a value from 0.0 to 1.0. The value 0.0 represents the maximum compression (or lowest quality) while the value 1.0 represents the least compression (or best quality).

Return Value
A data object containing the JPEG data, or nil if there was a problem generating the data. This function may return nil if the image has no data or if the underlying CGImageRef contains data in an unsupported bitmap format.

Discussion
If the image object's underlying image data has been purged, calling this method forces that data to be reloaded into memory.

Availability
Available in iPhone OS 2.0 and later.

Declared In
UIImage.h

UIImagePNGRepresentation

Returns the data for the specified image in PNG format

```
NSData * UIImagePNGRepresentation (
   UIImage *image
);
```

Parameters

image
The original image data.

Return Value
A data object containing the PNG data, or nil if there was a problem generating the data. This function may return nil if the image has no data or if the underlying CGImageRef contains data in an unsupported bitmap format.

Discussion
If the image object's underlying image data has been purged, calling this method forces that data to be reloaded into memory.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImage.h`

UIImageWriteToSavedPhotosAlbum

Adds the specified image to the user's Saved Photos album.

```
void UIImageWriteToSavedPhotosAlbum(UIImage *image, id completionTarget, SEL
completionSelector, void *contextInfo);
```

Parameters
`image`
The image to write to the user's device.

`completionTarget`
The object whose selector should be called after the image has been written to the user's device.

`comletionSelector`
The selector of the target object to call. This method should be of the form:

```
- (void)image:(UIImage *)image
        didFinishSavingWithError:(NSError *)error
        contextInfo:(void *)contextInfo;
```

`contextInfo`
An optional pointer to any context-specific data that you want passed to the completion selector.

Discussion
The use of the *`completionTarget`*, *`completionSelector`*, and *`contextInfo`* parameters is optional and necessary only if you want to be notified asynchronously when the function finishes writing the image to the user's Saved Photos album. If you do not want to be notified, pass `nil` for these parameters.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`UIImagePickerController.h`

UIRectClip

Modifies the current clipping path by intersecting it with the specified rectangle.

```
void UIRectClip (
    CGRect rect
);
```

Parameters

rect

The rectangle to intersect with the clipping region. If the width or height of the rectangle are less than 0, this function does not change the clipping path.

Discussion

Each call to this function permanently shrinks the clipping path of the current graphics context using the specified rectangle. You cannot use this function to expand the clipping region path. If the current graphics context is `nil`, this method does nothing.

If you need to return the clipping path to its original shape in your drawing code, you should save the current graphics context before calling this function. To save the current context, push a new graphics context onto the top of the stack using the `UIGraphicsPushContext` (page 645) function. When you are ready to restore the original clipping region, you can then use the `UIGraphicsPopContext` (page 645) function to remove the current context and restore the previous graphics state.

Availability

Available in iPhone OS 2.0 and later.

See Also

`UIGraphicsPushContext` (page 645)

`UIGraphicsPopContext` (page 645)

Declared In

`UIGraphics.h`

UIRectFill

Fills the specified rectangle with the current color.

```
void UIRectFill (
   CGRect rect
);
```

Parameters

rect

The rectangle defining the area in which to draw.

Discussion

Fills the specified rectangle using the fill color of the current graphics context and the `kCGBlendModeNormal` blend mode.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIGraphics.h`

UIRectFillUsingBlendMode

Fills a rectangle with the current fill color using the specified blend mode.

```
void UIRectFillUsingBlendMode (
   CGRect rect,
   CGBlendMode blendMode
);
```

Parameters

rect

The rectangle defining the area in which to draw.

blendMode

The blend mode to use during drawing.

Discussion

This function draws the rectangle in the current graphics context. If the current graphics context is `nil`, this method does nothing.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIGraphics.h`

UIRectFrame

Draws a frame around the inside of the specified rectangle.

```
void UIRectFrame (
   CGRect rect
);
```

Parameters

rect

The rectangle defining the area in which to draw.

Discussion

This function draws a frame around the inside of *rect* in the fill color of the current graphics context and using the `kCGBlendModeNormal` blend mode. The width is equal to 1.0 in the current coordinate system. Since the frame is drawn inside the rectangle, it is visible even if drawing is clipped to the rectangle. If the current graphics context is `nil`, this method does nothing.

Because this function does not draw directly on the line, but rather inside it, it uses the current fill color (not stroke color) when drawing.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIGraphics.h`

UIRectFrameUsingBlendMode

Draws a frame around the inside of a rectangle using the specified blend mode.

```
void UIRectFrameUsingBlendMode (
   CGRect rect,
   CGBlendMode blendMode
);
```

Parameters

rect

The rectangle defining the area in which to draw.

blendMode

The blend mode to use during drawing.

Discussion

This function draws a frame around the inside of rect in the fill color of the current graphics context and using the specified blend mode. The width is equal to 1.0 in the current coordinate system. Since the frame is drawn inside the rectangle, it is visible even if drawing is clipped to the rectangle. If the current graphics context is `nil`, this method does nothing.

Because this function does not draw directly on the line, but rather inside it, it uses the current fill color (not stroke color) when drawing.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`UIGraphics.h`

Document Revision History

This table describes the changes to *UIKit Framework Reference*.

Date	Notes
2008-05-18	New document that describes the programming interface for constructing and managing the user interface of iPhone applications.

Document Revision History

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z