UIKit Framework Reference

2010-04-22

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

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

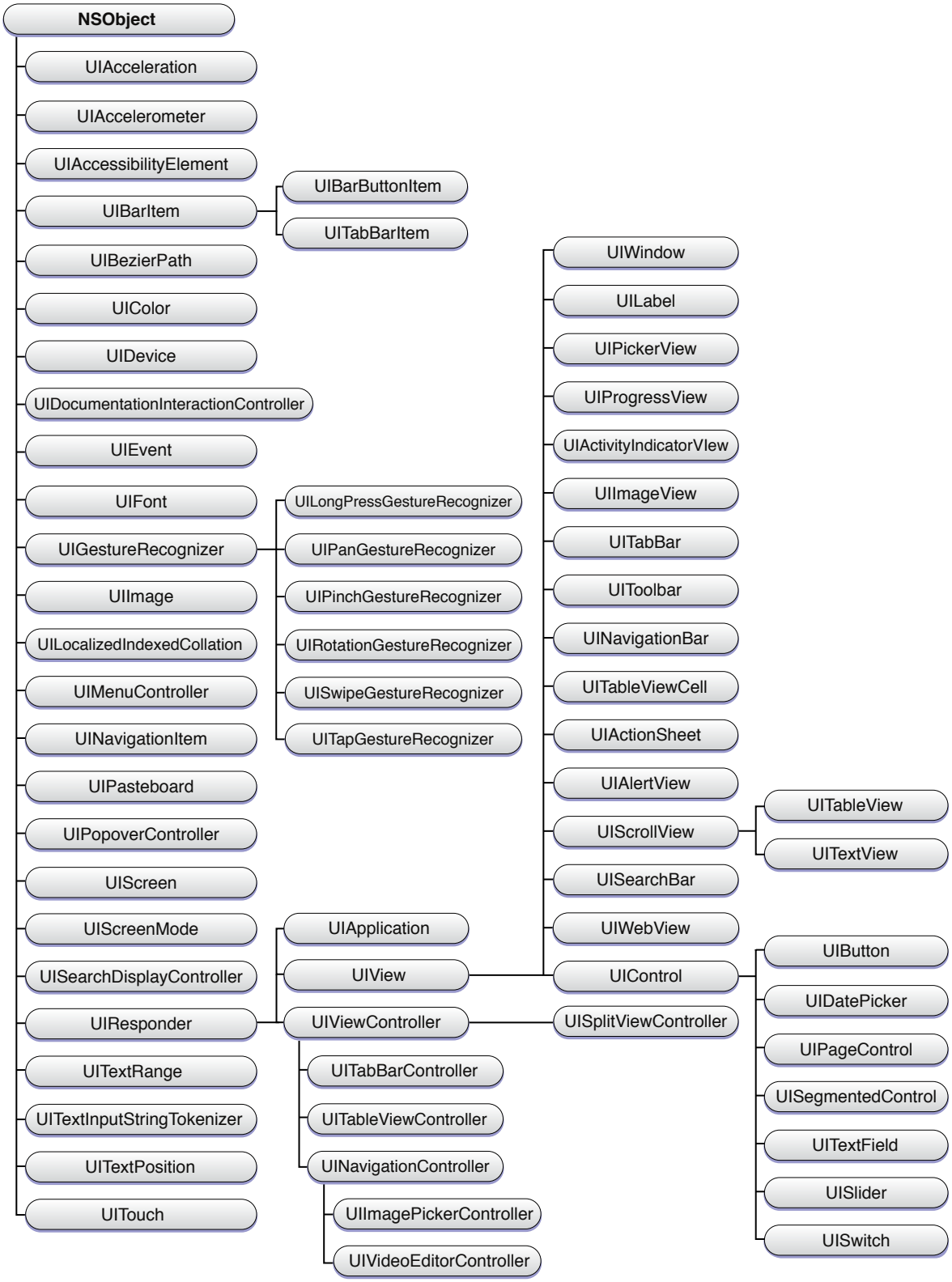
Introduction

Framework	/System/Library/Frameworks/UIKit.framework
Header file directories	/System/Library/Frameworks/UIKit.framework/Headers
Declared in	UIAccelerometer.h UIAccessibility.h UIAccessibilityAdditions.h UIAccessibilityConstants.h UIAccessibilityElement.h UIActionSheet.h UIActivityIndicatorView.h UIAlertView.h UIApplication.h UIBarButtonItem.h UIBarItem.h UIBezierPath.h UIButton.h UIColor.h UIControl.h UIDataDetectors.h UIDatePicker.h UIDevice.h UIDocumentInteractionController.h UIEvent.h UIFont.h UIGeometry.h UIGestureRecognizer.h UIGestureRecognizerSubclass.h UIGraphics.h UIImage.h UIImagePickerController.h UIImageView.h UIInterface.h UILabel.h UILocalizedIndexedCollation.h UILongPressGestureRecognizer.h UIMenuController.h UINavigationBar.h UINavigationController.h UINib.h UINibDeclarations.h UINibLoading.h UIPageControl.h UIPanGestureRecognizer.h UIPasteboard.h UIPickerView.h UIPinchGestureRecognizer.h

UIPopoverController.h
UIProgressView.h
UIResponder.h
UIRotationGestureRecognizer.h
UIScreen.h
UIScreenMode.h
UIScrollView.h
UISearchBar.h
UISearchDisplayController.h
UISegmentedControl.h
UISlider.h
UISplitViewController.h
UIStringDrawing.h
UISwipeGestureRecognizer.h
UISwitch.h
UITabBar.h
UITabBarController.h
UITabBarItem.h
UITableView.h
UITableViewCell.h
UITableViewController.h
UITapGestureRecognizer.h
UITextField.h
UITextInput.h
UITextInputTraits.h
UITextView.h
UIToolbar.h
UITouch.h
UIVideoEditorController.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 iOS. It provides an application object, event handling, drawing model, windows, views, and controls specifically designed for a touch screen interface. Figure I-1 (page 25) illustrates the classes in this framework.

Figure I-1 UIKit class hierarchy



Note: For the most part, UIKit classes should be used only from an application's main thread. This is particularly true for classes derived from `UIResponder` or that involve manipulating your application's user interface in any way.

Classes

NSBundle UIKit Additions Reference

Inherits from	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 29)
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 30).

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 `UINibExternalObjects` 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 Identifier field of the Interface Builder inspector window.

Availability
Available in iOS 2.0 and later.

Related Sample Code
KeyboardAccessory

Declared In
`UINibLoading.h`

Constants

Nib File Loading Options

The options that can be specified during nib loading.

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

Constants
`UINibProxiedObjectsKey`

In iOS 2.x, 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. (Deprecated. Use the `UINibExternalObjects` (page 31) key instead.)

Available in iOS 2.0 and later.

Deprecated in iOS 3.0.

Declared in `UINibLoading.h`.

`UINibExternalObjects`

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 iOS 3.0 and later.

Declared in `UINibLoading.h`.

NSCoder UIKit Additions Reference

Inherits from	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 37)
  Encodes a point and associates it with the specified key in the receiver's archive.
- `encodeCGRect:forKey:` (page 37)
  Encodes a rectangle and associates it with the specified key in the receiver's archive.
- `encodeCGSize:forKey:` (page 38)
  Encodes size information and associates it with the specified key in the receiver's archive.
- `encodeCGAffineTransform:forKey:` (page 36)
  Encodes an affine transform and associates it with the specified key in the receiver's archive.
- `encodeUIEdgeInsets:forKey:` (page 38)
  Encodes edge inset data and associates it with the specified key in the receiver's archive.

Decoding Data

- `decodeCGPointForKey:` (page 34)
  Decodes and returns the `CGPoint` structure associated with the specified key in the receiver's archive.
- `decodeCGRectForKey:` (page 35)
  Decodes and returns the `CGRect` structure associated with the specified key in the receiver's archive.

- `decodeCGSizeForKey:` (page 35)
  Decodes and returns the `CGSize` structure associated with the specified key in the receiver's archive.
- `decodeCGAffineTransformForKey:` (page 34)
  Decodes and returns the `CGAffineTransform` structure associated with the specified key in the receiver's archive.
- `decodeUIEdgeInsetsForKey:` (page 36)
  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 iOS 2.0 and later.

See Also
- `encodeCGAffineTransform:forKey:` (page 36)

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 iOS 2.0 and later.

See Also
`- encodeCGPoint:forKey:` (page 37)

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 iOS 2.0 and later.

See Also
`- encodeCGRect:forKey:` (page 37)

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 iOS 2.0 and later.

See Also
`- encodeCGSize:forKey:` (page 38)

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 iOS 2.0 and later.

See Also
`- encodeUIEdgeInsets:forKey:` (page 38)

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 iOS 2.0 and later.

See Also
`- decodeCGAffineTransformForKey:` (page 34)

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 iOS 2.0 and later.

See Also
`- decodeCGPointForKey:` (page 34)

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 iOS 2.0 and later.

See Also
- `decodeCGRectForKey:` (page 35)

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 iOS 2.0 and later.

See Also
- `decodeCGSizeForKey:` (page 35)

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 iOS 2.0 and later.

See Also
- `decodeUIEdgeInsetsForKey:` (page 36)

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 iOS 2.0 and later.
Declared in	UITableView.h
Companion guide	Collections Programming Topics

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 42) 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 42)
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 42) *property*
An index number identifying a row in a section of a table view. (read-only)

`section` (page 42) *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 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 42).

Availability
Available in iOS 2.0 and later.

Related Sample Code
AddMusic
BonjourWeb
GKRocket
MultipleDetailViews
WiTap

Declared In
`UITableView.h`

section

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

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

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb

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 iOS 2.0 and later.

Related Sample Code

BonjourWeb

WiTap

Declared In

`UITableView.h`

NSObject UIKit Additions Reference

Inherits from	none
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 45)
  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 iOS 2.0 and later.

See Also
`awakeAfterUsingCoder:` (NSObject class)
`initWithCoder:` (NSCoding protocol)
`initialize` (NSObject class)

Related Sample Code
SpeakHere

Declared In
`UINibLoading.h`

NSString UIKit Additions Reference

Inherits from	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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. None of these methods affects the contents of the string object itself, only how it is drawn on screen.

By default, strings are drawn using the native coordinate system of iOS, 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

Computing Metrics for a Single Line of Text

- `sizeWithFont:` (page 53)
  Returns the size of the string if it were to be rendered with the specified font on a single line.
- `sizeWithFont:forWidth:lineBreakMode:` (page 55)
  Returns the size of the string if it were to be rendered with the specified font and line attributes on a single line.
- `sizeWithFont:minFontSize:actualFontSize:forWidth:lineBreakMode:` (page 55)
  Returns the size of the string if it were rendered with the specified constraints, including a variable font size, on a single line.

Computing Metrics for Multiple Lines of Text

- `sizeWithFont:constrainedToSize:` (page 53)
  Returns the size of the string if it were rendered and constrained to the specified size.

- `sizeWithFont:constrainedToSize:lineBreakMode:` (page 54)
  Returns the size of the string if it were rendered with the specified constraints.

Drawing Strings on a Single Line

- `drawAtPoint:withFont:` (page 51)
  Draws the string in a single line at the specified point in the current graphics context using the specified font.
- `drawAtPoint:forWidth:withFont:lineBreakMode:` (page 49)
  Draws the string in a single line at the specified point in the current graphics context using the specified font and attributes.
- `drawAtPoint:forWidth:withFont:fontSize:lineBreakMode:baselineAdjustment:` (page 48)
  Draws the string in a single line at the specified point in the current graphics context using the specified font and attributes.
- `drawAtPoint:forWidth:withFont:minFontSize:actualFontSize:lineBreakMode:baselineAdjustment:` (page 50)
  Draws the string in a single line with the specified font and attributes, adjusting the font attributes as needed to render as much of the text as possible.

Drawing Strings in a Given Area

- `drawInRect:withFont:` (page 51)
  Draws the string in the current graphics context using the specified bounding rectangle and font.
- `drawInRect:withFont:lineBreakMode:` (page 52)
  Draws the string in the current graphics context using the specified bounding rectangle, font, and attributes.
- `drawInRect:withFont:lineBreakMode:alignment:` (page 52)
  Draws the string in the current graphics context using the specified bounding rectangle, font and attributes.

Instance Methods

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

Draws the string in a single line at the specified point in the current graphics 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 56).

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 iOS 2.0 and later.

See Also
- sizeWithFont:minFontSize:actualFontSize:forWidth:lineBreakMode: (page 55)

Declared In
UIStringDrawing.h

drawAtPoint:forWidth:withFont:lineBreakMode:

Draws the string in a single line at the specified point in the current graphics 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 56).

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 iOS 2.0 and later.

Declared In
UIStringDrawing.h

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

Draws the string in a single line 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 56).

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 iOS 2.0 and later.

Declared In
UIStringDrawing.h

drawAtPoint:withFont:

Draws the string in a single line at the specified point in the current graphics 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 iOS 2.0 and later.

Declared In
UIStringDrawing.h

drawInRect:withFont:

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

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

Parameters

rect

The bounding rectangle (in the current graphics 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 as much of the string as possible using the given font and constraints. This method uses the `UILineBreakModeWordWrap` (page 56) line break mode and the `UITextAlignmentLeft` (page 57) alignment.

Availability
Available in iOS 2.0 and later.

Declared In
UIStringDrawing.h

drawInRect:withFont:lineBreakMode:

Draws the string in the current graphics 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 graphics 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 56).

Return Value
The actual size of the rendered string.

Discussion
This method draws as much of the string as possible using the given font, line break mode, and size constraints. The text is drawn using the `UITextAlignmentLeft` (page 57) alignment.

Availability
Available in iOS 2.0 and later.

Declared In
UIStringDrawing.h

drawInRect:withFont:lineBreakMode:alignment:

Draws the string in the current graphics 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 graphics 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 56).

alignment
The alignment of the text inside the bounding rectangle. For a list of possible values, see "UITextAlignment" (page 57).

Return Value
The actual size of the rendered string.

Availability
Available in iOS 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 on a single line.

```
- (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
You can use this method to obtain the layout metrics you need to draw a string in your user interface. This method does not actually draw the string or alter the receiver's text in any way.

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

Availability
Available in iOS 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
You can use this method to obtain the layout metrics you need to draw a string in your user interface. This method does not actually draw the string or alter the receiver's text in any way.

This method computes the metrics needed to draw the specified string. This method lays out the receiver's text and attempts to make it fit the specified size using the specified font the `UILineBreakModeWordWrap` (page 56) line break option. During layout, the method may break the text onto multiple lines to make it fit better. If the receiver's text does not completely fit in the specified size, it lays out as much of the text as possible and truncates it (for layout purposes only) according to the specified line break mode. It then returns the size of the resulting truncated string. 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.

Availability
Available in iOS 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 56).

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

Discussion
You can use this method to obtain the layout metrics you need to draw a string in your user interface. This method does not actually draw the string or alter the receiver's text in any way.

This method computes the metrics needed to draw the specified string. This method lays out the receiver's text and attempts to make it fit the specified size using the specified font and line break options. During layout, the method may break the text onto multiple lines to make it fit better. If the receiver's text does not completely fit in the specified size, it lays out as much of the text as possible and truncates it (for layout purposes only) according to the specified line break mode. It then returns the size of the resulting truncated string. 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.

Availability
Available in iOS 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 on a single line.

```
- (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 56).

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

Discussion
You can use this method to obtain the layout metrics you need to draw a string in your user interface. This method does not actually draw the string or alter the receiver's text in any way.

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. If the size of the string exceeds the given width, this method truncates the text (for layout purposes only) using the specified line break mode until it does conform to the maximum width; it then returns the size of the resulting truncated string.

Availability
Available in iOS 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, on a single line.

```
- (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 56).

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

Discussion
You can use this method to obtain the layout metrics you need to draw a string in your user interface. This method does not actually draw the string or alter the receiver's text in any way.

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 until it reaches the specified minimum font size. If it reaches the minimum font size, the method begins truncating the text (for layout purposes only) until the resulting truncated string does fit the width; it then then returns the size of that truncated string.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared in UIStringDrawing.h.

`UILineBreakModeCharacterWrap`
: Wrap or clip the string at the closest character boundary.

  Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `UIStringDrawing.h`.

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

Available in iOS 2.0 and later.

Declared in `UIStringDrawing.h`.

`UITextAlignmentRight`
Align text along the right edge.

Available in iOS 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 iOS 2.0 and later.

Declared in `UIStringDrawing.h`.

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

Available in iOS 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 iOS 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 iOS 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 60)
Creates and returns a value object that contains the specified point structure.

`+ valueWithCGRect:` (page 61)
Creates and returns a value object that contains the specified rectangle structure.

`+ valueWithCGSize:` (page 61)
Creates and returns a value object that contains the specified size structure.

`+ valueWithCGAffineTransform:` (page 60)
Creates and returns a value object that contains the specified affine transform data.

`+ valueWithUIEdgeInsets:` (page 61)
Creates and returns a value object that contains the specified edge inset data.

Accessing Data

`- CGPointValue` (page 62)
Returns a point structure representing the data in the receiver.

`- CGRectValue` (page 63)
Returns a rectangle structure representing the data in the receiver.

- `CGSizeValue` (page 63)
  Returns a size structure representing the data in the receiver.
- `CGAffineTransformValue` (page 62)
  Returns an affine transform structure representing the data in the receiver.
- `UIEdgeInsetsValue` (page 63)
  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 iOS 2.0 and later.

See Also
- `CGAffineTransformValue` (page 62)

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 iOS 2.0 and later.

See Also
- `CGPointValue` (page 62)

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 iOS 2.0 and later.

See Also
- CGRectValue (page 63)

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 iOS 2.0 and later.

See Also
- CGSizeValue (page 63)

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 iOS 2.0 and later.

See Also

- UIEdgeInsetsValue (page 63)

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 iOS 2.0 and later.

See Also

+ valueWithCGAffineTransform: (page 60)

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 iOS 2.0 and later.

See Also

+ valueWithCGPoint: (page 60)

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 iOS 2.0 and later.

See Also
+ valueWithCGRect: (page 61)

Related Sample Code
KeyboardAccessory

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 iOS 2.0 and later.

See Also
+ valueWithCGSize: (page 61)

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 iOS 2.0 and later.

See Also
`+ valueWithUIEdgeInsets:` (page 61)

Declared In
`UIGeometry.h`

UIAcceleration Class Reference

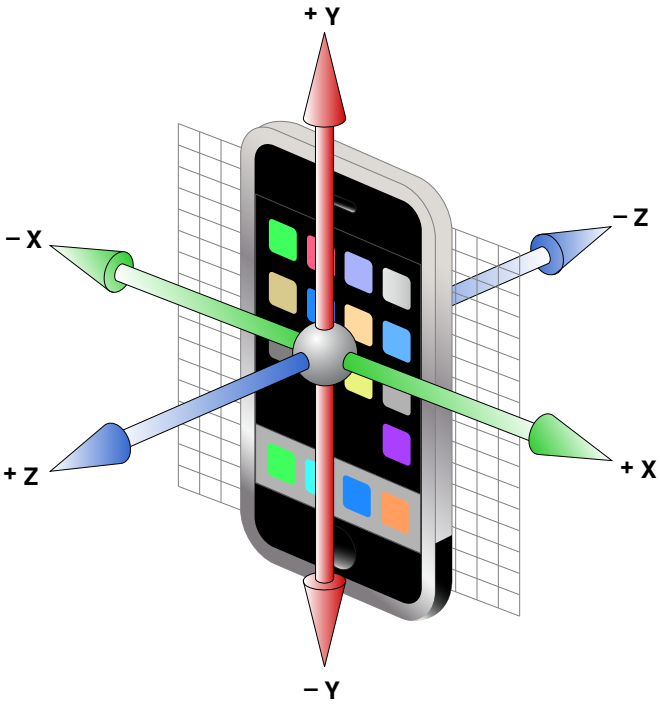
Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	

Overview

An instance of the `UIAcceleration` class, called an *acceleration event*, represents immediate, three-dimensional acceleration data. To receive accelerometer events, register an application object as a delegate of the shared `UIAccelerometer` object, as described in *UIAccelerometer Class Reference*.

Each acceleration event includes simultaneous acceleration readings along the three axes of the device, as shown in Figure 7-1.

Figure 7-1 Orientation of the device axes



The device accelerometer reports values for each axis in units of g-force, where a value of `1.0` represents acceleration of about +1 g along a given axis. When a device is laying still with its back on a horizontal surface, each acceleration event has approximately the following values:

```
x: 0
y: 0
z: -1
```

Individual acceleration values are of type `UIAccelerationValue` (page 68), equivalent to a `double`. Values can range over the accelerations found in normal use of a device.

Note: Acceleration event values are approximate—don't attempt to use them to make precise measurements. Apple recommends that you average accelerometer values over time to derive usable data.

If you want to detect specific types of motion as gestures—specifically, shaking motions—use the `UIEvent` class and its `UIEventTypeMotion` (page 266) event type. For details, see "Motion Events" in *Event Handling Guide for iOS*.

Tasks

Accessing the Acceleration Values

`x` (page 67) *property*
The acceleration value for the x axis of the device. (read-only)

`y` (page 67) *property*
The acceleration value for the y axis of the device. (read-only)

`z` (page 67) *property*
The acceleration value for the z axis of the device. (read-only)

`timestamp` (page 66) *property*
The relative time at which the acceleration event occurred. (read-only)

Properties

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

timestamp

The relative time at which the acceleration event occurred. (read-only)

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

Discussion
This value indicates the time relative to the device CPU time base register. Compare acceleration event timestamps to determine the elapsed time between them. Do not use a timestamp to determine the exact time at which an event occurred.

Availability
Available in iOS 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 (negative values) to right (positive values) across the face of the device.

Availability
Available in iOS 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 bottom (negative values) to top (positive values) across the face of the device.

Availability
Available in iOS 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 (negative values) to front (positive values) through the device.

Availability
Available in iOS 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, where the value 1.0 corresponds to the normal acceleration caused by gravity at the Earth's surface.

Availability
Available in iOS 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 iOS 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 71)
Returns the shared accelerometer object for the system.

Accessing the Accelerometer Properties

`updateInterval` (page 70) *property*
The interval at which to deliver acceleration data to the delegate.

`delegate` (page 70) *property*
The delegate object you want to receive acceleration events.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
`UIAccelerometer.h`

UIAccessibilityElement Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Availability	Available in iOS 3.0 and later.
Declared in	UIAccessibilityElement.h
Companion guide	Accessibility Programming Guide for iOS

Overview

The `UIAccessibilityElement` class encapsulates information about an item that should be accessible to users with disabilities, but that isn't accessible by default. For example, an icon or text image is not automatically accessible because it does not inherit from `UIView` (or `UIControl`). A view that contains such nonview items creates an instance of `UIAccessibilityElement` to represent each item that needs to be accessible.

The properties of an accessibility element provide information about the element, such as location and current value, to an assistive application. You might need to set an element's property even if you don't need to create an instance of `UIAccessibilityElement` to represent it. For example, if your application includes a button with a custom icon that means "solve," the button itself is already represented by an accessibility element because it is a subclass of `UIButton`. However, you need to supply information for the label and hint properties because this information is unique to this button. You can do this in Interface Builder or by setting the properties in the `UIAccessibility` informal protocol.

Tasks

Creating an Accessibility Element

`- initWithAccessibilityContainer:` (page 77)
Creates and initializes an accessibility element to represent an item in the specified container.

Accessing the Containing View

`accessibilityContainer` (page 74) *property*
The view that contains the accessibility element.

Determining Accessibility

`isAccessibilityElement` (page 76) *property*
A Boolean value indicating whether the item is an accessibility element an assistive application can access.

Accessing the Attributes of an Accessibility Element

`accessibilityLabel` (page 75) *property*
A string that succinctly identifies the accessibility element.

`accessibilityHint` (page 75) *property*
A string that briefly describes the result of performing an action on the accessibility element.

`accessibilityValue` (page 76) *property*
A string that represents the current value of the accessibility element.

`accessibilityFrame` (page 74) *property*
The frame of the accessibility element, in screen coordinates.

`accessibilityTraits` (page 75) *property*
The combination of traits that best characterize the accessibility element.

Properties

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

accessibilityContainer

The view that contains the accessibility element.

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

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

accessibilityFrame

The frame of the accessibility element, in screen coordinates.

```
@property(nonatomic, assign) CGRect accessibilityFrame
```

Discussion
When you create an accessibility element to represent an element in your application, you must set this property to the `CGRect` structure that specifies the object's screen location and size. (Objects that inherit from `UIView` include this information by default.)

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

accessibilityHint

A string that briefly describes the result of performing an action on the accessibility element.

```
@property(nonatomic, retain) NSString *accessibilityHint
```

Discussion
The hint is a brief, localized description of the result of performing an action on the element without identifying the element or the action. For example, the hint for a table row that contains an email message might be "Selects the message," but not "Tap this row to select the message."

By default, standard UIKit controls and views have system-provided hints. If you provide a custom control or view, however, you need to set this property appropriately so that assistive applications can supply accurate information to users with disabilities.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

accessibilityLabel

A string that succinctly identifies the accessibility element.

```
@property(nonatomic, retain) NSString *accessibilityLabel
```

Discussion
The label is a very short, localized string that identifies the accessibility element, but does not include the type of the control or view. For example, the label for a Save button is "Save," not "Save button."

By default, standard UIKit controls and views have labels that derive from their titles. If you provide a custom control or view, however, you need to set this property appropriately so that assistive applications can supply accurate information to users with disabilities.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

accessibilityTraits

The combination of traits that best characterize the accessibility element.

```
@property(nonatomic, assign) UIAccessibilityTraits accessibilityTraits
```

Discussion
A trait describes a single aspect of an element's behavior, state, or usage. Several traits are combined in this property (using an OR operation) to give a complete picture of the element to an assistive application. See "Accessibility Traits" in *UIAccessibility Protocol Reference* for a complete list of traits.

UIKit provides an appropriate combination of traits for all standard controls and views. When combining traits for a custom accessibility element, be sure to:

- Use common sense. Don't combine traits that characterize the element in mutually exclusive ways, such as combining the button and search-field traits.
- Combine the traits you select with the superclass's traits. Specifically, always combine your custom traits with `[super accessibilityTraits]` in the method you use to set a custom element's traits.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

accessibilityValue

A string that represents the current value of the accessibility element.

```
@property(nonatomic, retain) NSString *accessibilityValue
```

Discussion
The value is a localized string that contains the current value of an element. For example, the value of a slider might be 9.5 or 35% and the value of a text field is the text it contains.

Use the value property only when an accessibility element can have a value that is not represented by its label. For example, a volume slider's label might be "Volume," but its value is the current volume level. In this case, it's not enough for users to know the identity of the slider, because they also need to know its current value. The label of a Save button, on the other hand, tells users everything they need to know about the control; supplying the word "Save" as a value would be unnecessary and confusing.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

isAccessibilityElement

A Boolean value indicating whether the item is an accessibility element an assistive application can access.

```
@property(nonatomic, assign) BOOL isAccessibilityElement
```

Discussion
The default value for this property is `NO`. If the receiver is a UIKit control, the default value is `YES`.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

Instance Methods

initWithAccessibilityContainer:

Creates and initializes an accessibility element to represent an item in the specified container.

```
- (id)initWithAccessibilityContainer:(id)container
```

Parameters

container
The view that contains the item.

Return Value
An accessibility element to represent a non-view item in the container.

Discussion
In general, you do not create accessibility elements for items in your application because standard UIKit controls and views are accessible by default. However, if you have a view that contains nonview items, such as icons or text images, that need to be accessible to users with disabilities, you create accessibility elements to represent them. In this case, the containing view should implement the `UIAccessibilityContainer` informal protocol and use this method to create an accessibility element to represent each item that should be exposed to an assistive application.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityElement.h`

UIActionSheet Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIAlert.h
Related sample code	ToolbarSearch

Overview

Use the `UIActionSheet` class to present the user with a set of alternatives for how to proceed with a given task. You can also use action sheets to prompt the user to confirm a potentially dangerous action. The action sheet contains an optional title and one or more buttons, each of which corresponds to an action to take.

Use the properties and methods of this class to configure the action sheet's message, style, and buttons before presenting it. You should also assign a delegate to your action sheet. Your delegate object is responsible for performing the action associated with any buttons when they are tapped and should conform to the `UIActionSheetDelegate` protocol. For more information about implementing the methods of the delegate, see *UIActionSheetDelegate Protocol Reference*.

You can present an action sheet from a toolbar, tab bar, button bar item, or from a view. This class takes the starting view and current platform into account when determining how to present the action sheet. For applications running on iPhone and iPod touch devices, the action sheet typically slides up from the bottom of the window that owns the view. For applications running on iPad devices, the action sheet is typically displayed in a popover that is anchored to the starting view in an appropriate way. Taps outside of the popover automatically dismiss the action sheet, as do taps within any custom buttons. You can also dismiss it programmatically.

When presenting an action sheet on an iPad, there are times when you should not include a cancel button. If you are presenting just the action sheet, the system displays the action sheet inside a popover without using an animation. Because taps outside the popover dismiss the action sheet without selecting an item, this results in a default way to cancel the sheet. Including a cancel button would therefore only cause confusion. However, if you have an existing popover and are displaying an action sheet on top of other content using an animation, a cancel button is still appropriate. For more information see *iPad Human Interface Guidelines*.

Important: In iOS 4.0 and later, action sheets are not dismissed automatically when an application moves to the background. This behavior differs from earlier versions of the operating system, where action sheets were automatically cancelled (and their cancellation handler executed) as part of the termination sequence for the application. Now, it is up to you to decide whether to dismiss the action sheet (and execute its cancellation handler) or leave it visible for when your application moves back to the foreground. Remember that your application can still be terminated while in the background, so some type of action may be necessary in either case.

Tasks

Creating Action Sheets

`- initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:` (page 85)
Initializes the action sheet using the specified starting parameters.

Setting Properties

`delegate` (page 82) *property*
The receiver's delegate or `nil` if it doesn't have a delegate.

`title` (page 83) *property*
The string that appears in the receiver's title bar.

`visible` (page 83) *property*
A Boolean value that indicates whether the receiver is displayed. (read-only)

`actionSheetStyle` (page 81) *property*
The receiver's presentation style.

Configuring Buttons

`- addButtonWithTitle:` (page 84)
Adds a custom button to the action sheet.

`numberOfButtons` (page 83) *property*
The number of buttons on the action sheet. (read-only)

`- buttonTitleAtIndex:` (page 84)
Returns the title of the button at the specified index.

`cancelButtonIndex` (page 81) *property*
The index number of the cancel button.

`destructiveButtonIndex` (page 82) *property*
The index number of the destructive button.

`firstOtherButtonIndex` (page 83) *property*
The index of the first custom button. (read-only)

Presenting the Action Sheet

- `showFromTabBar:` (page 87)
  Displays an action sheet that originates from the specified tab bar.
- `showFromToolbar:` (page 88)
  Displays an action sheet that originates from the specified toolbar.
- `showInView:` (page 88)
  Displays an action sheet that originates from the specified view.
- `showFromBarButtonItem:animated:` (page 86)
  Displays an action sheet that originates from the specified bar button item.
- `showFromRect:inView:animated:` (page 87)
  Displays an action sheet that originates from the specified view.

Dismissing the Action Sheet

- `dismissWithClickedButtonIndex:animated:` (page 85)
  Dismisses the action sheet immediately using an optional animation.

Properties

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

actionSheetStyle

The receiver's presentation style.

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

Discussion
This property determines how the action sheet looks when it is presented. For a list of possible values, see the `UIActionSheetStyle` (page 89) constants.

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.h`

cancelButtonIndex

The index number of the cancel button.

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

Discussion
Button indices start at `0`. The default value of this property is normally `-1`, which indicates that no cancel button has been set. However, a cancel button may be created and set automatically by the `initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:` (page 85) method. If you use that method to create a cancel button, you should not change the value of this property.

When presenting an action sheet on an iPad, there are times when you should not include a cancel button. For more information on when you should include a cancel button, see the class overview or *iPad Human Interface Guidelines*.

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.h`

delegate

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

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

Discussion
For a list of methods your delegate object can implement, see *UIActionSheetDelegate Protocol Reference*.

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.h`

destructiveButtonIndex

The index number of the destructive button.

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

Discussion
Button indices start at `0`. The default value of this property is normally `-1`, which indicates that no destructive button has been set. However, a destructive button may be created and set automatically by the `initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:` (page 85) method. If you use that method to create a destructive button, you should not change the value of this property.

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.h`

firstOtherButtonIndex

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

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

Discussion
Button indices start at `0`. The default value of this property is `-1`, which indicates that there are no other custom buttons.

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.h`

numberOfButtons

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.h`

title

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIActionSheet.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 iOS 2.0 and later.

Declared In
UIActionSheet.h

Instance Methods

addButtonWithTitle:

Adds a custom button to the action sheet.

```
- (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 iOS 2.0 and later.

See Also
UIActionSheet (page 79)

Declared In
UIActionSheet.h

buttonTitleAtIndex:

Returns the title of the button at the specified 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 iOS 2.0 and later.

See Also
- showInView: (page 88)

Declared In
UIActionSheet.h

dismissWithClickedButtonIndex:animated:

Dismisses the action sheet immediately using an optional animation.

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

Parameters

buttonIndex

The index of the button that was clicked. Button indices start at 0.

animated

Specify YES to animate the dismissal of the action sheet or NO to remove the action sheet without an animation.

Discussion

You can use this method to dismiss the action sheet programmatically as needed. The action sheet also calls this method itself in response to the user tapping one of the buttons in the action sheet.

In iOS 4.0, you may want to call this method whenever your application moves to the background. An action sheet is not dismissed automatically when an application moves to the background. This behavior differs from previous versions of the operating system, where they were canceled automatically when the application was terminated. Dismissing the action sheet gives your application a chance to save changes or abort the operation and perform any necessary cleanup in case your application is terminated later.

Availability

Available in iOS 2.0 and later.

Declared In

UIActionSheet.h

initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle: otherButtonTitles:

Initializes the action sheet using the specified starting parameters.

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

Parameters

title

A string to display in the title area of the action sheet. Pass nil if you do not want to display any text in the title area.

delegate

The receiver's delegate object. Although this parameter may be nil, the delegate is used to respond to taps in the action sheet and should usually be provided.

cancelButtonTitle

The title of the cancel button. This button is added to the action sheet automatically and assigned an appropriate index, which is available from the cancelButtonIndex (page 81) property. This button is displayed in black to indicate that it represents the cancel action. Specify nil if you do not want a cancel button or are presenting the action sheet on an iPad.

destructiveButtonTitle

The title of the destructive button. This button is added to the action sheet automatically and assigned an appropriate index, which is available from the `destructiveButtonIndex` (page 82) property. This button is displayed in red to indicate that it represents a destructive behavior. Specify `nil` if you do not want a destructive button.

otherButtonTitles, ...

The titles of any additional buttons you want to add. This parameter consists of a `nil`-terminated, comma-separated list of strings. For example, to specify two additional buttons, you could specify the value `@"Button 1", @"Button 2", nil`.

Return Value
A newly initialized action sheet.

Discussion
The action sheet automatically sets the appearance of the destructive and cancel buttons. If the action sheet contains only one button, it does not apply the custom colors associated with the destructive and cancel buttons.

Availability
Available in iOS 2.0 and later.

Related Sample Code
ToolbarSearch

Declared In
`UIActionSheet.h`

showFromBarButtonItem:animated:

Displays an action sheet that originates from the specified bar button item.

```
- (void)showFromBarButtonItem:(UIBarButtonItem *)item animated:(BOOL)animated
```

Parameters

item

The bar button item from which the action sheet originates.

animated

Specify `YES` to animate the presentation of the action sheet or `NO` to present it immediately without any animation effects.

Discussion
On iPad, this method presents the action sheet in a popover and adds the toolbar that owns the button to the popover's list of passthrough views. Thus, taps in the toolbar result in the action methods of the corresponding toolbar items being called. If you want the popover to be dismissed when a different toolbar item is tapped, you must implement that behavior in your action handler methods.

Availability
Available in iOS 3.2 and later.

Declared In
`UIActionSheet.h`

showFromRect:inView:animated:

Displays an action sheet that originates from the specified view.

```
- (void)showFromRect:(CGRect)rect inView:(UIView *)view animated:(BOOL)animated
```

Parameters

rect

The portion of *view* from which to originate the action sheet.

view

The view from which to originate the action sheet.

animated

Specify `YES` to animate the presentation of the action sheet or `NO` to present it immediately without any animation effects.

Discussion

On iPad, this method displays the action sheet in a popover whose arrow points to the specified rectangle of the view. The popover does not overlap the specified rectangle.

Availability

Available in iOS 3.2 and later.

Declared In

`UIActionSheet.h`

showFromTabBar:

Displays an action sheet that originates from the specified tab bar.

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

Parameters

view

The tab bar from which the action sheet originates.

Discussion

The appearance of the action sheet is animated.

On iPad, this method centers the action sheet in the middle of the screen. Generally, if you want to present an action sheet relative to a tab bar in an iPad application, you should use the `showFromRect:inView:animated:` (page 87) method instead.

Availability

Available in iOS 2.0 and later.

See Also

`- showFromToolbar:` (page 88)

Declared In

`UIActionSheet.h`

showFromToolbar:

Displays an action sheet that originates from the specified toolbar.

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

Parameters

view

The toolbar from which the action sheet originates.

Discussion

The appearance of the action sheet is animated.

On iPad, this method centers the action sheet in the middle of the screen. Generally, if you want to present an action sheet relative to a toolbar in an iPad application, you should use the `showFromBarButtonItem:animated:` (page 86) method instead.

Availability

Available in iOS 2.0 and later.

See Also

`- showFromTabBar:` (page 87)

Declared In

`UIActionSheet.h`

showInView:

Displays an action sheet that originates from the specified view.

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

Parameters

view

The view from which the action sheet originates.

Discussion

The appearance of the action sheet is animated.

On iPad, this method centers the action sheet in the middle of the screen. Generally, if you want to present an action sheet in an iPad application, you should use the `showFromRect:inView:animated:` (page 87) method to display the action sheet instead.

Availability

Available in iOS 2.0 and later.

Related Sample Code

ToolbarSearch

Declared In

`UIActionSheet.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 89).

Available in iOS 2.0 and later.

Declared in `UIActionSheet.h`.

`UIActionSheetStyleDefault`

The default style.

Available in iOS 2.0 and later.

Declared in `UIActionSheet.h`.

`UIActionSheetStyleBlackTranslucent`

A black translucent style.

Available in iOS 2.0 and later.

Declared in `UIActionSheet.h`.

`UIActionSheetStyleBlackOpaque`

A black opaque style.

Available in iOS 2.0 and later.

Declared in `UIActionSheet.h`.

Availability

Available in iOS 2.0 and later.

Declared In

`UIActionSheet.h`

UIActivityIndicatorView Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIActivityIndicatorView.h
Related sample code	BonjourWeb CryptoExercise WiTap

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 94) and `stopAnimating` (page 94) methods. If the `hidesWhenStopped` (page 92) property is set to `YES`, the indicator is automatically hidden when animation stops.

Tasks

Initializing an UIActivityIndicatorView Object

`- initWithActivityIndicatorStyle:` (page 93)
Initializes and returns an activity-indicator object.

Managing the Activity Indicator

`- startAnimating` (page 94)
Starts the animation of the progress indicator.

- `stopAnimating` (page 94)
  Stops the animation of the progress indicator.
- `isAnimating` (page 93)
  Returns whether the receiver is animating.

`hidesWhenStopped` (page 92) *property*
A Boolean value that controls whether the receiver is hidden when the animation is stopped.

Managing the Indicator Style

`activityIndicatorViewStyle` (page 92) *property*
The style of the indeterminate progress indicator.

Properties

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

activityIndicatorViewStyle

The style of the indeterminate progress indicator.

```
@property UIActivityIndicatorViewStyle activityIndicatorViewStyle
```

Discussion
See `UIActivityIndicatorStyle` (page 94) for valid constants. The default value is `UIActivityIndicatorViewStyleWhite` (page 95).

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb
WiTap

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 703) property (`UIView`) to `YES` when receiver is not animating. If the `hidesWhenStopped` (page 92) property is `NO`, the receiver is not hidden when animation stops. You stop an animating progress indicator with the `stopAnimating` (page 94) method.

Availability
Available in iOS 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 94) 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 92) property.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `startAnimating` (page 94)
- `stopAnimating` (page 94)

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 94) is called.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb
CryptoExercise
WiTap

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 94). When animating is stopped, the indicator is hidden, unless `hidesWhenStopped` (page 92) is `NO`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 iOS 2.0 and later.

Declared in `UIActivityIndicatorView.h`.

`UIActivityIndicatorViewStyleWhite`

The standard white style of indicator (the default).

Available in iOS 2.0 and later.

Declared in `UIActivityIndicatorView.h`.

`UIActivityIndicatorViewStyleGray`

The standard gray style of indicator.

Available in iOS 2.0 and later.

Declared in `UIActivityIndicatorView.h`.

Discussion

You set the value of the `activityIndicatorViewStyle` (page 92) property with these constants.

Availability

Available in iOS 2.0 and later.

Declared In

`UIActivityIndicatorView.h`

UIAlertView Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIAlert.h
Related sample code	AddMusic CryptoExercise GKRocket GKTank WiTap

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 action sheet (an instance of `UIActionSheet`).

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 103) method to display an alert view once it is configured.

Important: In iOS 4.0 and later, alert views are not dismissed automatically when an application moves to the background. This behavior differs from earlier versions of the operating system, where alert views were automatically cancelled (and their cancellation handler executed) as part of the termination sequence for the application. Now, it is up to you to decide whether to dismiss the alert view (and execute its cancellation handler) or leave it visible for when your application moves back to the foreground. Remember that your application can still be terminated while in the background, so some type of action may be necessary in either case.

Tasks

Creating Alert Views

- `initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:` (page 102)
  Convenience method for initializing an alert view.

Setting Properties

`delegate` (page 99) *property*
The receiver's delegate or `nil` if it doesn't have a delegate.

`title` (page 100) *property*
The string that appears in the receiver's title bar.

`message` (page 100) *property*
Descriptive text that provides more details than the title.

`visible` (page 100) *property*
A Boolean value that indicates whether the receiver is displayed. (read-only)

Configuring Buttons

- `addButtonWithTitle:` (page 101)
  Adds a button to the receiver with the given title.

`numberOfButtons` (page 100) *property*
The number of buttons on the alert view. (read-only)

- `buttonTitleAtIndex:` (page 101)
  Returns the title of the button at the given index.

`cancelButtonIndex` (page 99) *property*
The index number of the cancel button.

`firstOtherButtonIndex` (page 99) *property*
The index of the first other button. (read-only)

Displaying

`- show` (page 103)
Displays the receiver using animation.

Dismissing

`- dismissWithClickedButtonIndex:animated:` (page 102)
Dismisses the receiver, optionally with animation.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

Declared In
`UIAlertView.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 iOS 2.0 and later.

Declared In
`UIAlertView.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 iOS 2.0 and later.

Declared In
`UIAlertView.h`

message

Descriptive text that provides more details than the title.

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIAlertView.h`

numberOfButtons

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIAlertView.h`

title

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIAlertView.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 iOS 2.0 and later.

Related Sample Code
GKRocket
GKTank

Declared In
`UIAlertView.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 iOS 2.0 and later.

See Also
`@property message` (page 100)

Related Sample Code
GKRocket

Declared In
`UIAlertView.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 iOS 2.0 and later.

See Also
"Displaying" (page 99)

Declared In
UIAlertView.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.

Discussion
In iOS 4.0, you may want to call this method whenever your application moves to the background. An alert view is not dismissed automatically when an application moves to the background. This behavior differs from previous versions of the operating system, where they were canceled automatically when the application was terminated. Dismissing the alert view gives your application a chance to save changes or abort the operation and perform any necessary cleanup in case your application is terminated later.

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKRocket

Declared In
UIAlertView.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 101) specifying this title.

otherButtonTitles,
The title of another button.

Using this argument is equivalent to invoking `addButtonWithTitle:` (page 101) with this title to add more buttons.

...
Titles of additional buttons to add to the receiver, terminated with `nil`.

Return Value
Newly initialized alert view.

Availability
Available in iOS 2.0 and later.

See Also
`- addButtonWithTitle:` (page 101)

Related Sample Code
AddMusic
CryptoExercise
GKRocket
GKTank
WiTap

Declared In
`UIAlertView.h`

show

Displays the receiver using animation.

```
- (void)show
```

Availability
Available in iOS 2.0 and later.

Related Sample Code
AddMusic
CryptoExercise
GKRocket
GKTank
WiTap

Declared In
`UIAlertView.h`

UIApplication Class Reference

Inherits from	UIResponder : NSObject
Conforms to	UIActionSheetDelegate NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIApplication.h
Related sample code	AddMusic GKRocket GLSprite MoviePlayer WiTap

Overview

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

Every application must have exactly one instance of `UIApplication` (or a subclass of `UIApplication`). When an application is launched, the `UIApplicationMain` (page 1030) 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 115) 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 121) method. For example, an application opening an email URL with this method may cause the mail client to launch and display the message.

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 touch events, and turn proximity sensing (of the user's face) off and on again.

For iOS 3.0, `UIApplication` has added methods for remote-notification registration, for triggering of the undo-redo UI (`applicationSupportsShakeToEdit` (page 110)), and for determining whether any installed application can open a URL (`canOpenURL:` (page 118)).

In iOS 4.0, `UIApplication` has added methods and constants for managing background execution, for scheduling and canceling local notifications, and for controlling the reception of remote-control events.

`UIApplication` defines a delegate that must adopt the `UIApplicationDelegate` protocol and implement some of the protocol methods.

Subclassing Notes

You might decide to subclass `UIApplication` to override `sendEvent:` (page 124) or `sendAction:to:from:forEvent:` (page 123) to implement custom event and action dispatching. However, there is rarely a valid need to extend this class; the application delegate (`UIApplicationDelegate` is sufficient for most occasions. If you do subclass `UIApplication`, be very sure of what you are trying to accomplish with the subclass.

Tasks

Getting the Application Instance

`+ sharedApplication` (page 115)
Returns the singleton application instance.

Setting and Getting the Delegate

`delegate` (page 110) *property*
The delegate of the application object.

Getting Application Windows

`keyWindow` (page 111) *property*
The application's key window. (read-only)

`windows` (page 115) *property*
The application's visible windows. (read-only)

Controlling and Handling Events

`- sendEvent:` (page 124)
Dispatches an event to the appropriate responder objects in the application.

`- sendAction:to:from:forEvent:` (page 123)
Sends an action message identified by selector to a specified target.

- `beginIgnoringInteractionEvents` (page 117)
  Tells the receiver to suspend the handling of touch-related events.
- `endIgnoringInteractionEvents` (page 120)
  Tells the receiver to resume the handling of touch-related events.
- `isIgnoringInteractionEvents` (page 121)
  Returns whether the receiver is ignoring events initiated by touches on the screen.

`applicationSupportsShakeToEdit` (page 110) *property*
A Boolean value that determines whether shaking the device displays the undo-redo user interface.

`proximitySensingEnabled` (page 113) *property*
A Boolean value that determines whether proximity sensing is enabled. (Deprecated. The properties `proximityMonitoringEnabled` (page 242) and `proximityState` (page 243) of the `UIDevice` class are the replacements.)

Opening a URL Resource

- `openURL:` (page 121)
  Opens the resource at the specified URL.
- `canOpenURL:` (page 118)
  Returns whether an application can open a given URL resource.

Registering for Remote Notifications

- `registerForRemoteNotificationTypes:` (page 122)
  Register to receive notifications of the specified types from a provider via Apple Push Service.
- `unregisterForRemoteNotifications` (page 128)
  Unregister for notifications received from Apple Push Service.
- `enabledRemoteNotificationTypes` (page 119)
  Returns the types of notifications the application accepts.

Managing Application Activity

`idleTimerDisabled` (page 111) *property*
A Boolean value that controls whether the idle timer is disabled for the application.

Managing Background Execution

`applicationState` (page 109) *property*
The runtime state of the application. (read-only)

`backgroundTimeRemaining` (page 110) *property*
The amount of time the application has to run in the background. (read-only)

- `beginBackgroundTaskWithExpirationHandler:` (page 116)
  Marks the beginning of a new long-running background task.

- `endBackgroundTask:` (page 119)
  Marks the end of a specific long-running background task.
- `setKeepAliveTimeout:handler:` (page 125)
  Configures a periodic handler for VoIP applications.
- `clearKeepAliveTimeout` (page 119)
  Removes a previously installed periodic handler block.

Registering for Local Notifications

- `scheduleLocalNotification:` (page 123)
  Schedules a local notification for delivery at its encapsulated date and time.
- `presentLocalNotificationNow:` (page 122)
  Presents a local notification immediately.
- `cancelLocalNotification:` (page 118)
  Cancels the delivery of the specified scheduled local notification.
- `cancelAllLocalNotifications` (page 117)
  Cancels the delivery of all scheduled local notifications.
- `scheduledLocalNotifications` (page 123)
  Returns all currently scheduled local notifications.

Determining the Availability of Protected Content

`protectedDataAvailable` (page 112) *property*
A Boolean value indicating whether content protection is active. (read-only)

Registering for Remote Control Events

- `beginReceivingRemoteControlEvents` (page 117)
  Tells the application to begin receiving remote-control events.
- `endReceivingRemoteControlEvents` (page 120)
  Tells the application to stop receiving remote-control events.

Managing Status Bar Orientation

- `setStatusBarOrientation:animated:` (page 126)
  Sets the application's status bar to the specified orientation, optionally animating the transition.

`statusBarOrientation` (page 114) *property*
The current orientation of the application's status bar.

`statusBarOrientationAnimationDuration` (page 114) *property*
The animation duration in seconds for the status bar during a 90 degree orientation change. (read-only)

Controlling Application Appearance

- `setStatusBarHidden:withAnimation:` (page 126)
  Hides or shows the status bar, optionally animating the transition.
- `setStatusBarHidden:animated:` (page 126)
  Hides or shows the status bar, optionally animating the transition. (Deprecated. Use `setStatusBarHidden:withAnimation:` (page 126) instead.)

`statusBarHidden` (page 113) *property*
A Boolean value that determines whether the status bar is hidden.

- `setStatusBarStyle:animated:` (page 127)
  Sets the style of the status bar, optionally animating the transition to the new style.

`statusBarStyle` (page 114) *property*
The current style of the status bar.

`statusBarFrame` (page 113) *property*
The frame rectangle defining the area of the status bar. (read-only)

`networkActivityIndicatorVisible` (page 112) *property*
A Boolean value that turns an indicator of network activity on or off.

`applicationIconBadgeNumber` (page 109) *property*
The number currently set as the badge of the application icon in Springboard.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

Declared In
`UIApplication.h`

applicationState

The runtime state of the application. (read-only)

```
@property(nonatomic,readonly) UIApplicationState applicationState
```

Discussion
An application may be active, inactive, or running in the background. You can use the value in this property to determine which of these states the application is currently in.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

applicationSupportsShakeToEdit

A Boolean value that determines whether shaking the device displays the undo-redo user interface.

```
@property(nonatomic) BOOL applicationSupportsShakeToEdit
```

Discussion
The default value is `YES`. Set the property to `NO` if you don't want your application to display the Undo and Redo buttons when users shake the device.

Availability
Available in iOS 3.0 and later.

Declared In
`UIApplication.h`

backgroundTimeRemaining

The amount of time the application has to run in the background. (read-only)

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

Discussion
This property contains the amount of time the application has to run in the background before it may be forcibly terminated by the system. While the application is running in the foreground, the value in this property remains suitably large. If the application starts one or more long-running tasks using the `beginBackgroundTaskWithExpirationHandler:` (page 116) method and then transitions to the background, the value of this property is adjusted to reflect the amount of time the application has left to run.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Related Sample Code
WiTap

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. This includes audio applications. With appropriate use of Audio Session Services, playback and recording proceed uninterrupted when the screen turns off. The only applications that should disable the idle timer are mapping applications, games, or similar programs with sporadic user interaction.

Availability
Available in iOS 2.0 and later.

Related Sample Code
aurioTouch
GKTank

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 115) array that is most recently sent the `makeKeyAndVisible` (page 798) message.

Availability
Available in iOS 2.0 and later.

See Also
`@property windows` (page 115)

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Declared In
`UIApplication.h`

protectedDataAvailable

A Boolean value indicating whether content protection is active. (read-only)

```
@property(nonatomic,readonly,getter=isProtectedDataAvailable) BOOL
    protectedDataAvailable
```

Discussion
The value of this property is `NO` if content protection is enabled and the device is currently locked. The value of this property is set to `YES` if the device is unlocked or if content protection is not enabled.

When the value of this property is `NO`, files that were assigned the `NSFileProtectionComplete` protection key cannot be read or written by your application. The user must unlock the device before your application can access them.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

proximitySensingEnabled

A Boolean value that determines whether proximity sensing is enabled. (Deprecated in iOS 3.0. The properties `proximityMonitoringEnabled` (page 242) and `proximityState` (page 243) of the `UIDevice` class are the replacements.)

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

Discussion
`YES` if proximity sensing is enabled; otherwise `NO`. Enabling proximity sensing tells iOS 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 iOS 2.0 and later.
Deprecated in iOS 3.0.

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 iOS 2.0 and later.

See Also
`@property statusBarHidden` (page 113)
`@property statusBarStyle` (page 114)

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 iOS 2.0 and later.

See Also
`- setStatusBarHidden:animated:` (page 126)

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 128) 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 *View Controller Programming Guide for iOS*.

Availability
Available in iOS 2.0 and later.

See Also
`- setStatusBarOrientation:animated:` (page 126)

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 iOS 2.0 and later.

See Also
`- setStatusBarOrientation:animated:` (page 126)

Declared In
`UIApplication.h`

statusBarStyle

The current style of the status bar.

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

Discussion
The value of the property is a UIStatusBarStyle (page 129) constant that indicates the style of status. The default style is `UIStatusBarStyleDefault` (page 129). The animation slides the status bar out for the old orientation and slides it in for the new orientation.

Availability
Available in iOS 2.0 and later.

See Also
`@property statusBarHidden` (page 113)
`@property statusBarFrame` (page 113)

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 back to front.

Availability
Available in iOS 2.0 and later.

See Also
`@property keyWindow` (page 111)

Related Sample Code
MoviePlayer

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 1030) function.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AddMusic
BonjourWeb
GKRocket
MoviePlayer
WiTap

Declared In
`UIApplication.h`

Instance Methods

beginBackgroundTaskWithExpirationHandler:

Marks the beginning of a new long-running background task.

```
- (UIBackgroundTaskIdentifier)beginBackgroundTaskWithExpirationHandler:(void(^)(void))handler
```

Parameters

handler

A handler to be called shortly before the application's remaining background time reaches 0. You should use this handler to clean up and mark the end of the background task. Failure to end the task explicitly will result in the termination of the application.

Return Value
A unique identifier for the new background task. You must pass this value to the `endBackgroundTask:` method to mark the end of this task. This method returns `UIBackgroundTaskInvalid` (page 134) if running in the background is not possible.

Discussion
This method lets your application continue to run for a period of time after it transitions to the background. You should call this method at times where leaving a task unfinished might be detrimental to your application's user experience. For example, your application could call this method to ensure that had enough time to transfer an important file to a remote server or at least attempt to make the transfer and note any errors. You should not use this method simply to keep your application running after it moves to the background.

Each call to this method must be balanced by a matching call to the `endBackgroundTask:` method. Applications running background tasks have a finite amount of time in which to run them. (You can find out how much time is available using the `backgroundTimeRemaining` (page 110) property.) If you do not call `endBackgroundTask:` for each task before time expires, the system terminates the application. If you provide a block object in the *handler* parameter, the system calls your handler before time expires to give you a chance to end the task.

You can call this method at any point in your application's execution. You may also call this method multiple times to mark the beginning of several background tasks that run in parallel. However, each task must be ended separately. You identify a given task using the value returned by this method.

Availability
Available in iOS 4.0 and later.

See Also
- `endBackgroundTask:` (page 119)

Declared In
`UIApplication.h`

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 120) method.

Availability
Available in iOS 2.0 and later.

See Also
- `isIgnoringInteractionEvents` (page 121)

Declared In
`UIApplication.h`

beginReceivingRemoteControlEvents

Tells the application to begin receiving remote-control events.

```
- (void)beginReceivingRemoteControlEvents
```

Discussion
Remote-control events originate as commands issued by headsets and external accessories that are intended to control multimedia presented by an application. To stop the reception of remote-control events, you must call `endReceivingRemoteControlEvents` (page 120).

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

cancelAllLocalNotifications

Cancels the delivery of all scheduled local notifications.

```
- (void)cancelAllLocalNotifications
```

Availability
Available in iOS 4.0 and later.

See Also
- `cancelLocalNotification:` (page 118)
- `scheduleLocalNotification:` (page 123)
- `presentLocalNotificationNow:` (page 122)

Declared In
`UIApplication.h`

cancelLocalNotification:

Cancels the delivery of the specified scheduled local notification.

```
- (void)cancelLocalNotification:(UILocalNotification *)notification
```

Parameters

notification

The local notification to cancel.

Discussion
Calling this method also programmatically dismisses the notification if it is currently displaying an alert.

Availability
Available in iOS 4.0 and later.

See Also
- `cancelAllLocalNotifications` (page 117)
- `scheduleLocalNotification:` (page 123)
- `presentLocalNotificationNow:` (page 122)

Declared In
`UIApplication.h`

canOpenURL:

Returns whether an application can open a given URL resource.

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

Parameters

url

A URL object that identifies a given resource. The URL's scheme—possibly a custom scheme—identifies which application can handle the URL.

Return Value
`NO` if no application is available that will accept the URL; otherwise, returns `YES`.

Discussion
This method guarantees that that if `openURL:` (page 121) is called, another application will be launched to handle it. It does not guarantee that the full URL is valid.

Availability
Available in iOS 3.0 and later.

Declared In
UIApplication.h

clearKeepAliveTimeout

Removes a previously installed periodic handler block.

```
- (void)clearKeepAliveTimeout
```

Discussion
If your VoIP application no longer needs to be woken up at periodic intervals, you can use this method to remove any previously installed handler.

Availability
Available in iOS 4.0 and later.

Declared In
UIApplication.h

enabledRemoteNotificationTypes

Returns the types of notifications the application accepts.

```
- (UIRemoteNotificationType)enabledRemoteNotificationTypes
```

Return Value
A bit mask whose values indicate the types of notifications the user has requested for the application. See `UIRemoteNotificationType` (page 132) for valid bit-mask values.

Discussion
The values in the returned bit mask indicate the types of notifications currently enabled for the application. These types are first set when the application calls the `registerForRemoteNotificationTypes:` (page 122) method to register itself with Apple Push Notification Service. Thereafter, the user may modify these accepted notification types in the Notifications preference of the Settings application. This method returns those initial or modified values. iOS does not display or play notification types specified in the notification payload that are not one of the enabled types. For example, the application might accept icon-badging as a form of notification, but might reject sounds and alert messages, even if they are specified in the notification payload.

Availability
Available in iOS 3.0 and later.

See Also
- `unregisterForRemoteNotifications` (page 128)

Declared In
UIApplication.h

endBackgroundTask:

Marks the end of a specific long-running background task.

```
- (void)endBackgroundTask:(UIBackgroundTaskIdentifier)identifier
```

Parameters

identifier

An identifier returned by the `beginBackgroundTaskWithExpirationHandler:` method.

Discussion

You must call this method to end a task that was started using the `beginBackgroundTaskWithExpirationHandler:` method. If you do not, the system may terminate your application.

Availability

Available in iOS 4.0 and later.

See Also

`- beginBackgroundTaskWithExpirationHandler:` (page 116)

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 117) method, the animation or transition concludes. Nested calls of this method should match nested calls of the `beginIgnoringInteractionEvents` method.

Availability

Available in iOS 2.0 and later.

See Also

`- isIgnoringInteractionEvents` (page 121)

Declared In

`UIApplication.h`

endReceivingRemoteControlEvents

Tells the application to stop receiving remote-control events.

```
- (void)endReceivingRemoteControlEvents
```

Discussion

Remote-control events originate as commands issued by headsets and external accessories that are intended to control multimedia presented by an application.

Availability

Available in iOS 4.0 and later.

See Also
- `beginReceivingRemoteControlEvents` (page 117)

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 117) and `endIgnoringInteractionEvents` (page 120) calls are at least one level deep.

Availability
Available in iOS 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.

You may call `canOpenURL:` (page 118) before calling this one to verify that there is an application that can handle it.

Availability
Available in iOS 2.0 and later.

See Also
`application:handleOpenURL:` (page 843) (`UIApplicationDelegate`)

Related Sample Code
BonjourWeb

Declared In
UIApplication.h

presentLocalNotificationNow:

Presents a local notification immediately.

```
- (void)presentLocalNotificationNow:(UILocalNotification *)notification
```

Parameters

notification

A local notification that the operating system presents for the application immediately, regardless of the value of the notification's `fireDate` property. Applications running in the background state can immediately present local notifications when there are incoming chats, messages, or updates. Because the operating system copies *notification*, you may release it once you have scheduled it.

Availability
Available in iOS 4.0 and later.

See Also
- `scheduleLocalNotification:` (page 123)
- `cancelLocalNotification:` (page 118)
- `cancelAllLocalNotifications` (page 117)

Declared In
UIApplication.h

registerForRemoteNotificationTypes:

Register to receive notifications of the specified types from a provider via Apple Push Service.

```
- (void)registerForRemoteNotificationTypes:(UIRemoteNotificationType)types
```

Parameters

types

A bit mask specifying the types of notifications the application accepts. See `UIRemoteNotificationType` (page 132) for valid bit-mask values.

Discussion
When you send this message, the device initiates the registration process with Apple Push Service. If it succeeds, the application delegate receives a device token in the `application:didRegisterForRemoteNotificationsWithDeviceToken:` (page 842) method; if registration doesn't succeed, the delegate is informed via the `application:didFailToRegisterForRemoteNotificationsWithError:` (page 838) method. If the application delegate receives a device token, it should connect with its provider and pass it the token.

iOS does not display or play notification types specified in the notification payload that are not one of the requested ones. For example, if alert messages are not one of the accepted notification types, iOS does not display an alert even if one is specified in the notification payload. To find out what the application's current notification types are, call the `enabledRemoteNotificationTypes` (page 119) method.

Availability
Available in iOS 3.0 and later.

See Also
- `unregisterForRemoteNotifications` (page 128)

Declared In
`UIApplication.h`

scheduledLocalNotifications

Returns all currently scheduled local notifications.

```
- (NSArray *)scheduledLocalNotifications
```

Return Value
An array of `UILocalNotification` instances representing the current scheduled local notifications.

Availability
Available in iOS 4.0 and later.

See Also
- `cancelLocalNotification:` (page 118)

Declared In
`UIApplication.h`

scheduleLocalNotification:

Schedules a local notification for delivery at its encapsulated date and time.

```
- (void)scheduleLocalNotification:(UILocalNotification *)notification
```

Parameters

notification

A local notification that the operating system delivers for the application at the date and time specified in the `fireDate` property of *notification*. Because the operating system copies *notification*, you may release it once you have scheduled it.

Availability
Available in iOS 4.0 and later.

See Also
- `presentLocalNotificationNow:` (page 122)
- `cancelLocalNotification:` (page 118)
- `cancelAllLocalNotifications` (page 117)

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 parameters 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 iOS 2.0 and later.

See Also

`- sendEvent:` (page 124)

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. iOS calls this method for public events only.

Availability
Available in iOS 2.0 and later.

See Also
`- sendAction:to:from:forEvent:` (page 123)

Declared In
`UIApplication.h`

setKeepAliveTimeout:handler:

Configures a periodic handler for VoIP applications.

```
- (BOOL)setKeepAliveTimeout:(NSTimeInterval)timeout
    handler:(void(^)(void))keepAliveHandler
```

Parameters

timeout
The maximum interval (measured in seconds) at which your application should be woken up to check its VoIP connection. The minimum acceptable timeout value is 600 seconds.

keepAliveHandler
A block that performs the tasks needed to maintain your VoIP network connection.

Return Value
`YES` if the handler was installed or `NO` if it was not.

Discussion
A voice-over-IP (VoIP) application can use this method to install a handler whose job is to maintain the application's network connection with a VoIP server. This handler is guaranteed to be called before the specified timeout value but may be called at a slightly different time interval in order to better align execution of your handler with other system tasks, and thereby save power. Your handler has a maximum of 30 seconds to perform any needed tasks and exit. If it does not exit before time expires, the application is terminated.

Timeout values and handlers are not persisted between application launches. Therefore, if your application is terminated for any reason, you must reinstall the handler during the next launch cycle.

For calls to this method to succeed, the application must have the `voip` value in the array associated with the `UIBackgroundModes` key in its `Info.plist` file. Calling this method replaces the previously installed handler and timeout values, if any.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

setStatusBarHidden:animated:

Hides or shows the status bar, optionally animating the transition. (Deprecated in iOS 3.2. Use `setStatusBarHidden:withAnimation:` (page 126) instead.)

```
- (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 fades the status bar out or in at the top of the interface, depending on the value of *hidden*.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 3.2.

See Also

`@property statusBarHidden` (page 113)

Declared In

`UIApplication.h`

setStatusBarHidden:withAnimation:

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

```
- (void)setStatusBarHidden:(BOOL)hiddenwithAnimation:(UIStatusBarAnimation)animation
```

Parameters

hidden

`YES` to hide the status bar, `NO` to show the status bar.

animation

A constant that indicates whether there should be an animation and, if one is requested, whether it should fade the status bar in or out or whether it should slide the status bar in or out.

Discussion

See the descriptions of the constants of the `UIStatusBarAnimation` (page 130) type for more information.

Availability

Available in iOS 3.2 and later.

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 128) for details. The default value is `UIInterfaceOrientationPortrait` (page 128).

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 114) 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 iOS 2.0 and later.

See Also

`@property statusBarOrientation` (page 114)

`@property statusBarOrientationAnimationDuration` (page 114)

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 129) 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 iOS 2.0 and later.

See Also

`@property statusBarStyle` (page 114)

Related Sample Code

AddMusic

Declared In

`UIApplication.h`

unregisterForRemoteNotifications

Unregister for notifications received from Apple Push Service.

```
- (void)unregisterForRemoteNotifications
```

Discussion
You should call this method in rare circumstances only, such as when a new version of the application drops support for remote notifications. Users can temporarily prevent applications from receiving remote notifications through the Notifications section of the Settings application. Applications unregistered through this method can always re-register.

Availability
Available in iOS 3.0 and later.

See Also
- `registerForRemoteNotificationTypes:` (page 122)
- `enabledRemoteNotificationTypes` (page 119)

Declared In
`UIApplication.h`

Constants

UIInterfaceOrientation

The orientation of the application's user interface.

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

Constants

`UIInterfaceOrientationPortrait`
The device is in portrait mode, with the device held upright and the home button on the bottom.
Available in iOS 2.0 and later.
Declared in `UIApplication.h`.

`UIInterfaceOrientationPortraitUpsideDown`
The device is in portrait mode but upside down, with the device held upright and the home button at the top.
Available in iOS 2.0 and later.
Declared in `UIApplication.h`.

`UIInterfaceOrientationLandscapeLeft`
The device is in landscape mode, with the device held upright and the home button on the left side.
Available in iOS 2.0 and later.
Declared in `UIApplication.h`.

`UIInterfaceOrientationLandscapeRight`
The device is in landscape mode, with the device held upright and the home button on the right side.

Available in iOS 2.0 and later.

Declared in `UIApplication.h`.

Discussion
You use these constants in the `statusBarOrientation` (page 114) property and the `setStatusBarOrientation:animated:` (page 126) method. Notice that `UIDeviceOrientationLandscapeRight`is assigned to `UIInterfaceOrientationLandscapeLeft` and `UIDeviceOrientationLandscapeLeft` is assigned to `UIInterfaceOrientationLandscapeRight`; the reason for this is that rotating the device requires rotating the content in the opposite direction.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared in `UIApplication.h`.

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

Available in iOS 2.0 and later.

Declared in `UIApplication.h`.

`UIStatusBarStyleBlackOpaque`
An opaque black style.

Available in iOS 2.0 and later.

Declared in `UIApplication.h`.

Special Considerations

On iPad devices, the `UIStatusBarStyleDefault` and `UIStatusBarStyleBlackTranslucent` styles default to the `UIStatusBarStyleBlackOpaque` appearance.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

UIStatusBarAnimation

The animation applied to the status bar as it is hidden or made visible.

```
typedef enum {
    UIStatusBarAnimationNone,
    UIStatusBarAnimationFade,
    UIStatusBarAnimationSlide,
} UIStatusBarAnimation;
```

Constants

`UIStatusBarAnimationNone`

No animation is applied to the status bar as it is shown or hidden.

Available in iOS 3.2 and later.

Declared in `UIApplication.h`.

`UIStatusBarAnimationFade`

The status bar fades in and out as it is shown or hidden, respectively.

Available in iOS 3.2 and later.

Declared in `UIApplication.h`.

`UIStatusBarAnimationSlide`

The status bar slides in or out as it is shown or hidden, respectively.

Available in iOS 3.2 and later.

Declared in `UIApplication.h`.

Discussion

Constants of the `UIStatusBarAnimation` type are arguments of the `setStatusBarHidden:withAnimation:` (page 126) method.

Availability

Available in iOS 3.2 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 iOS 2.0 and later.

Declared in `UIApplication.h`.

Declared In

`UIApplication.h`

Launch Options Keys

Keys used to access values in the launch options dictionary passed to the `application:didFinishLaunchingWithOptions:` (page 839) method of the application delegate.

```
NSString *const UIApplicationLaunchOptionsURLKey;
NSString *const UIApplicationLaunchOptionsSourceApplicationKey;
NSString *const UIApplicationLaunchOptionsRemoteNotificationKey;
NSString *const UIApplicationLaunchOptionsAnnotationKey;
NSString *const UIApplicationLaunchOptionsLocalNotificationKey;
NSString *const UIApplicationLaunchOptionsLocationKey;
```

Constants

`UIApplicationLaunchOptionsURLKey`

You use this key to access contents of the dictionary passed in the second parameter of the `UIApplicationDelegate` method `application:didFinishLaunchingWithOptions:` (page 839). It returns an `NSURL` object that another application specified in `openURL:` (page 121), which resulted in iOS launching the application to handle the URL resource.

This key is also used to access the same value in the `userInfo` dictionary of the notification named `UIApplicationDidFinishLaunchingNotification` (page 136).

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

`UIApplicationLaunchOptionsSourceApplicationKey`

You use this key to access contents of the dictionary passed in the second parameter of the `UIApplicationDelegate` method `application:didFinishLaunchingWithOptions:` (page 839). It returns an `NSString` object that represents the bundle ID of the application that requested the launch by calling `openURL:` (page 121).

This key is also used to access the same value in the `userInfo` dictionary of the notification named `UIApplicationDidFinishLaunchingNotification` (page 136).

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

`UIApplicationLaunchOptionsRemoteNotificationKey`

You use this key to access contents of the dictionary passed in the second parameter of the `UIApplicationDelegate` method `application:didFinishLaunchingWithOptions:` (page 839). It returns a dictionary representing the payload of the remote notification.

See the description of `application:didReceiveRemoteNotification:` (page 841) for further information. This key is also used to access the same value in the `userInfo` dictionary of the notification named `UIApplicationDidFinishLaunchingNotification` (page 136).

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

`UIApplicationLaunchOptionsAnnotationKey`

You use this key to access contents of the dictionary passed in the second parameter of the `UIApplicationDelegate` method `application:didFinishLaunchingWithOptions:` (page 839). It returns an object representing the annotation property list.

Available in iOS 3.2 and later.

Declared in `UIApplication.h`.

`UIApplicationLaunchOptionsLocalNotificationKey`

You use this key to access contents of the dictionary passed in the second parameter of the `UIApplicationDelegate` method `application:didFinishLaunchingWithOptions:` (page 839). It returns the `UILocalNotification` object associated with the local notification just presented to the application.

See the description of `application:didReceiveLocalNotification:` (page 840) for further information. This key is also used to access the same value in the `userInfo` dictionary of the notification named `UIApplicationDidFinishLaunchingNotification` (page 136).

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

`UIApplicationLaunchOptionsLocationKey`

The presence of this key indicates that the application was launched in response to an incoming location event. You should use this as a signal to create and configure a new `CLLocationManager` object and start location services again. Upon doing so, your delegate receives the corresponding location data.

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

UserInfo Dictionary Keys

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

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

Constants

`UIApplicationStatusBarOrientationUserInfoKey`

Accesses an `NSNumber` object that encapsulates a `UIInterfaceOrientation` value indicating the current orientation (see UIInterfaceOrientation (page 128)). This key is used with `UIApplicationDidChangeStatusBarOrientationNotification` (page 135) and `UIApplicationWillChangeStatusBarOrientationNotification` (page 137) notifications.

Available in iOS 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 135) and `UIApplicationWillChangeStatusBarFrameNotification` (page 137) notifications.

Available in iOS 2.0 and later.

Declared in `UIApplication.h`.

Declared In

`UIApplication.h`

UIRemoteNotificationType

Constants indicating the types of notifications the application accepts.

```
typedef enum {
    UIRemoteNotificationTypeNone    = 0,
    UIRemoteNotificationTypeBadge   = 1 << 0,
    UIRemoteNotificationTypeSound   = 1 << 1,
    UIRemoteNotificationTypeAlert   = 1 << 2
} UIRemoteNotificationType;
```

Constants

`UIRemoteNotificationTypeNone`

The application accepts no notifications.

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

`UIRemoteNotificationTypeBadge`

The application accepts notifications that badge the application icon.

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

`UIRemoteNotificationTypeSound`

The application accepts alert sounds as notifications.

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

`UIRemoteNotificationTypeAlert`

The application accepts alert messages as notifications.

Available in iOS 3.0 and later.

Declared in `UIApplication.h`.

Discussion

One or more of the values in the `UIRemoteNotificationType` bit mask are passed to iOS as the argument of the `registerForRemoteNotificationTypes:` (page 122) method. Thereafter, iOS filters notifications for the application based on these values. You can always get the current notification types by calling the `enabledRemoteNotificationTypes` (page 119) method.

Availability

Available in iOS 3.0 and later.

Declared In

`UIApplication.h`

UIBackgroundTaskIdentifier

A unique token that identifies a request to run in the background.

```
typedef NSUInteger UIBackgroundTaskIdentifier;
```

Availability

Available in iOS 4.0 and later.

Declared In

`UIApplication.h`

Background Task Constants

Constants used when running in the background.

```
const UIBackgroundTaskIdentifier UIBackgroundTaskInvalid;
const NSTimeInterval UIMinimumKeepAliveTimeout;
```

Constants

`UIBackgroundTaskInvalid`

An token indicating an invalid task request. This constant should be used to initialize variables or to check for errors.

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

`UIMinimumKeepAliveTimeout`

The minimum amount of time (measured in seconds) an application may run a critical background task in the background.

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

UIApplicationState

The running states of an application

```
typedef enum {
    UIApplicationStateActive,
    UIApplicationStateInactive,
    UIApplicationStateBackground
} UIApplicationState;
```

Constants

`UIApplicationStateActive`

The application is running in the foreground and currently receiving events.

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

`UIApplicationStateInactive`

The application is running in the foreground but is not receiving events. This might happen as a result of an interruption or because the application is transitioning to or from the background.

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

`UIApplicationStateBackground`

The application is running in the background.

Available in iOS 4.0 and later.

Declared in `UIApplication.h`.

Notifications

All `UIApplication` notifications are posted by the application instance returned by `sharedApplication` (page 115).

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 iOS 2.0 and later.

Declared In
`UIApplication.h`

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 132) to access this value.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationDidChangeStatusBarOrientationNotification

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 128)). Use `UIApplicationStatusBarOrientationUserInfoKey` (page 132) to access this value

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationDidEnterBackgroundNotification

Posted when the application enters the background.

The `object` of the notification is the `UIApplication` object. There is no `userInfo` dictionary.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

UIApplicationDidFinishLaunchingNotification

Posted immediately after the application finishes launching.

If the application was launched as a result of in remote notification targeted at it or because another application opened a URL resource claimed the posting application (the notification `object`), this notification contains a `userInfo` dictionary. You can access the contents of the dictionary using the `UIApplicationLaunchOptionsURLKey` (page 131) and `UIApplicationLaunchOptionsSourceApplicationKey` (page 131) constants (for URLs), the `UIApplicationLaunchOptionsRemoteNotificationKey` (page 131) constant (for remote notifications), and the `UIApplicationLaunchOptionsLocalNotificationKey` (page 132) constant (for local notifications). If the notification was posted for a normal application launch, there is no `userInfo` dictionary.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationProtectedDataDidBecomeAvailable

Posted when the protected files become available for your code to access.

This notification does not contain a `userInfo` dictionary.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

UIApplicationProtectedDataWillBecomeUnavailable

Posted shortly before protected files are locked down and become inaccessible.

Upon receiving this notification, clients should release any references to protected files. This notification does not contain a `userInfo` dictionary.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationWillChangeStatusBarOrientationNotification

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 128)). Use `UIApplicationStatusBarOrientationUserInfoKey` (page 132) to access this value.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

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 132) to access this value.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

UIApplicationWillEnterForegroundNotification

Posted shortly before an application leaves the background state on its way to becoming the active application.

The `object` of the notification is the `UIApplication` object. There is no `userInfo` dictionary.

Availability
Available in iOS 4.0 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in	UIBarButtonItem.h
Related sample code	AddMusic BonjourWeb MultipleDetailViews SpeakHere ToolbarSearch

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 142)
Creates and returns a new item containing the specified system item.

`- initWithCustomView:` (page 143)
Creates and returns a new item using the specified custom view.

`- initWithImage:style:target:action:` (page 143)
Creates and returns a new item using the specified image and other properties.

`- initWithTitle:style:target:action:` (page 144)
Creates and returns a new item using the specified title and other properties.

Getting and Setting Properties

`target` (page 141) *property*
The object that receives an action when the item is selected.

`action` (page 140) *property*
The selector defining the action message to send to the target object when the user taps this bar button item.

`style` (page 141) *property*
The style of the item.

`possibleTitles` (page 141) *property*
Collection of possible titles to display on the bar.

`width` (page 141) *property*
The width of the item.

`customView` (page 140) *property*
A custom view representing the item.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

See Also
`@property target` (page 141)

Declared In
`UIBarButtonItem.h`

customView

A custom view representing the item.

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

Availability
Available in iOS 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 iOS 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 148). The default value is `UIBarButtonItemStylePlain` (page 148).

Availability
Available in iOS 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`, the action message is passed up the responder chain where it may be handled by any object implementing a method corresponding to the selector held by the `action` (page 140) property. The default value is `nil`.

Availability
Available in iOS 2.0 and later.

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 iOS 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 144).

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 iOS 2.0 and later.

See Also
`- initWithImage:style:target:action:` (page 143)
`- initWithTitle:style:target:action:` (page 144)

Related Sample Code
AddMusic
BonjourWeb
ToolbarSearch
WiTap

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 iOS 2.0 and later.

Related Sample Code

AddMusic

ToolbarSearch

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 148).

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 iOS 2.0 and later.

See Also

- `initWithBarButtonSystemItem:target:action:` (page 142)
- `initWithTitle:style:target:action:` (page 144)

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 148).

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 iOS 2.0 and later.

See Also
- initWithBarButtonSystemItem:target:action: (page 142)
- initWithImage:style:target:action: (page 143)

Related Sample Code
GKRocket
ToolbarSearch

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,
    UIBarButtonSystemItemUndo,        // iOS 3.0
    UIBarButtonSystemItemRedo,        // iOS 3.0
    UIBarButtonSystemItemPageCurl,    // iOS 4.0
} UIBarButtonSystemItem;
```

Constants

`UIBarButtonSystemItemDone`

The system Done button. Localized.

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemCancel`

The system Cancel button. Localized.

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemEdit`

The system Edit button. Localized.

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemSave`

The system Save button. Localized.

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemAdd`

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

Available in iOS 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 iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemFixedSpace`

Blank space to add between other items. Only the `width` (page 141) property is used when this value is set.

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemCompose`

The system compose button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemReply`

The system reply button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemAction`

The system action button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemOrganize`

The system organize button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemBookmarks`

The system bookmarks button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemSearch`

The system search button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemRefresh`

The system refresh button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemStop`

The system stop button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemCamera`

The system camera button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemTrash`

The system trash button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemPlay`

The system play button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemPause`

The system pause button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemRewind`

The system rewind button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemFastForward`

The system fast forward button. 

Available in iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemUndo`

The system undo button. 

Available in iOS 3.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemRedo`

The system redo button. 

Available in iOS 3.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonSystemItemPageCurl`
The system page curl button.

Available in iOS 4.0 and later.

Declared in `UIBarButtonItem.h`.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

`UIBarButtonItemStyleBordered`
A simple button style with a border.

Available in iOS 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 iOS 2.0 and later.

Declared in `UIBarButtonItem.h`.

Availability
Available in iOS 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 iOS 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 150) *property*
A Boolean value indicating whether the item is enabled.

`image` (page 150) *property*
The image used to represent the item.

`imageInsets` (page 150) *property*
The image inset or outset for each edge.

`title` (page 151) *property*
The title displayed on the item.

`tag` (page 151) *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 Programming Language*.

enabled

A Boolean value indicating whether the item is enabled.

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

Discussion
If `NO`, the item is drawn partially dimmed to indicate it is disabled. The default value is `YES`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AddMusic
SpeakHere
ToolbarSearch

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 iOS 2.0 and later.

See Also
`@property imageInsets` (page 150)

Declared In
`UIBarItem.h`

imageInsets

The image inset or outset for each edge.

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

Discussion
The default value is `UIEdgeInsetsZero` (page 1017).

Availability
Available in iOS 2.0 and later.

See Also
`@property image` (page 150)

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 iOS 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 iOS 2.0 and later.

Related Sample Code
MultipleDetailViews
SpeakHere

Declared In
`UIBarItem.h`

UIBezierPath Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIBezierPath.h
Companion guide	iPad Programming Guide

Overview

The `UIBezierPath` class lets you define a path consisting of straight and curved line segments and render that path in your custom views. You use this class initially to specify just the geometry for your path. Paths can define simple shapes such as rectangles, ovals, and arcs or they can define complex polygons that incorporate a mixture of straight and curved line segments. After defining the shape, you can use additional methods of this class to render the path in the current drawing context.

A `UIBezierPath` object combines the geometry of a path with attributes that describe the path during rendering. You set the geometry and attributes separately and can change them independent of one another. Once you have the object configured the way you want it, you can tell it to draw itself in the current context. Because the creation, configuration, and rendering process are all distinct steps, Bezier path objects can be reused easily in your code. You can even use the same object to render the same shape multiple times, perhaps changing the rendering options between successive drawing calls.

You set the geometry of a path by manipulating the path's current point. When you create a new empty path object, the current point is undefined and must be set explicitly. To move the current point without drawing a segment, you use the `moveToPoint:` (page 171) method. All other methods result in the addition of either a line or curve segments to the path. The methods for adding new segments always assume you are starting at the current point and ending at some new point that you specify. After adding the segment, the end point of the new segment automatically becomes the current point.

A single Bezier path object can contain any number of open or closed subpaths, where each subpath represents a connected series of path segments. Calling the `closePath` (page 168) method closes a subpath by adding a straight line segment from the current point to the first point in the subpath. Calling the `moveToPoint:` method ends the current subpath (without closing it) and sets the starting point of the next subpath. The subpaths of a Bezier path object share the same drawing attributes and must be manipulated as a group. To draw subpaths with different attributes, you must put each subpath in its own `UIBezierPath` object.

After configuring the geometry and attributes of a Bezier path, you draw the path in the current graphics context using the `stroke` (page 172) and `fill` (page 169) methods. The `stroke` method traces the outline of the path using the current stroke color and the attributes of the Bezier path object. Similarly, the `fill` method fills in the area enclosed by the path using the current fill color. (You set the stroke and fill color using the `UIColor` class.)

In addition to using a Bezier path object to draw shapes, you can also use it to define a new clipping region. The `addClip` (page 164) method intersects the shape represented by the path object with the current clipping region of the graphics context. During subsequent drawing, only content that lies within the new intersection region is actually rendered to the graphics context.

Tasks

Creating a UIBezierPath Object

`+ bezierPath` (page 160)
Creates and returns a new `UIBezierPath` object.

`+ bezierPathWithRect:` (page 162)
Creates and returns a new `UIBezierPath` object initialized with a rectangular path.

`+ bezierPathWithOvalInRect:` (page 162)
Creates and returns a new `UIBezierPath` object initialized with an oval path inscribed in the specified rectangle

`+ bezierPathWithRoundedRect:cornerRadius:` (page 163)
Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

`+ bezierPathWithRoundedRect:byRoundingCorners:cornerRadii:` (page 163)
Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

`+ bezierPathWithArcCenter:radius:startAngle:endAngle:clockwise:` (page 160)
Creates and returns a new `UIBezierPath` object initialized with an arc of a circle.

`+ bezierPathWithCGPath:` (page 161)
Creates and returns a new `UIBezierPath` object initialized with the contents of a Core Graphics path.

Constructing a Path

`- moveToPoint:` (page 171)
Moves the receiver's current point to the specified location.

`- addLineToPoint:` (page 166)
Appends a straight line to the receiver's path.

`- addArcWithCenter:radius:startAngle:endAngle:clockwise:` (page 164)
Appends an arc to the receiver's path.

`- addCurveToPoint:controlPoint1:controlPoint2:` (page 165)
Appends a cubic Bézier curve to the receiver's path.

`- addQuadCurveToPoint:controlPoint:` (page 167)
Appends a quadratic Bézier curve to the receiver's path.

- `closePath` (page 168)
  Closes the most recently added subpath.
- `removeAllPoints` (page 171)
  Removes all points from the receiver, effectively deleting all subpaths.
- `appendPath:` (page 167)
  Appends the contents of the specified path object to the receiver's path.

`CGPath` (page 156) *property*
The Core Graphics representation of the path.

`currentPoint` (page 157) *property*
The current point in the graphics path. (read-only)

Accessing Drawing Properties

`lineWidth` (page 159) *property*
The line width of the path.

`lineCapStyle` (page 158) *property*
The shape of the paths end points when stroked.

`lineJoinStyle` (page 158) *property*
The shape of the joints between connected segments of a stroked path.

`miterLimit` (page 159) *property*
The limiting value that helps avoid spikes at junctions between connected line segments.

`flatness` (page 158) *property*
The factor that determines the rendering accuracy for curved path segments.

`usesEvenOddFillRule` (page 159) *property*
A Boolean indicating whether the even-odd winding rule is in use for drawing paths.

- `setLineDash:count:phase:` (page 171)
  Sets the line-stroking pattern for the path.
- `getLineDash:count:phase:` (page 170)
  Retrieves the line-stroking pattern for the path.

Drawing Paths

- `fill` (page 169)
  Paints the region enclosed by the receiver's path using the current drawing properties.
- `fillWithBlendMode:alpha:` (page 170)
  Paints the region enclosed by the receiver's path using the specified blend mode and transparency values.
- `stroke` (page 172)
  Draws a line along the receiver's path using the current drawing properties.
- `strokeWithBlendMode:alpha:` (page 172)
  Draws a line along the receiver's path using the specified blend mode and transparency values.

Clipping Paths

`- addClip` (page 164)
Intersects the area enclosed by the receiver's path with the clipping path of the current graphics context and makes the resulting shape the current clipping path.

Hit Detection

`- containsPoint:` (page 169)
Returns a Boolean value indicating whether the area enclosed by the receiver contains the specified point.

`empty` (page 157) *property*
A Boolean value indicating whether the path has any valid elements. (read-only)

`bounds` (page 156) *property*
The bounding rectangle of the path. (read-only)

Applying Transformations

`- applyTransform:` (page 168)
Transforms all points in the path using the specified affine transform matrix.

Properties

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

bounds

The bounding rectangle of the path. (read-only)

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

Discussion
The value in this property represents the smallest rectangle that completely encloses all points in the path, including any control points for Bézier and quadratic curves.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

CGPath

The Core Graphics representation of the path.

```
@property(nonatomic) CGPathRef CGPath
```

Discussion
This property contains a snapshot of the path at any given point in time. Getting this property returns an immutable path object that you can pass to Core Graphics functions. The path object itself is owned by the `UIBezierPath` object and is valid only until you make further modifications to the path.

You can set the value of this property to a path you built using the functions of the Core Graphics framework. When setting a new path, this method makes a copy of the path you provide.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

currentPoint

The current point in the graphics path. (read-only)

```
@property(nonatomic, readonly) CGPoint currentPoint
```

Discussion
The value in this property represents the starting point for new line and curve segments. If the path is currently empty, this property contains the value `CGPointZero`.

Availability
Available in iOS 3.2 and later.

See Also
`@property empty` (page 157)

Declared In
`UIBezierPath.h`

empty

A Boolean value indicating whether the path has any valid elements. (read-only)

```
@property(readonly, getter=isEmpty) BOOL empty
```

Discussion
Valid path elements include commands to move to a specified point, draw a line or curve segment, or close the path. Thus, a path is not considered empty even if all you do is call the `moveToPoint:` (page 171) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

flatness

The factor that determines the rendering accuracy for curved path segments.

```
@property(nonatomic) CGFloat flatness
```

Discussion
The flatness value measures the largest permissible distance (measured in pixels) between a point on the true curve and a point on the rendered curve. Smaller values result in smoother curves but require more computation time. Larger values result in more jagged curves but are rendered much faster. The default flatness value is `0.6`.

In most cases, you should not change the flatness value. However, you might increase the flatness value temporarily to minimize the amount of time it takes to draw a shape temporarily (such as during scrolling).

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

lineCapStyle

The shape of the paths end points when stroked.

```
@property(nonatomic) CGLineCap lineCapStyle
```

Discussion
The line cap style is applied to the start and end points of any open subpaths. This property does not affect closed subpaths. The default line cap style is `kCGLineCapButt`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

lineJoinStyle

The shape of the joints between connected segments of a stroked path.

```
@property(nonatomic) CGLineJoin lineJoinStyle
```

Discussion
The default line join style is `kCGLineJoinMiter`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

lineWidth

The line width of the path.

```
@property(nonatomic) CGFloat lineWidth
```

Discussion
The line width defines the thickness of the receiver's stroked path. A width of 0 is interpreted as the thinnest line that can be rendered on a particular device. The actual rendered line width may vary from the specified width by as much as 2 device pixels, depending on the position of the line with respect to the pixel grid and the current anti-aliasing settings. The width of the line may also be affected by scaling factors specified in the current transformation matrix of the active graphics context.

The default line width is 1.0.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

miterLimit

The limiting value that helps avoid spikes at junctions between connected line segments.

```
@property(nonatomic) CGFloat miterLimit
```

Discussion
The miter limit helps you avoid spikes in paths that use the `kCGLineJoinMiter` join style. If the ratio of the miter length—that is, the diagonal length of the miter join—to the line thickness exceeds the miter limit, the joint is converted to a bevel join. The default miter limit is 10, which results in the conversion of miters whose angle at the joint is less than 11 degrees.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

usesEvenOddFillRule

A Boolean indicating whether the even-odd winding rule is in use for drawing paths.

```
@property(nonatomic) BOOL usesEvenOddFillRule
```

Discussion
If `YES`, the path is filled using the even-odd rule. If `NO`, it is filled using the non-zero rule. Both rules are algorithms to determine which areas of a path to fill with the current fill color. A ray is drawn from a point inside a given region to a point anywhere outside the path's bounds. The total number of crossed path lines (including implicit path lines) and the direction of each path line are then interpreted as follows:

- For the even-odd rule, if the total number of path crossings is odd, the point is considered to be inside the path and the corresponding region is filled. If the number of crossings is even, the point is considered to be outside the path and the region is not filled.

- For the non-zero rule, the crossing of a left-to-right path counts as +1 and the crossing of a right-to-left path counts as -1. If the sum of the crossings is nonzero, the point is considered to be inside the path and the corresponding region is filled. If the sum is 0, the point is outside the path and the region is not filled.

The default value of this property is `NO`. For more information about winding rules and how they are applied to subpaths, see *Quartz 2D Programming Guide*.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

Class Methods

bezierPath

Creates and returns a new `UIBezierPath` object.

```
+ (UIBezierPath *)bezierPath
```

Return Value
A new empty path object.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithArcCenter:radius:startAngle:endAngle:clockwise:

Creates and returns a new `UIBezierPath` object initialized with an arc of a circle.

```
+ (UIBezierPath *)bezierPathWithArcCenter:(CGPoint)center radius:(CGFloat)radius
    startAngle:(CGFloat)startAngle endAngle:(CGFloat)endAngle
    clockwise:(BOOL)clockwise
```

Parameters

center
Specifies the center point of the circle (in the current coordinate system) used to define the arc.

radius
Specifies the radius of the circle used to define the arc.

startAngle
Specifies the starting angle of the arc (measured in radians).

endAngle
Specifies the end angle of the arc (measured in radians).

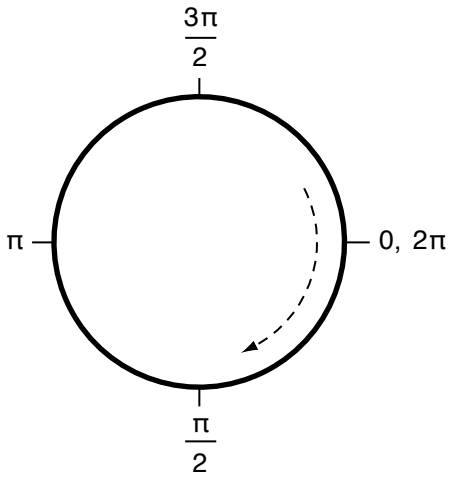
clockwise

The direction in which to draw the arc.

Return Value
A new path object with the specified arc.

Discussion
This method creates an open subpath. The created arc lies on the perimeter of the specified circle. When drawn in the default coordinate system, the start and end angles are based on the unit circle shown in Figure 16-1. For example, specifying a start angle of `0` radians, an end angle of `π` radians, and setting the `clockwise` parameter to `YES` draws the bottom half of the circle. However, specifying the same start and end angles but setting the *`clockwise`* parameter set to `NO` draws the top half of the circle.

Figure 16-1 Angles in the default coordinate system



After calling this method, the current point is set to the point on the arc at the end angle of the circle.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithCGPath:

Creates and returns a new `UIBezierPath` object initialized with the contents of a Core Graphics path.

```
+ (UIBezierPath *)bezierPathWithCGPath:(CGPathRef)CGPath
```

Parameters

`CGPath`

The Core Graphics path from which to obtain the initial path information. If this parameter is `nil`, the method raises an exception.

Return Value
A new path object with the specified path information.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithOvalInRect:

Creates and returns a new `UIBezierPath` object initialized with an oval path inscribed in the specified rectangle

```
+ (UIBezierPath *)bezierPathWithOvalInRect:(CGRect)rect
```

Parameters

rect

The rectangle in which to inscribe an oval.

Return Value
A new path object with the oval path.

Discussion
This method creates a closed subpath that approximates the oval using a sequence of Bézier curves. The path is created in a clockwise direction (relative to the default coordinate system). If the *rect* parameter specifies a square, the inscribed path is a circle.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithRect:

Creates and returns a new `UIBezierPath` object initialized with a rectangular path.

```
+ (UIBezierPath *)bezierPathWithRect:(CGRect)rect
```

Parameters

rect

The rectangle describing the path to create.

Return Value
A new path object with the rectangular path.

Discussion
This method creates a closed subpath by starting at the origin of *rect* and adding line segments in a clockwise direction (relative to the default coordinate system).

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithRoundedRect:byRoundingCorners:cornerRadii:

Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

```
+ (UIBezierPath *)bezierPathWithRoundedRect:(CGRect)rect
    byRoundingCorners:(UIRectCorner)corners cornerRadii:(CGSize)cornerRadii
```

Parameters

rect

The rectangle that defines the basic shape of the path.

corners

A bitmask value that identifies the corners that you want rounded. You can use this parameter to round only a subset of the corners of the rectangle.

cornerRadii

The radius of each corner oval. Values larger than half the rectangle's width or height are clamped appropriately to half the width or height.

Return Value
A new path object with the rounded rectangular path.

Discussion
This method creates a closed subpath, proceeding in a clockwise direction (relative to the default coordinate system) as it creates the necessary line and curve segments.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithRoundedRect:cornerRadius:

Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

```
+ (UIBezierPath *)bezierPathWithRoundedRect:(CGRect)rect
    cornerRadius:(CGFloat)cornerRadius
```

Parameters

rect

The rectangle that defines the basic shape of the path

cornerRadius

The radius of each corner oval. A value of `0` results in a rectangle without rounded corners. Values larger than half the rectangle's width or height are clamped appropriately to half the width or height.

Return Value
A new path object with the rounded rectangular path.

Discussion
This method creates a closed subpath, proceeding in a clockwise direction (relative to the default coordinate system) as it creates the necessary line and curve segments.

Availability
Available in iOS 3.2 and later.

Declared In
UIBezierPath.h

Instance Methods

addArcWithCenter:radius:startAngle:endAngle:clockwise:

Appends an arc to the receiver's path.

```
- (void)addArcWithCenter:(CGPoint)center radius:(CGFloat)radius
    startAngle:(CGFloat)startAngle endAngle:(CGFloat)endAngle
    clockwise:(BOOL)clockwise
```

Parameters

center
Specifies the center point of the circle (in the current coordinate system) used to define the arc.

radius
Specifies the radius of the circle used to define the arc.

startAngle
Specifies the starting angle of the arc (measured in radians).

endAngle
Specifies the end angle of the arc (measured in radians).

clockwise
The direction in which to draw the arc.

Discussion
This method adds the specified arc beginning at the current point. The created arc lies on the perimeter of the specified circle. When drawn in the default coordinate system, the start and end angles are based on the unit circle shown in Figure 16-1 (page 161). For example, specifying a start angle of `0` radians, an end angle of π radians, and setting the `clockwise` parameter to `YES` draws the bottom half of the circle. However, specifying the same start and end angles but setting the *`clockwise`* parameter set to `NO` draws the top half of the circle.

After calling this method, the current point is set to the point on the arc at the end angle of the circle.

Availability
Available in iOS 4.0 and later.

Declared In
UIBezierPath.h

addClip

Intersects the area enclosed by the receiver's path with the clipping path of the current graphics context and makes the resulting shape the current clipping path.

```
- (void)addClip
```

Discussion
This method modifies the visible drawing area of the current graphics context. After calling it, subsequent drawing operations result in rendered content only if they occur within the fill area of the specified path.

Important: If you need to remove the clipping region to perform subsequent drawing operations, you must save the current graphics state (using the `CGContextSaveGState` function) before calling this method. When you no longer need the clipping region, you can then restore the previous drawing properties and clipping region using the `CGContextRestoreGState` function.

The `usesEvenOddFillRule` (page 159) property is used to determine whether the even-odd or non-zero rule is used to determine the area enclosed by the path.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

addCurveToPoint:controlPoint1:controlPoint2:

Appends a cubic Bézier curve to the receiver's path.

```
- (void)addCurveToPoint:(CGPoint)endPoint controlPoint1:(CGPoint)controlPoint1
    controlPoint2:(CGPoint)controlPoint2
```

Parameters

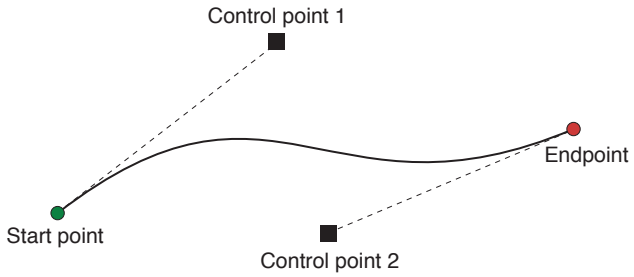
endPoint
The end point of the curve.

controlPoint1
The first control point to use when computing the curve.

controlPoint2
The second control point to use when computing the curve.

Discussion
This method appends a cubic Bézier curve from the current point to the end point specified by the *endPoint* parameter. The two control points define the curvature of the segment. Figure 16-2 shows an approximation of a cubic Bézier curve given a set of initial points. The exact curvature of the segment involves a complex mathematical relationship between all of the points and is well documented online.

Figure 16-2 A cubic Bézier curve



You must set the path's current point (using the `moveToPoint:` (page 171) method or through the previous creation of a line or curve segment) before you call this method. If the path is empty, this method does nothing. After adding the curve segment, this method updates the current point to the value in *point*.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

addLineToPoint:

Appends a straight line to the receiver's path.

```
- (void)addLineToPoint:(CGPoint)point
```

Parameters

point

The destination point of the line segment, specified in the current coordinate system.

Discussion
This method creates a straight line segment starting at the current point and ending at the point specified by the *point* parameter. After adding the line segment, this method updates the current point to the value in *point*.

You must set the path's current point (using the `moveToPoint:` (page 171) method or through the previous creation of a line or curve segment) before you call this method. If the path is empty, this method does nothing.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

addQuadCurveToPoint:controlPoint:

Appends a quadratic Bézier curve to the receiver's path.

```
- (void)addQuadCurveToPoint:(CGPoint)endPoint controlPoint:(CGPoint)controlPoint
```

Parameters

endPoint

The end point of the curve.

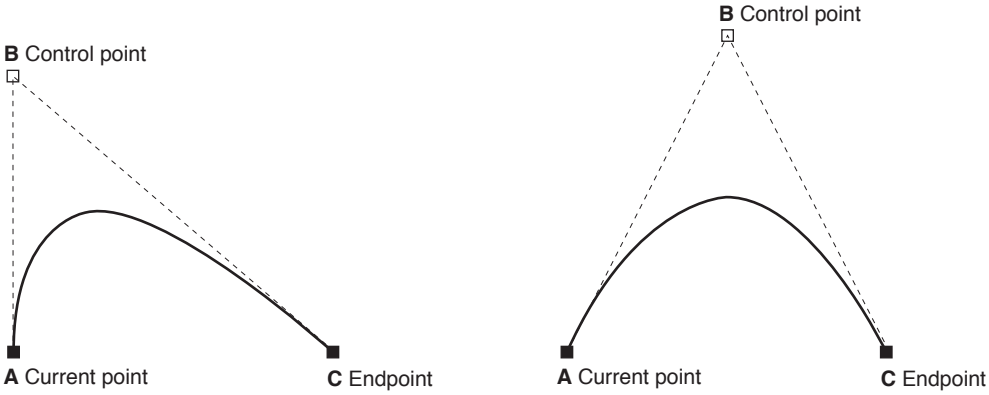
controlPoint

The control point of the curve.

Discussion

This method appends a quadratic Bézier curve from the current point to the end point specified by the *endPoint* parameter. The relationships between the current point, control point, and end point are what defines the actual curve. Figure 16-3 shows some examples of quadratic curves and the approximate curve shape based on some sample points. The exact curvature of the segment involves a complex mathematical relationship between the points and is well documented online.

Figure 16-3 Quadratic curve examples



You must set the path's current point (using the `moveToPoint:` (page 171) method or through the previous creation of a line or curve segment) before you call this method. If the path is empty, this method does nothing. After adding the curve segment, this method updates the current point to the value in *point*.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

appendPath:

Appends the contents of the specified path object to the receiver's path.

```
- (void)appendPath:(UIBezierPath *)bezierPath
```

Parameters

bezierPath

The path to add to the receiver.

Discussion

This method adds the commands used to create the path in *bezierPath* to the end of the receiver's path. This method does not explicitly try to connect the subpaths in the two objects, although the operations in *bezierPath* might still cause that effect.

Availability

Available in iOS 3.2 and later.

Declared In

UIBezierPath.h

applyTransform:

Transforms all points in the path using the specified affine transform matrix.

```
- (void)applyTransform:(CGAffineTransform)transform
```

Parameters

transform

The transform matrix to apply to the path.

Discussion

This method applies the specified transform to the path's points immediately. The modifications made to the path object are permanent. If you do not want to permanently modify a path object, you should consider applying the transform to a copy.

Availability

Available in iOS 3.2 and later.

Declared In

UIBezierPath.h

closePath

Closes the most recently added subpath.

```
- (void)closePath
```

Discussion

This method closes the current subpath by creating a line segment between the first and last points in the subpath. This method subsequently updates the current point to the end of the newly created line segment, which is also the first point in the now closed subpath.

Availability

Available in iOS 3.2 and later.

Declared In

UIBezierPath.h

containsPoint:

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

```
- (BOOL)containsPoint:(CGPoint)point
```

Parameters

point

The point to test against the path, specified in the path object's coordinate system.

Return Value

`YES` if the point is considered to be within the path's enclosed area or `NO` if it is not.

Discussion

The receiver contains the specified point if that point is in a portion of a closed subpath that would normally be painted during a fill operation. This method uses the value of the `usesEvenOddFillRule` (page 159) property to determine which parts of the subpath would be filled.

A point is not considered to be enclosed by the path if it is inside an open subpath, regardless of whether that area would be painted during a fill operation. Therefore, to determine mouse hits on open paths, you must create a copy of the path object and explicitly close any subpaths (using the `closePath` (page 168) method) before calling this method.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

fill

Paints the region enclosed by the receiver's path using the current drawing properties.

```
- (void)fill
```

Discussion

This method fills the path using the current fill color and drawing properties. If the path contains any open subpaths, this method implicitly closes them before painting the fill region.

The painted region includes the pixels right up to, but not including, the path line itself. For paths with large line widths, this can result in overlap between the fill region and the stroked path (which is itself centered on the path line).

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

fillWithBlendMode:alpha:

Paints the region enclosed by the receiver's path using the specified blend mode and transparency values.

```
- (void)fillWithBlendMode:(CGBlendMode)blendMode alpha:(CGFloat)alpha
```

Parameters

blendMode

The blend mode determines how the filled path is composited with any existing rendered content.

alpha

The amount of transparency to apply to the filled path. Values can range between `0.0` (transparent) and `1.0` (opaque). Values outside this range are clamped to `0.0` or `1.0`.

Discussion

This method fills the path using the current fill color and drawing properties (plus the specified blend mode and transparency value). If the path contains any open subpaths, this method implicitly closes them before painting the fill region.

The painted region includes the pixels right up to, but not including, the path line itself. For paths with large line widths, this can result in overlap between the fill region and the stroked path (which is itself centered on the path line).

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

getLineDash:count:phase:

Retrieves the line-stroking pattern for the path.

```
- (void)getLineDash:(CGFloat *)pattern count:(NSInteger *)count phase:(CGFloat
    *)phase
```

Parameters

pattern

On input, a C-style array of floating point values, or `nil` if you do not want the pattern values. On output, this array contains the lengths (measured in points) of the line segments and gaps in the pattern. The values in the array alternate, starting with the first line segment length, followed by the first gap length, followed by the second line segment length, and so on.

count

On input, a pointer to an integer or `nil` if you do not want the number of pattern entries. On output, the number of entries written to *pattern*.

phase

On input, a pointer to a floating point value or `nil` if you do not want the phase. On output, this value contains the offset at which to start drawing the pattern, measured in points along the dashed-line pattern. For example, a phase of 6 in the pattern 5-2-3-2 would cause drawing to begin in the middle of the first gap.

Discussion
The array in the *pattern* parameter must be large enough to hold all of the returned values in the pattern. If you are not sure how many values there might be, you can call this method twice. The first time you call it, do not pass a value for *pattern* but use the returned value in the *count* parameter to allocate an array of floating-point numbers that you can then pass in the second time.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

moveToPoint:

Moves the receiver's current point to the specified location.

```
- (void)moveToPoint:(CGPoint)point
```

Parameters

point

A point in the current coordinate system.

Discussion
This method implicitly ends the current subpath (if any) and sets the current point to the value in the *point* parameter. When ending the previous subpath, this method does not actually close the subpath. Therefore, the first and last points of the previous subpath are not connected to each other.

For many path operations, you must call this method before issuing any commands that cause a line or curve segment to be drawn.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

removeAllPoints

Removes all points from the receiver, effectively deleting all subpaths.

```
- (void)removeAllPoints
```

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

setLineDash:count:phase:

Sets the line-stroking pattern for the path.

```
- (void)setLineDash:(const CGFloat *)pattern count:(NSInteger)count
    phase:(CGFloat)phase
```

Parameters

pattern

A C-style array of floating point values that contains the lengths (measured in points) of the line segments and gaps in the pattern. The values in the array alternate, starting with the first line segment length, followed by the first gap length, followed by the second line segment length, and so on.

count

The number of values in *pattern*.

phase

The offset at which to start drawing the pattern, measured in points along the dashed-line pattern. For example, a phase value of `6` for the pattern 5-2-3-2 would cause drawing to begin in the middle of the first gap.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

stroke

Draws a line along the receiver's path using the current drawing properties.

```
- (void)stroke
```

Discussion
The drawn line is centered on the path with its sides parallel to the path segment. This method applies the current drawing properties to the rendered path.

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

strokeWithBlendMode:alpha:

Draws a line along the receiver's path using the specified blend mode and transparency values.

```
- (void)strokeWithBlendMode:(CGBlendMode)blendMode alpha:(CGFloat)alpha
```

Parameters

blendMode

The blend mode determines how the stroked path is composited with any existing rendered content.

alpha

The amount of transparency to apply to the stroked path. Values can range between `0.0` (transparent) and `1.0` (opaque). Values outside this range are clamped to `0.0` or `1.0`.

Discussion
The drawn line is centered on the path with its sides parallel to the path segment. This method applies the current stroke color and drawing properties (plus the specified blend mode and transparency value) to the rendered path.

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

Constants

UIRectCorner

The corners of a rectangle.

```
enum {
    UIRectCornerTopLeft     = 1 << 0,
    UIRectCornerTopRight    = 1 << 1,
    UIRectCornerBottomLeft  = 1 << 2,
    UIRectCornerBottomRight = 1 << 3,
    UIRectCornerAllCorners  = ~0
};
typedef NSUInteger UIRectCorner;
```

Constants

`UIRectCornerTopLeft`
The top-left corner of the rectangle.
Available in iOS 3.2 and later.
Declared in `UIBezierPath.h`.

`UIRectCornerTopRight`
The top-right corner of the rectangle.
Available in iOS 3.2 and later.
Declared in `UIBezierPath.h`.

`UIRectCornerBottomLeft`
The bottom-left corner of the rectangle.
Available in iOS 3.2 and later.
Declared in `UIBezierPath.h`.

`UIRectCornerBottomRight`
The bottom-right corner of the rectangle.
Available in iOS 3.2 and later.
Declared in `UIBezierPath.h`.

`UIRectCornerAllCorners`

All corners of the rectangle.

Available in iOS 3.2 and later.

Declared in `UIBezierPath.h`.

Discussion

The specified constants reflect the corners of a rectangle that has not been modified by an affine transform and is drawn in the default coordinate system (where the origin is in the upper-left corner and positive values extend down and to the right).

UIButton Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	
Related sample code	AddMusic CryptoExercise

Overview

An instance of the `UIButton` class implements a button on the touch screen. A button intercepts touch events and sends an action message to a target object when 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 these accessors, you can specify a different appearance for each button state.

Tasks

Creating Buttons

`+ buttonWithType:` (page 184)
Creates and returns a new button of the specified type.

Configuring Button Title

`buttonType` (page 178) *property*
The button type. (read-only)

`font` (page 181) *property*
The font used to display text on the button. (Deprecated. Use the `font` property of the `titleLabel` (page 183) instead.)

`lineBreakMode` (page 182) *property*
The line break mode to use when drawing text. (Deprecated. Use the `lineBreakMode` property of the `titleLabel` (page 183) instead.)

`titleShadowOffset` (page 184) *property*
The offset of the shadow used to display the receiver's title. (Deprecated. Use the `shadowOffset` property of the `titleLabel` (page 183) instead.)

`titleLabel` (page 183) *property*
A view that displays the value of the `currentTitle` property for a button. (read-only)

`reversesTitleShadowWhenHighlighted` (page 182) *property*
A Boolean value that determines whether the title shadow changes when the button is highlighted.

- `setTitle:forState:` (page 188)
Sets the title to use for the specified state.

- `setTitleColor:forState:` (page 189)
Sets the color of the title to use for the specified state.

- `setTitleShadowColor:forState:` (page 189)
Sets the color of the title shadow to use for the specified state.

- `titleColorForState:` (page 190)
Returns the title color used for a state.

- `titleForState:` (page 190)
Returns the title used for a state.

- `titleShadowColorForState:` (page 191)
Returns the shadow color of the title used for a state.

Configuring Button Images

`adjustsImageWhenHighlighted` (page 178) *property*
A Boolean value that determines whether the image changes when the button is highlighted.

`adjustsImageWhenDisabled` (page 177) *property*
A Boolean value that determines whether the image changes when the button is disabled.

`showsTouchWhenHighlighted` (page 183) *property*
A Boolean value that determines whether tapping the button causes it to glow.

- `backgroundImageForState:` (page 185)
Returns the background image used for a button state.

- `imageForState:` (page 186)
Returns the image used for a button state.

- `setBackgroundImage:forState:` (page 187)
Sets the background image to use for the specified button state.

- `setImage:forState:` (page 188)
Sets the image to use for the specified state.

Configuring Edge Insets

`contentEdgeInsets` (page 178) *property*
The inset or outset margins for the edges of the button content drawing rectangle.

`titleEdgeInsets` (page 183) *property*
The inset or outset margins for the edges of the button title drawing rectangle.

`imageEdgeInsets` (page 181) *property*
The inset or outset margins for the edges of the button image drawing rectangle.

Getting the Current State

`currentTitle` (page 180) *property*
The current title that is displayed on the button. (read-only)

`currentTitleColor` (page 180) *property*
The color used to display the title. (read-only)

`currentTitleShadowColor` (page 180) *property*
The color of the title's shadow. (read-only)

`currentImage` (page 179) *property*
The current image displayed on the button. (read-only)

`currentBackgroundImage` (page 179) *property*
The current background image displayed on the button. (read-only)

`imageView` (page 182) *property*
The button's image view. (read-only)

Getting Dimensions

`- backgroundRectForBounds:` (page 185)
Returns the rectangle in which the receiver draws its background.

`- contentRectForBounds:` (page 186)
Returns the rectangle in which the receiver draws its entire content.

`- titleRectForContentRect:` (page 191)
Returns the rectangle in which the receiver draws its title.

`- imageRectForContentRect:` (page 187)
Returns the rectangle in which the receiver draws its image.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

See Also
`@property adjustsImageWhenHighlighted` (page 178)

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 iOS 2.0 and later.

See Also
`@property adjustsImageWhenDisabled` (page 177)

Declared In
`UIButton.h`

buttonType

The button type. (read-only)

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

Discussion
See `UIButtonType` (page 191) for the possible values.

Availability
Available in iOS 2.0 and later.

Declared In
`UIButton.h`

contentEdgeInsets

The inset or outset margins for the edges of the button content drawing rectangle.

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

Discussion
Use this property to resize and reposition the effective drawing rectangle for the button content. The content comprises the button image and button title. You can specify a different value for each of the four insets (top, left, bottom, right). A positive value shrinks, or insets, that edge—moving it closer to the center of the button. A negative value expands, or outsets, that edge. Use the `UIEdgeInsetsMake` (page 1034) function to construct a value for this property. The default value is `UIEdgeInsetsZero` (page 1017).

Availability
Available in iOS 2.0 and later.

See Also
`@property imageEdgeInsets` (page 181)
`@property titleEdgeInsets` (page 183)

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 iOS 2.0 and later.

See Also
`@property currentImage` (page 179)

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 iOS 2.0 and later.

See Also
`@property currentBackgroundImage` (page 179)

Declared In
UIButton.h

currentTitle

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

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

Discussion
The value for this property is set automatically whenever the button state changes. The value may be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
- `setTitle:forState:` (page 188)
- `@property currentTitleColor` (page 180)
- `@property currentTitleShadowColor` (page 180)
- `@property titleLabel` (page 183)

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 iOS 2.0 and later.

See Also
- `@property currentTitle` (page 180)
- `@property currentTitleShadowColor` (page 180)

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 iOS 2.0 and later.

See Also
`@property currentTitle` (page 180)
`@property currentTitleColor` (page 180)

Declared In
`UIButton.h`

font

The font used to display text on the button. (Deprecated in iOS 3.0. Use the `font` property of the `titleLabel` (page 183) instead.)

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

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

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property titleLabel` (page 183)

Declared In
`UIButton.h`

imageEdgeInsets

The inset or outset margins for the edges of the button image drawing rectangle.

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

Discussion
Use this property to resize and reposition the effective drawing rectangle for the button image. You can specify a different value for each of the four insets (top, left, bottom, right). A positive value shrinks, or insets, that edge—moving it closer to the center of the button. A negative value expands, or outsets, that edge. Use the `UIEdgeInsetsMake` (page 1034) function to construct a value for this property. The default value is `UIEdgeInsetsZero` (page 1017).

Availability
Available in iOS 2.0 and later.

See Also
`@property contentEdgeInsets` (page 178)
`@property titleEdgeInsets` (page 183)

Declared In
`UIButton.h`

imageView

The button's image view. (read-only)

```
@property(nonatomic,readonly,retain) UIImageView *imageView
```

Discussion
Although this property is read-only, its own properties are read/write. Use these properties to configure the appearance and behavior of the button's view. For example:

```
UIButton *button                  = [UIButton buttonWithType:
UIButtonTypeRoundedRect];
button.imageView.exclusiveTouch    = YES;
```

The `imageView` property returns a value even if the button has not been displayed yet. The value of the property is `nil` for system buttons.

Availability
Available in iOS 3.0 and later.

Declared In
`UIButton.h`

lineBreakMode

The line break mode to use when drawing text. (Deprecated in iOS 3.0. Use the `lineBreakMode` property of the `titleLabel` (page 183) instead.)

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

Discussion
This property is one of the constants described in the `UILineBreakMode` (page 56) enumeration in *NSString UIKit Additions Reference*. The default value is `UILineBreakModeMiddleTruncation` (page 57).

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property titleLabel` (page 183)

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 iOS 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 iOS 2.0 and later.

See Also
@property adjustsImageWhenHighlighted (page 178)

Declared In
UIButton.h

titleEdgeInsets

The inset or outset margins for the edges of the button title drawing rectangle.

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

Discussion
Use this property to resize and reposition the effective drawing rectangle for the button title. You can specify a different value for each of the four insets (top, left, bottom, right). A positive value shrinks, or insets, that edge—moving it closer to the center of the button. A negative value expands, or outsets, that edge. Use the UIEdgeInsetsMake (page 1034) function to construct a value for this property. The default value is UIEdgeInsetsZero (page 1017).

Availability
Available in iOS 2.0 and later.

See Also
@property contentEdgeInsets (page 178)
@property imageEdgeInsets (page 181)

Declared In
UIButton.h

titleLabel

A view that displays the value of the currentTitle property for a button. (read-only)

```
@property(nonatomic,readonly,retain) UILabel *titleLabel
```

Discussion
Although this property is read-only, its own properties are read/write. Use these properties to configure the appearance of the button label. For example:

```
UIButton *button                  = [UIButton buttonWithType:
UIButtonTypeRoundedRect];
button.titleLabel.font            = [UIFont systemFontOfSize: 12];
button.titleLabel.lineBreakMode   = UILineBreakModeTailTruncation;
button.titleLabel.shadowOffset    = CGSizeMake (1.0, 0.0);
```

The `titleLabel` property returns a value even if the button has not been displayed yet. The value of the property is `nil` for system buttons.

Availability
Available in iOS 3.0 and later.

See Also
`@property currentTitle` (page 180)

Declared In
`UIButton.h`

titleShadowOffset

The offset of the shadow used to display the receiver's title. (Deprecated in iOS 3.0. Use the `shadowOffset` property of the `titleLabel` (page 183) instead.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property titleLabel` (page 183)

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 191) for the possible values.

Return Value

A newly created button.

Availability

Available in iOS 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 227).

Return Value

The background image used for the specified state.

Availability

Available in iOS 2.0 and later.

See Also

`- setBackgroundImage:forState:` (page 187)

Declared In

`UIButton.h`

backgroundRectForBounds:

Returns the rectangle in which the receiver draws its background.

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

Parameters

bounds

The bounding rectangle of the receiver.

Return Value

The rectangle in which the receiver draws its background.

Availability
Available in iOS 2.0 and later.

See Also
- `contentRectForBounds:` (page 186)

Declared In
`UIButton.h`

contentRectForBounds:

Returns the rectangle in which the receiver draws its entire content.

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

Parameters

bounds
The bounding rectangle for the receiver.

Return Value
The rectangle in which 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 iOS 2.0 and later.

See Also
- `titleRectForContentRect:` (page 191)
- `imageRectForContentRect:` (page 187)
- `backgroundRectForBounds:` (page 185)

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 227).

Return Value
The image used for the specified state.

Availability
Available in iOS 2.0 and later.

See Also
- `setImage:forState:` (page 188)

Declared In
`UIButton.h`

imageRectForContentRect:

Returns the rectangle in which the receiver draws its image.

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

Parameters

contentRect
The content rectangle for the receiver.

Return Value
The rectangle in which the receiver draws its image.

Availability
Available in iOS 2.0 and later.

See Also
- `contentRectForBounds:` (page 186)
- `titleRectForContentRect:` (page 191)

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 227).

Discussion
In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 227) 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 iOS 2.0 and later.

See Also
- `backgroundImageForState:` (page 185)

Related Sample Code
AddMusic

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 227).

Discussion
In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 227) 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 iOS 2.0 and later.

See Also
`- imageForState:` (page 186)

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 227).

Discussion
In general, if a property is not specified for a state, the default is to use the `UIControlStateNormal` (page 227) 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 iOS 2.0 and later.

See Also
- titleForState: (page 190)

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 227).

Discussion
In general, if a property is not specified for a state, the default is to use the UIControlStateNormal (page 227) 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 iOS 2.0 and later.

See Also
- titleColorForState: (page 190)

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 227).

Discussion
In general, if a property is not specified for a state, the default is to use the UIControlStateNormal (page 227) 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 iOS 2.0 and later.

See Also
- titleShadowColorForState: (page 191)

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 227).

Return Value
The color of the title for the specified state.

Availability
Available in iOS 2.0 and later.

See Also
- setTitleColor:forState: (page 189)

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 227).

Return Value
The title for the specified state.

Availability
Available in iOS 2.0 and later.

See Also
- setTitle:forState: (page 188)

Declared In
UIButton.h

titleRectForContentRect:

Returns the rectangle in which the receiver draws its title.

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

Parameters

contentRect

The content rectangle for the receiver.

Return Value

The rectangle in which the receiver draws its title.

Availability

Available in iOS 2.0 and later.

See Also

- `contentRectForBounds:` (page 186)
- `imageRectForContentRect:` (page 187)

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 227).

Return Value

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

Availability

Available in iOS 2.0 and later.

See Also

- `setTitleShadowColor:forState:` (page 189)

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 iOS 2.0 and later.
Declared in `UIButton.h`.

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

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

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

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

`UIButtonTypeContactAdd`
A contact add button.
Available in iOS 2.0 and later.
Declared in `UIButton.h`.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	UIColor.h UIInterface.h
Related sample code	BonjourWeb MoviePlayer ScrollViewSuite SpeakHere WiTap

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 iOS-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 200)
Creates and returns a color object using the specified opacity and grayscale values.

`+ colorWithHue:saturation:brightness:alpha:` (page 198)
Creates and returns a color object using the specified opacity and HSB color space component values.

+ colorWithRed:green:blue:alpha: (page 199)
Creates and returns a color object using the specified opacity and RGB component values.

+ colorWithCGColor: (page 198)
Creates and returns a color object using the specified Quartz color reference.

+ colorWithPatternImage: (page 199)
Creates and returns a color object using the specified image.

- colorWithAlphaComponent: (page 206)
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 209)
Initializes and returns a color object using the specified opacity and grayscale values.

- initWithHue:saturation:brightness:alpha: (page 207)
Initializes and returns a color object using the specified opacity and HSB color space component values.

- initWithRed:green:blue:alpha: (page 208)
Initializes and returns a color object using the specified opacity and RGB component values.

- initWithCGColor: (page 206)
Initializes and returns a color object using the specified Quartz color reference.

- initWithPatternImage: (page 208)
Initializes and returns a color object using the specified Quartz color reference.

Creating a UIColor with Preset Component Values

+ blackColor (page 196)
Returns a color object whose grayscale value is 0.0 and whose alpha value is 1.0.

+ darkGrayColor (page 201)
Returns a color object whose grayscale value is 1/3 and whose alpha value is 1.0.

+ lightGrayColor (page 203)
Returns a color object whose grayscale value is 2/3 and whose alpha value is 1.0.

+ whiteColor (page 205)
Returns a color object whose grayscale value is 1.0 and whose alpha value is 1.0.

+ grayColor (page 201)
Returns a color object whose grayscale value is 0.5 and whose alpha value is 1.0.

+ redColor (page 204)
Returns a color object whose RGB values are 1.0, 0.0, and 0.0 and whose alpha value is 1.0.

+ greenColor (page 202)
Returns a color object whose RGB values are 0.0, 1.0, and 0.0 and whose alpha value is 1.0.

+ blueColor (page 196)
Returns a color object whose RGB values are 0.0, 0.0, and 1.0 and whose alpha value is 1.0.

+ cyanColor (page 200)
Returns a color object whose RGB values are 0.0, 1.0, and 1.0 and whose alpha value is 1.0.

+ yellowColor (page 206)
Returns a color object whose RGB values are 1.0, 1.0, and 0.0 and whose alpha value is 1.0.

+ magentaColor (page 203)
Returns a color object whose RGB values are 1.0, 0.0, and 1.0 and whose alpha value is 1.0.

+ orangeColor (page 204)
Returns a color object whose RGB values are 1.0, 0.5, and 0.0 and whose alpha value is 1.0.

+ purpleColor (page 204)
Returns a color object whose RGB values are 0.5, 0.0, and 0.5 and whose alpha value is 1.0.

+ brownColor (page 197)
Returns a color object whose RGB values are 0.6, 0.4, and 0.2 and whose alpha value is 1.0.

+ clearColor (page 197)
Returns a color object whose grayscale and alpha values are both 0.0.

System Colors

+ lightTextColor (page 203)
Returns the system color used for displaying text on a dark background.

+ darkTextColor (page 201)
Returns the system color used for displaying text on a light background.

+ groupTableViewBackgroundColor (page 202)
Returns the system color used for the background of a grouped table.

+ viewFlipsideBackgroundColor (page 205)
Returns the system color used for the back side of a view while it is being flipped.

+ scrollViewTexturedBackgroundColor (page 205)
Returns the system pattern color used to render the area behind scrollable content.

Retrieving Color Information

CGColor (page 196) *property*
The Quartz color reference that corresponds to the receiver's color. (read-only)

Drawing Operations

- set (page 209)
Sets the color of subsequent stroke and fill operations to the color that the receiver represents.

- setFill (page 210)
Sets the color of subsequent fill operations to the color that the receiver represents.

- setStroke (page 210)
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 Programming Language*.

CGColor

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

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

Availability
Available in iOS 2.0 and later.

Related Sample Code
ScrollViewSuite
SpeakHere

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 iOS 2.0 and later.

Related Sample Code
BonjourWeb
GKRocket
MoviePlayer
ScrollViewSuite
WiTap

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
aurioTouch
BonjourWeb
MoviePlayer
ScrollViewSuite
WiTap

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 iOS 2.0 and later.

See Also

- initWithCGColor: (page 206)

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 iOS 2.0 and later.

See Also

- initWithHue:saturation:brightness:alpha: (page 207)

Related Sample Code
WiTap

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 iOS 2.0 and later.

See Also
`- initWithPatternImage:` (page 208)

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 iOS 2.0 and later.

See Also
`- initWithRed:green:blue:alpha:` (page 208)

Related Sample Code
AddMusic

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 iOS 2.0 and later.

See Also
`- initWithWhite:alpha:` (page 209)

Related Sample Code
BonjourWeb
WiTap

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer
WiTap

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 iOS 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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer
ScrollViewSuite

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 iOS 2.0 and later.

Related Sample Code
AddMusic
BonjourWeb
CryptoExercise

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer
ScrollViewSuite

Declared In
`UIColor.h`

scrollViewTexturedBackgroundColor

Returns the system pattern color used to render the area behind scrollable content.

```
+ (UIColor *)scrollViewTexturedBackgroundColor
```

Return Value
The `UIColor` object.

Availability
Available in iOS 3.2 and later.

Declared In
`UIInterface.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 iOS 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 iOS 2.0 and later.

Related Sample Code
aurioTouch
MoviePlayer
ScrollViewSuite
WiTap

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 iOS 2.0 and later.

Related Sample Code
GKRocket
MoviePlayer

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 iOS 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 iOS 2.0 and later.

See Also

`+ colorWithCGColor:` (page 198)

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 iOS 2.0 and later.

See Also

`+ colorWithHue:saturation:brightness:alpha:` (page 198)

Related Sample Code

GKRocket

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 iOS 2.0 and later.

See Also
`+ colorWithPatternImage:` (page 199)

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 iOS 2.0 and later.

See Also
`+ colorWithRed:green:blue:alpha:` (page 199)

Related Sample Code
SpeakHere

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 iOS 2.0 and later.

See Also
`+ colorWithWhite:alpha:` (page 200)

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 iOS 2.0 and later.

See Also
- `setFill` (page 210)
- `setStroke` (page 210)

Related Sample Code
GKRocket
SpeakHere

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 iOS 2.0 and later.

See Also
- `set` (page 209)

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 iOS 2.0 and later.

See Also
- `set` (page 209)

Declared In
`UIColor.h`

UIControl Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 211) 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 iOS 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 *iOS Application 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 222). 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 224), 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 223) | `UIControlEventTouchDragEnter` (page 223)).

You prepare a control for sending an action message by calling `addTarget:action:forControlEvents:` (page 218) 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 222). This results in `UIControl` sending the action to `UIApplication` in a `sendAction:to:from:forEvent:` (page 123) 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 221), evaluate the passed-in selector, target object, or `"Note"` (page 211) 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 219), `continueTrackingWithTouch:withEvent:` (page 220), `endTrackingWithTouch:withEvent:` (page 220).

Tasks

Preparing and Sending Action Messages

- `sendAction:to:forEvent:` (page 221)
  In response to a given event, forwards an action message to the application object for dispatching to a target.
- `sendActionsForControlEvents:` (page 222)
  Sends action messages for the given control events.

- `addTarget:action:forControlEvents:` (page 218)
  Adds a target and action for a particular event (or events) to an internal dispatch table.
- `removeTarget:action:forControlEvents:` (page 220)
  Removes a target and action for a particular event (or events) from an internal dispatch table.
- `actionsForTarget:forControlEvent:` (page 217)
  Returns the actions that are associated with a target and a particular control event.
- `allTargets` (page 219)
  Returns all target objects associated with the receiver.
- `allControlEvents` (page 218)
  Returns all control events associated with the receiver.

Setting and Getting Control Attributes

`state` (page 216) *property*
A Boolean value that indicates the state of the receiver. (read-only)

`enabled` (page 215) *property*
A Boolean value that determines whether the receiver is enabled.

`selected` (page 215) *property*
A Boolean value that determines the receiver's selected state.

`highlighted` (page 215) *property*
A Boolean value that determines whether the receiver is highlighted.

`contentVerticalAlignment` (page 214) *property*
The vertical alignment of content (text or image) within the receiver.

`contentHorizontalAlignment` (page 214) *property*
The horizontal alignment of content (text or image) within the receiver.

Tracking Touches and Redrawing Controls

- `beginTrackingWithTouch:withEvent:` (page 219)
  Sent the control when a touch related to the given event enters its bounds.
- `continueTrackingWithTouch:withEvent:` (page 220)
  Sent continuously to the control as it tracks a touch related to the given event within its bounds.
- `endTrackingWithTouch:withEvent:` (page 220)
  Sent to the control when the last touch for the given event completely ends, telling it to stop tracking.
- `cancelTrackingWithEvent:` (page 219)
  Tells the control to cancel tracking related to the given event.

`tracking` (page 217) *property*
A Boolean value that indicates whether the receiver is currently tracking touches related to an event. (read-only)

`touchInside` (page 216) *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 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 226) for descriptions of valid constants.

Discussion

The value of this property is a constant that specifies the alignment of text or image within the receiver. The default is `UIControlContentHorizontalAlignmentLeft` (page 226).

Availability

Available in iOS 2.0 and later.

See Also

`@property contentVerticalAlignment` (page 214)

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 225) for descriptions of valid constants.

Discussion

This value of this property is a constant that specifies the alignment of text or image within the receiver. The default is `UIControlContentVerticalAlignmentTop` (page 225).

Availability

Available in iOS 2.0 and later.

See Also

`@property contentHorizontalAlignment` (page 214)

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 iOS 2.0 and later.

See Also
`@property state` (page 216)

Related Sample Code
CryptoExercise

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 iOS 2.0 and later.

See Also
`@property state` (page 216)

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 iOS 2.0 and later.

See Also
`@property state` (page 216)

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 227) bit-mask constants that specify the state of the `UIControl` object; for information on these constants, see "Control State" (page 226). Note that the control can be in more than one state, for example, both disabled and selected (`UIControlStateDisabled` (page 227) | `UIControlStateSelected` (page 227)).This attribute is read only—there is no corresponding setter method.

Availability
Available in iOS 2.0 and later.

See Also
`@property enabled` (page 215)
`@property selected` (page 215)
`@property highlighted` (page 215)

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 iOS 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 iOS 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 211) that specifies a particular user action on the control; for a list of these constants, see "Control Events" (page 222).

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 iOS 2.0 and later.

See Also
- `sendAction:to:forEvent:` (page 221)
- `sendActionsForControlEvents:` (page 222)
- `addTarget:action:forControlEvents:` (page 218)

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 222) 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 iOS 2.0 and later.

See Also

- `removeTarget:action:forControlEvents:` (page 220)
- `actionsForTarget:forControlEvent:` (page 217)

Declared In

`UIControl.h`

allControlEvents

Returns all control events associated with the receiver.

```
- (UIControlEvents)allControlEvents
```

Return Value

One or more "`Note`" (page 211) constants that specify the current control events associated with the receiver; for a list of these constants, see "Control Events" (page 222)list of all events that have at least one action.

Availability

Available in iOS 2.0 and later.

See Also

- `allTargets` (page 219)

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 iOS 2.0 and later.

See Also
`- allControlEvents` (page 218)

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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
UIControl.h

endTrackingWithTouch:withEvent:

Sent to the control when the last touch for the given event completely ends, 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 iOS 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. Pass `nil` to remove all targets paired with *action* and the specified control events.

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 222) for bitmask constants.

Availability
Available in iOS 2.0 and later.

See Also
- addTarget:action:forControlEvents: (page 218)

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 222) might call this method repeatedly, once for each specified control event.

Availability
Available in iOS 2.0 and later.

See Also
- addTarget:action:forControlEvents: (page 218)
- sendActionsForControlEvents: (page 222)

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 222) for bitmask constants.

Discussion
`UIControl` implements this method to send all action messages associated with *controlEvents*, repeatedly invoking `sendAction:to:forEvent:` (page 221) in the process. The list of targets and actions it looks up is constructed from prior invocations of `addTarget:action:forControlEvents:` (page 218).

Availability
Available in iOS 2.0 and later.

See Also
- `sendAction:to:forEvent:` (page 221)
- `addTarget:action:forControlEvents:` (page 218)

Declared In
`UIControl.h`

Constants

UIControlEvents

The type for the `enum` constants listed in "Control Events" (page 222).

```
typedef NSUInteger UIControlEvents;
```

Availability
Available in iOS 2.0 and later.

Declared In
`UIControl.h`

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 iOS 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 iOS 2.0 and later.

Declared in `UIControl.h`.

`UIControlEventTouchDragInside`

An event where a finger is dragged inside the bounds of the control.

Available in iOS 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 iOS 2.0 and later.

Declared in `UIControl.h`.

`UIControlEventTouchDragEnter`

An event where a finger is dragged into the bounds of the control.

Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventTouchCancel`
: A system event canceling the current touches for the control.

  Available in iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventEditingChanged`
: A touch making an editing change in a `UITextField` objet.

  Available in iOS 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 iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventEditingDidEndOnExit`
: A touch ending an editing session in a `UITextField` object.

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventAllTouchEvents`
: All touch events.

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventAllEditingEvents`
: All editing touches for `UITextField` objects.

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventApplicationReserved`
: A range of control-event values available for application use.

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventSystemReserved`
: A range of control-event values reserved for internal framework use.

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlEventAllEvents`
: All events, including system events.

  Available in iOS 2.0 and later.

  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 iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlContentVerticalAlignmentTop`
: Aligns the content vertically at the top in the control (the default).

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlContentVerticalAlignmentBottom`
: Aligns the content vertically at the bottom in the control

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

`UIControlContentVerticalAlignmentFill`
: Aligns the content vertically to fill the content rectangle; images may be stretched.

  Available in iOS 2.0 and later.

  Declared in `UIControl.h`.

Discussion
You use these constants as the value of the `contentVerticalAlignment` (page 214) property.

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 iOS 2.0 and later.
Declared in `UIControl.h`.

`UIControlContentHorizontalAlignmentLeft`
Aligns the content horizontally from the left of the control (the default).
Available in iOS 2.0 and later.
Declared in `UIControl.h`.

`UIControlContentHorizontalAlignmentRight`
Aligns the content horizontally from the right of the control
Available in iOS 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 iOS 2.0 and later.
Declared in `UIControl.h`.

Discussion
You use these constants as the value of the `contentHorizontalAlignment` (page 214) property.

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 188) 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,
    UIControlStateApplication          = 0x00FF0000,
    UIControlStateReserved             = 0xFF000000
};
```

Constants

`UIControlStateNormal`

The normal, or default state of a control—that is, enabled but neither selected or highlighted.

Available in iOS 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 215) property.

Available in iOS 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 215) property.

Available in iOS 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 215) property.

Available in iOS 2.0 and later.

Declared in `UIControl.h`.

`UIControlStateApplication`

Additional control-state flags available for application use.

Available in iOS 2.0 and later.

Declared in `UIControl.h`.

`UIControlStateReserved`

Control-state flags reserved for internal framework use.

Available in iOS 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 226). Use the `state` (page 216) property to retrieve the current state bits set for a control.

Availability
Available in iOS 2.0 and later.

Declared In
`UIControl.h`

UIDatePicker Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 224). 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 230) *property*
The calendar to use for the date picker.

`date` (page 231) *property*
The date displayed by the date picker.

`locale` (page 232) *property*
The locale used by the date picker.

- `setDate:animated:` (page 234)
  Sets the date to display in the date picker, with an option to animate the setting.

`timeZone` (page 233) *property*
The time zone reflected in the date displayed by the date picker.

Configuring the Date Picker Mode

`datePickerMode` (page 231) *property*
The mode of the date picker.

Configuring Temporal Attributes

`maximumDate` (page 232) *property*
The maximum date that a date picker can show.

`minimumDate` (page 232) *property*
The minimum date that a date picker can show.

`minuteInterval` (page 233) *property*
The interval at which the date picker should display minutes.

`countDownDuration` (page 231) *property*
The seconds from which the countdown timer counts down.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

See Also
`@property locale` (page 232)
`@property timeZone` (page 233)

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 235), this value is ignored. The default value is 0.0 and the maximum value is 23:59 (86,399 seconds).

Availability
Available in iOS 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 235); 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 234) method.

Availability
Available in iOS 2.0 and later.

See Also
`- setDate:animated:` (page 234)

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 235). See "Date Picker Mode" (page 234) for a list of mode constants.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property calendar` (page 230)
`@property timeZone` (page 233)

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 232) 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 235)).

Availability
Available in iOS 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 232) 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 235)).

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

See Also
`@property calendar` (page 230)
`@property locale` (page 232)

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 iOS 2.0 and later.

See Also
`@property date` (page 231)

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 iOS 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 iOS 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 iOS 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 iOS 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 231) property.

UIDevice Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIDevice.h
Related sample code	CryptoExercise GKTank GLSprite

Overview

The `UIDevice` class provides 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 also use the `UIDevice` instance to detect changes in the device's characteristics, such as physical orientation. You get the current orientation using the `orientation` (page 242) property or receive change notifications by registering for the `UIDeviceOrientationDidChangeNotification` (page 249) notification. Before using either of these techniques to get orientation data, you must enable data delivery using the `beginGeneratingDeviceOrientationNotifications` (page 245) method. When you no longer need to track the device orientation, call the `endGeneratingDeviceOrientationNotifications` (page 246) method to disable the delivery of notifications.

Similarly, you can use the `UIDevice` instance to obtain information and notifications about changes to the battery's charge state (described by the `batteryState` (page 240) property) and charge level (described by the `batteryLevel` (page 239) property). The `UIDevice` instance also provides access to the proximity sensor state (described by the `proximityState` (page 243) property). The proximity sensor detects whether the user is holding the device close to their face. Enable battery monitoring or proximity sensing only when you need it.

Tasks

Getting the Shared Device Instance

`+ currentDevice` (page 245)
Returns an object representing the current device.

Determining the Available Features

`multitaskingSupported` (page 241) *property*
A Boolean value indicating whether multitasking is supported on the current device. (read-only)

Identifying the Device and Operating System

`uniqueIdentifier` (page 244) *property*
A string unique to each device based on various hardware details. (read-only)

`name` (page 242) *property*
The name identifying the device. (read-only)

`systemName` (page 243) *property*
The name of the operating system running on the device represented by the receiver. (read-only)

`systemVersion` (page 244) *property*
The current version of the operating system. (read-only)

`model` (page 241) *property*
The model of the device. (read-only)

`localizedModel` (page 241) *property*
The model of the device as a localized string. (read-only)

`userInterfaceIdiom` (page 244) *property*
The style of interface to use on the current device. (read-only)

Getting the Device Orientation

`orientation` (page 242) *property*
Returns the physical orientation of the device. (read-only)

`generatesDeviceOrientationNotifications` (page 240) *property*
A Boolean value that indicates whether the receiver generates orientation notifications (`YES`) or not (`NO`). (read-only)

`- beginGeneratingDeviceOrientationNotifications` (page 245)
Begins the generation of notifications of device orientation changes.

`- endGeneratingDeviceOrientationNotifications` (page 246)
Ends the generation of notifications of device orientation changes.

Getting the Device Battery State

`batteryLevel` (page 239) *property*
The battery charge level for the device.

`batteryMonitoringEnabled` (page 239) *property*
A Boolean value indicating whether battery monitoring is enabled (`YES`) or not (`NO`).

`batteryState` (page 240) *property*
The battery state for the device.

Using the Proximity Sensor

`proximityMonitoringEnabled` (page 242) *property*
A Boolean value indicating whether proximity monitoring is enabled (`YES`) or not (`NO`).

`proximityState` (page 243) *property*
A Boolean value indicating whether the proximity sensor is close to the user (`YES`) or not (`NO`).

Properties

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

batteryLevel

The battery charge level for the device.

```
@property (nonatomic,readonly) float batteryLevel
```

Discussion
Battery level ranges from 0.0 (fully discharged) to 1.0 (100% charged). Before accessing this property, ensure that battery monitoring is enabled.

If battery monitoring is not enabled, battery state is `UIDeviceBatteryStateUnknown` (page 246) and the value of this property is –1.0.

Availability
Available in iOS 3.0 and later.

See Also
`@property batteryState` (page 240)
`@property batteryMonitoringEnabled` (page 239)

Declared In
`UIDevice.h`

batteryMonitoringEnabled

A Boolean value indicating whether battery monitoring is enabled (`YES`) or not (`NO`).

```
@property (nonatomic,getter=isBatteryMonitoringEnabled) BOOL batteryMonitoringEnabled
```

Discussion
Enable battery monitoring only when your application needs to be notified of changes to the battery state. Otherwise, disable battery monitoring. The default value is `NO`.

Availability
Available in iOS 3.0 and later.

See Also
`@property batteryLevel` (page 239)
`@property batteryState` (page 240)

Declared In
`UIDevice.h`

batteryState

The battery state for the device.

```
@property (nonatomic,readonly) UIDeviceBatteryState batteryState
```

Discussion
The value for `batteryState` is one of the constants in "UIDeviceBatteryState" (page 246).

If battery monitoring is not enabled, the value of this property is `UIDeviceBatteryStateUnknown` (page 246).

Availability
Available in iOS 3.0 and later.

See Also
`@property batteryLevel` (page 239)
`@property batteryMonitoringEnabled` (page 239)

Declared In
`UIDevice.h`

generatesDeviceOrientationNotifications

A Boolean value that indicates whether the receiver generates orientation notifications (`YES`) or not (`NO`). (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 249) 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 iOS 2.0 and later.

See Also
- `beginGeneratingDeviceOrientationNotifications` (page 245)
- `endGeneratingDeviceOrientationNotifications` (page 246)

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 241).

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
`UIDevice.h`

multitaskingSupported

A Boolean value indicating whether multitasking is supported on the current device. (read-only)

```
@property(nonatomic,readonly,getter=isMultitaskingSupported) BOOL
    multitaskingSupported
```

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

See Also
`@property systemName` (page 243)

Related Sample Code
CryptoExercise

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 247) for descriptions of the possible values.

The value of this property always returns 0 unless orientation notifications have been enabled by calling `beginGeneratingDeviceOrientationNotifications` (page 245).

Availability
Available in iOS 2.0 and later.

See Also
`@property generatesDeviceOrientationNotifications` (page 240)
`- beginGeneratingDeviceOrientationNotifications` (page 245)

Declared In
`UIDevice.h`

proximityMonitoringEnabled

A Boolean value indicating whether proximity monitoring is enabled (`YES`) or not (`NO`).

```
@property (nonatomic,getter=isProximityMonitoringEnabled) BOOL
    proximityMonitoringEnabled
```

Discussion
Enable proximity monitoring only when your application needs to be notified of changes to the proximity state. Otherwise, disable proximity monitoring. The default value is `NO`.

Not all iOS devices have proximity sensors. To determine if proximity monitoring is available, attempt to enable it. If the value of the `proximityState` (page 243) property remains `NO`, proximity monitoring is not available.

Availability
Available in iOS 3.0 and later.

See Also
`@property proximityState` (page 243)

Declared In
`UIDevice.h`

proximityState

A Boolean value indicating whether the proximity sensor is close to the user (`YES`) or not (`NO`).

```
@property (nonatomic,readonly) BOOL proximityState
```

Availability
Available in iOS 3.0 and later.

See Also
`@property proximityMonitoringEnabled` (page 242)

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 iOS 2.0 and later.

See Also
`@property name` (page 242)
`@property systemVersion` (page 244)

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 iOS 2.0 and later.

See Also
`@property systemName` (page 243)

Related Sample Code
GLSprite

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 but cannot publicly be tied to a user account. You can use it, for example, to store high scores for a game in a central server or to control access to registered products. The unique device identifier is sometimes referred to by its abbreviation UDID.

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKTank

Declared In
`UIDevice.h`

userInterfaceIdiom

The style of interface to use on the current device. (read-only)

```
@property(nonatomic,readonly) UIUserInterfaceIdiom userInterfaceIdiom
```

Discussion
For universal applications, you can use this property to tailor the behavior of your application for a specific type of device. For example, iPhone and iPad devices have different screen sizes, so you might want to create different views and controls based on the type of the current device.

Availability
Available in iOS 3.2 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 access the properties of the returned `UIDevice` instance to obtain information about the device. You must instantiate the `UIDevice` instance before registering to receive device notifications.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise
GKTank
GLSprite

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 249) 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 iOS 2.0 and later.

See Also
- `endGeneratingDeviceOrientationNotifications` (page 246)
`@property orientation` (page 242)
`@property generatesDeviceOrientationNotifications` (page 240)

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 249) 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 iOS 2.0 and later.

See Also
- `beginGeneratingDeviceOrientationNotifications` (page 245)

Declared In
`UIDevice.h`

Constants

UIDeviceBatteryState

The battery power state of the device.

```
typedef enum {
    UIDeviceBatteryStateUnknown,
    UIDeviceBatteryStateUnplugged,
    UIDeviceBatteryStateCharging,
    UIDeviceBatteryStateFull,
} UIDeviceBatteryState;
```

Constants
`UIDeviceBatteryStateUnknown`
The battery state for the device cannot be determined.
Available in iOS 3.0 and later.
Declared in `UIDevice.h`.

`UIDeviceBatteryStateUnplugged`
The device is not plugged into power; the battery is discharging.

Available in iOS 3.0 and later.

Declared in `UIDevice.h`.

`UIDeviceBatteryStateCharging`
The device is plugged into power and the battery is less than 100% charged.

Available in iOS 3.0 and later.

Declared in `UIDevice.h`.

`UIDeviceBatteryStateFull`
The device is plugged into power and the battery is 100% charged.

Available in iOS 3.0 and later.

Declared in `UIDevice.h`.

Discussion
These constants are used by the `batteryState` (page 240) property.

UIDeviceOrientation

The physical orientation of the device.

```
typedef enum {
    UIDeviceOrientationUnknown,
    UIDeviceOrientationPortrait,
    UIDeviceOrientationPortraitUpsideDown,
    UIDeviceOrientationLandscapeLeft,
    UIDeviceOrientationLandscapeRight,
    UIDeviceOrientationFaceUp,
    UIDeviceOrientationFaceDown
} UIDeviceOrientation;
```

Constants
`UIDeviceOrientationUnknown`
The orientation of the device cannot be determined.

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `UIDevice.h`.

`UIDeviceOrientationFaceUp`
The device is held parallel to the ground with the screen facing upwards.

Available in iOS 2.0 and later.

Declared in `UIDevice.h`.

`UIDeviceOrientationFaceDown`
The device is held parallel to the ground with the screen facing downwards.

Available in iOS 2.0 and later.

Declared in `UIDevice.h`.

Discussion
The `orientation` (page 242) 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.

UIUserInterfaceIdiom

The type of interface that should be used on the current device

```
typedef enum {
   UIUserInterfaceIdiomPhone,
   UIUserInterfaceIdiomPad,
} UIUserInterfaceIdiom;
```

Constants
`UIUserInterfaceIdiomPhone`
The user interface should be designed for iPhone and iPod touch.

Available in iOS 3.2 and later.

Declared in `UIDevice.h`.

`UIUserInterfaceIdiomPad`
The user interface should be designed for iPad.

Available in iOS 3.2 and later.

Declared in `UIDevice.h`.

Notifications

All `UIDevice` notifications are posted by the singleton device instance returned by `currentDevice` (page 245).

UIDeviceBatteryLevelDidChangeNotification

Posted when the battery level changes.

Notifications for battery level change are sent no more frequently than once per minute. Do not attempt to calculate battery drainage rate or battery time remaining; drainage rate can change frequently depending on built-in applications as well as your application.

You can obtain the battery level by getting the value of the `batteryLevel` (page 239) property.

Availability
Available in iOS 3.0 and later.

Declared In
`UIDevice.h`

UIDeviceBatteryStateDidChangeNotification

Posted when battery state changes.

You can obtain the battery state by getting the value of the `batteryState` (page 240) property.

Availability
Available in iOS 3.0 and later.

Declared In
`UIDevice.h`

UIDeviceOrientationDidChangeNotification

Posted when the orientation of the device changes.

You can obtain the new orientation by getting the value of the `orientation` (page 242) property.

Availability
Available in iOS 2.0 and later.

Declared In
`UIDevice.h`

UIDeviceProximityStateDidChangeNotification

Posted when the state of the proximity sensor changes.

You can obtain the proximity state by getting the value of the `proximityState` (page 243) property.

Availability
Available in iOS 3.0 and later.

Declared In
`UIDevice.h`

UIDocumentInteractionController Class Reference

Inherits from	NSObject
Conforms to	UIActionSheetDelegate NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIDocumentInteractionController.h
Companion guide	iPad Programming Guide

Overview

The `UIDocumentInteractionController` class provides in-application support for managing user interactions with files in the local system. For example, a mail program might use this class to allow the user to preview and open attachments inline with a mail message. You use this class to present the user with an appropriate interface for previewing, opening, or copying the specified file.

After presenting the user interface, the document interaction controller handles all interactions needed to support the preview and menu display. However, in some cases the object does use its associated delegate to determine the appropriate way to respond to specific commands. You can also use the delegate to monitor interactions occurring within the presented interface. For more information about the methods you can implement in your delegate, see *UIDocumentInteractionControllerDelegate Protocol Reference*.

Tasks

Creating the Document Interaction Controller

`+ interactionControllerWithURL:` (page 255)
Creates and returns a new `UIDocumentationInteractionController` object initialized with the specified URL.

Presenting and Dismissing a Document Preview

`- presentPreviewAnimated:` (page 259)
Displays a full-screen preview of the target document.

- `dismissPreviewAnimated:` (page 256)
  Dismisses the currently active document preview.

Presenting and Dismissing Menus

- `presentOptionsMenuFromRect:inView:animated:` (page 258)
  Displays an options menu and anchors it to the specified location in the view.
- `presentOptionsMenuFromBarButtonItem:animated:` (page 258)
  Displays an options menu and anchors it to the specified bar button item.
- `presentOpenInMenuFromRect:inView:animated:` (page 257)
  Displays a menu for opening the document and anchors that menu to the specified view.
- `presentOpenInMenuFromBarButtonItem:animated:` (page 256)
  Displays a menu for opening the document and anchors that menu to the specified bar button item.
- `dismissMenuAnimated:` (page 255)
  Dismisses the currently active menu.

Accessing the Target Document's Attributes

`URL` (page 254) *property*
The URL identifying the target file on the local filesystem.

`UTI` (page 255) *property*
The type of the target file.

`name` (page 254) *property*
The name of the target file.

`icons` (page 254) *property*
The images associated with the target file. (read-only)

`annotation` (page 253) *property*
Custom property list information for the target file.

Accessing the Controller Attributes

`gestureRecognizers` (page 253) *property*
The built-in gesture recognizers supported for document interactions. (read-only)

`delegate` (page 253) *property*
The delegate you want to receive document interaction notifications.

Properties

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

annotation

Custom property list information for the target file.

```
@property(nonatomic,retain) id annotation
```

Discussion
You can use this property to pass information about the document type to the application responsible for opening it. The type of this object should be one of the types used to contain property list information, which includes an `NSDictionary`, `NSArray`, `NSData`, `NSString`, `NSNumber`, or `NSDate`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

delegate

The delegate you want to receive document interaction notifications.

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

Discussion
You can use a delegate object to track user interactions with menu items displayed by the document interaction controller. For more information about the methods of the `UIDocumentInteractionControllerDelegate Protocol Reference` protocol, see *UIDocumentInteractionControllerDelegate Protocol Reference*.

The default value of this property is `nil`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

gestureRecognizers

The built-in gesture recognizers supported for document interactions. (read-only)

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

Discussion
The objects in this array all descend from the `UIGestureRecognizer` class. You can attach these gesture recognizers to the view you use to represent the document. Those gestures trigger the appropriate document interactions automatically.

Currently the document interaction controller supports only tap and long press gestures. Tap gestures initiate a preview of the document and long press gestures display the options menu.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

icons

The images associated with the target file. (read-only)

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

Discussion
This property contains an array of `UIImage` objects containing the available icons for the given file. The images in the array are sorted from smallest to largest, with the smallest image located at index 0. The returned array always contains at least one image.

The images themselves are provided by the system and determined by the UTI of the file. Applications can register custom icons for their associated files by including the appropriate meta information in their `Info.plist` file. If no custom icon exists, the images in this property represent the generic document icon.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

name

The name of the target file.

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

Discussion
This property contains the filename without any preceding path information. The default value of this property is derived from the path information in the `URL` (page 254) property. You can change the value of this property as needed if you want to associate a different name with the file.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

URL

The URL identifying the target file on the local filesystem.

```
@property(nonatomic,retain) NSURL *URL
```

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

UTI

The type of the target file.

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

Discussion
The value of this property is used to determine which applications are capable of opening the document. The default value is determined automatically whenever possible. However, if the document is a custom type that cannot be determined readily, the value of this property may be `nil`. If you know the type of the document, you can set the value of this property explicitly.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

Class Methods

interactionControllerWithURL:

Creates and returns a new `UIDocumentationInteractionController` object initialized with the specified URL.

```
+ (UIDocumentInteractionController *)interactionControllerWithURL:(NSURL *)url
```

Parameters

url

A URL that specifies the location of the desired document. This parameter is retained. It can be changed later by modifying the `URL` (page 254) property.

Return Value
A new document interaction controller object.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

Instance Methods

dismissMenuAnimated:

Dismisses the currently active menu.

```
- (void)dismissMenuAnimated:(BOOL)animated
```

Parameters

animated

Specify YES to animate the dismissal of the document preview or NO to dismiss it immediately.

Discussion

You can use this method to dismiss the menu programmatically. The document interaction controller may also dismiss the menu automatically in response to user actions.

Availability

Available in iOS 3.2 and later.

Declared In

UIDocumentInteractionController.h

dismissPreviewAnimated:

Dismisses the currently active document preview.

```
- (void)dismissPreviewAnimated:(BOOL)animated
```

Parameters

animated

Specify YES to animate the dismissal of the document preview or NO to dismiss it immediately.

Discussion

You can use this method to dismiss the document preview programmatically. The document interaction controller may also dismiss the document preview automatically in response to user actions.

Availability

Available in iOS 3.2 and later.

Declared In

UIDocumentInteractionController.h

presentOpenInMenuFromBarButtonItem:animated:

Displays a menu for opening the document and anchors that menu to the specified bar button item.

```
- (BOOL)presentOpenInMenuFromBarButtonItem:(UIBarButtonItem *)item
    animated:(BOOL)animated
```

Parameters

item

The bar button item to which to anchor the menu.

animated

Specify YES to animate the appearance of the menu or NO to display it immediately.

Return Value

YES if this method was able to display the menu or NO if it was not.

Discussion
When building the menu, this method includes only those applications capable of opening the current document. This determination is made based on the type of the document (as specified by the `UTI` (page 255) property) and the document types supported by the installed applications. To support one or more document types, an application must register those types in its `Info.plist` file using the `CFBundleDocumentTypes` key.

This method displays the menu asynchronously. The document interaction controller dismisses the menu automatically when the user selects an appropriate application. You can also dismiss it programmatically using the `dismissMenuAnimated:` (page 255) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

presentOpenInMenuFromRect:inView:animated:

Displays a menu for opening the document and anchors that menu to the specified view.

```
- (BOOL)presentOpenInMenuFromRect:(CGRect)rect inView:(UIView *)view
    animated:(BOOL)animated
```

Parameters

rect
The location (in the coordinate system of *view*) at which to anchor the menu.

view
The view from which to display the menu.

animated
Specify `YES` to animate the appearance of the menu or `NO` to display it immediately.

Return Value
`YES` if this method was able to display the menu or `NO` if it was not.

Discussion
When building the menu, this method includes only those applications capable of opening the current document. This determination is made based on the type of the document (as specified by the `UTI` (page 255) property) and the document types supported by the installed applications. To support one or more document types, an application must register those types in its `Info.plist` file using the `CFBundleDocumentTypes` key.

This method displays the options menu asynchronously. The document interaction controller dismisses the menu automatically when the user selects an appropriate option. You can also dismiss it programmatically using the `dismissMenuAnimated:` (page 255) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

presentOptionsMenuFromBarButtonItem:animated:

Displays an options menu and anchors it to the specified bar button item.

```
- (BOOL)presentOptionsMenuFromBarButtonItem:(UIBarButtonItem *)item
    animated:(BOOL)animated
```

Parameters

item

The bar button item to which to anchor the menu.

animated

Specify `YES` to animate the appearance of the menu or `NO` to display it immediately.

Return Value

`YES` if the options menu was displayed or `NO` if it was not. The options menu may not be displayed in cases where there are no appropriate items to include in the menu.

Discussion

The contents of the options menu are built dynamically based on the type of the document. Options that cannot be performed on the current document are not included in the menu. For example, if the document cannot be opened by any known applications, the menu does not include options for opening it.

This method displays the options menu asynchronously. The document interaction controller dismisses the menu automatically when the user selects an appropriate option. You can also dismiss it programmatically using the `dismissMenuAnimated:` (page 255) method.

Availability

Available in iOS 3.2 and later.

Declared In

`UIDocumentInteractionController.h`

presentOptionsMenuFromRect:inView:animated:

Displays an options menu and anchors it to the specified location in the view.

```
- (BOOL)presentOptionsMenuFromRect:(CGRect)rect inView:(UIView *)view
    animated:(BOOL)animated
```

Parameters

rect

The location (in the coordinate system of *view*) at which to anchor the menu.

view

The view from which to display the options menu.

animated

Specify `YES` to animate the appearance of the menu or `NO` to display it immediately.

Return Value

`YES` if the options menu was displayed or `NO` if it was not. The options menu may not be displayed in cases where there are no appropriate items to include in the menu.

Discussion
The contents of the options menu are built dynamically based on the type of the document. Options that cannot be performed on the current document are not included in the menu. For example, if the document cannot be opened by any known applications, the menu does not include options for opening it.

This method displays the options menu asynchronously. The menu is dismissed automatically when the user selects one of the available options. You can also dismiss it programmatically using the `dismissMenuAnimated:` (page 255) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

presentPreviewAnimated:

Displays a full-screen preview of the target document.

```
- (BOOL)presentPreviewAnimated:(BOOL)animated
```

Parameters

animated
Specify `YES` to animate the appearance of the document preview or `NO` to display it immediately.

Return Value
`YES` if this method was able to display the document preview or `NO` if it was not.

Discussion
Before calling this method, you must provide a delegate object that implements the `documentInteractionControllerViewControllerForPreview:` (page 856) method. The view controller returned by that method is used to present the document preview modally.

If your delegate implements the `documentInteractionControllerViewForPreview:` (page 857) and `documentInteractionControllerRectForPreview:` (page 856) methods, the view and rectangle returned by those methods is used as the starting point for the animation used to display the document preview. If the animated parameter is `YES` but your delegate does not implement the `documentInteractionControllerViewForPreview:` method (or that method returns `nil`), the document preview is animated into place using a crossfade transition.

This method displays the document preview asynchronously. The document interaction controller dismisses the document preview automatically in response to appropriate user interactions. You can also dismiss the preview programmatically using the `dismissPreviewAnimated:` (page 256) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

UIEvent Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIEvent.h
Companion guide	Event Handling Guide for iOS
Related sample code	aurioTouch GKTank MoviePlayer ScrollViewSuite WiTap

Overview

A `UIEvent` object (or, simply, an event object) represents an event in iOS. There are three general types of event: touch events, motion events, and remote-control events. Remote-control events allow a responder object to receive commands from an external accessory or headset so that it can manage manage audio and video—for example, playing a video or skipping to the next audio track. Motion events were introduced in iOS 3.0 and remote-control events in iOS 4.0.

A touch type of 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 a touch 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 467). 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 264)) or only those for a given view or window (`touchesForView:` (page 265) or `touchesForWindow:` (page 265)). It can also distinguish an event object from objects representing other events by querying an object for the time of its creation (`timestamp` (page 263)).

A `UIEvent` object representing a touch event 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.

You can obtain event types and subtypes from the `type` (page 263) and `subtype` (page 263) properties. `UIEvent` defines event types for touch, motion, and remote-control events. It also defines a motion subtype for "shake" events and a series of subtype constants for remote-control events, such as “play” and “previous track.” The first responder or any responder in the responder chain implements the motion-related methods of `UIResponder` (such as `motionBegan:withEvent:` (page 463)) to handle shaking-motion events. To handle remote-control events, a responder object must implement the `remoteControlReceivedWithEvent:` (page 466) method of `UIResponder`.

The `touchesForGestureRecognizer:` (page 264) method, which was introduced in iOS 3.2, allows you to query a gesture-recognizer object (an instance of a subclass of `UIGestureRecognizer`) for the touches it is currently handling.

Tasks

Getting the Touches for an Event

- `allTouches` (page 264)
  Returns all touch objects associated with the receiver.
- `touchesForView:` (page 265)
  Returns the touch objects that belong to a given view for the event represented by the receiver.
- `touchesForWindow:` (page 265)
  Returns the touch objects that belong to a given window for the event represented by the receiver.

Getting Event Attributes

`timestamp` (page 263) *property*
The time when the event occurred. (read-only)

Getting the Event Type

`type` (page 263) *property*
Returns the type of the event. (read-only)

`subtype` (page 263) *property*
Returns the subtype of the event. (read-only)

Getting the Touches for a Gesture Recognizer

- `touchesForGestureRecognizer:` (page 264)
  Returns the touch objects that are being delivered to the specified gesture recognizer.

Properties

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

subtype

Returns the subtype of the event. (read-only)

```
@property(readonly) UIEventSubtype subtype
```

Discussion
The `UIEventSubtype` (page 266) constant returned by this property indicates the subtype of the event in relation to the general type, which is returned from the `type` (page 263) property.

Availability
Available in iOS 3.0 and later.

Declared In
`UIEvent.h`

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. For a description of this time value, see the description of the `systemUptime` method of the `NSProcessInfo` class.

Availability
Available in iOS 2.0 and later.

Declared In
`UIEvent.h`

type

Returns the type of the event. (read-only)

```
@property(readonly) UIEventType type
```

Discussion
The `UIEventType` (page 266) constant returned by this property indicates the general type of this event, for example, whether it is a touch or motion event.

Availability
Available in iOS 3.0 and later.

See Also
`@property subtype` (page 263)

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 iOS 2.0 and later.

See Also
- `touchesForView:` (page 265)
- `touchesForWindow:` (page 265)

Related Sample Code
aurioTouch
GKTank

Declared In
`UIEvent.h`

touchesForGestureRecognizer:

Returns the touch objects that are being delivered to the specified gesture recognizer.

```
- (NSSet *)touchesForGestureRecognizer:(UIGestureRecognizer *)gesture
```

Parameters
gesture
An instance of a subclass of the abstract base class UIGestureRecognizer. This gesture-recognizer object must be attached to a view to receive the touches hit-tested to that view and its subviews.

Return Value
An set of `UITouch` objects representing the touches being delivered to the specified gesture recognizer for the event represented by the receiver.

Availability
Available in iOS 3.2 and later.

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 iOS 2.0 and later.

See Also
- allTouches (page 264)
- touchesForWindow: (page 265)

Related Sample Code
ScrollViewSuite

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 iOS 2.0 and later.

See Also
- allTouches (page 264)
- touchesForView: (page 265)

Declared In
UIEvent.h

Constants

UIEventType

Specifies the general type of an event

```
typedef enum {
    UIEventTypeTouches,
    UIEventTypeMotion,
    UIEventTypeRemoteControl,
} UIEventType;
```

Constants

`UIEventTypeTouches`
The event is related to touches on the screen.

Available in iOS 3.0 and later.

Declared in `UIEvent.h`.

`UIEventTypeMotion`
The event is related to motion of the device, such as when the user shakes it.

Available in iOS 3.0 and later.

Declared in `UIEvent.h`.

`UIEventTypeRemoteControl`
The event is a remote-control event. Remote-control events originate as commands received from a headset or external accessory for the purposes of controlling multimedia on the device.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

Discussion

You can obtain the type of an event from the `type` (page 263) property. To further identify the event, you might also need to determine its subtype, which you obtain from the `subtype` (page 263) property.

Availability

Available in iOS 3.0 and later.

Declared In

`UIEvent.h`

UIEventSubtype

Specifies the subtype of the event in relation to its general type.

```
typedef enum {
    UIEventSubtypeNone                              = 0,

    UIEventSubtypeMotionShake                       = 1,
```

```
    UIEventSubtypeRemoteControlPlay                = 100,
    UIEventSubtypeRemoteControlPause               = 101,
    UIEventSubtypeRemoteControlStop                = 102,
    UIEventSubtypeRemoteControlTogglePlayPause     = 103,
    UIEventSubtypeRemoteControlNextTrack           = 104,
    UIEventSubtypeRemoteControlPreviousTrack       = 105,
    UIEventSubtypeRemoteControlBeginSeekingBackward = 106,
    UIEventSubtypeRemoteControlEndSeekingBackward  = 107,
    UIEventSubtypeRemoteControlBeginSeekingForward = 108,
    UIEventSubtypeRemoteControlEndSeekingForward   = 109,
} UIEventSubtype;
```

Constants

`UIEventSubtypeNone`

The event has no subtype. This is the subtype for events of the `UIEventTypeTouches` (page 266) general type.

Available in iOS 3.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeMotionShake`

The event is related to the user shaking the device. It is a subtype for the `UIEventTypeMotion` (page 266) general event type.

Available in iOS 3.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlPlay`

A remote-control event for playing audio or video. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlPause`

A remote-control event for pausing audio or video. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlStop`

A remote-control event for stopping audio or video from playing. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlTogglePlayPause`

A remote-control event for toggling audio or video between play and pause. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlNextTrack`

A remote-control event for skipping to the next audio or video track. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlPreviousTrack`
A remote-control event for skipping to the previous audio or video track. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlBeginSeekingBackward`
A remote-control event to start seeking backward through the audio or video medium. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlEndSeekingBackward`
A remote-control event to end seeking backward through the audio or video medium. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlBeginSeekingForward`
A remote-control event to start seeking forward through the audio or video medium. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

`UIEventSubtypeRemoteControlEndSeekingForward`
A remote-control event to end seeking forward through the audio or video medium. It is a subtype of the `UIEventTypeRemoteControl` (page 266) general event type.

Available in iOS 4.0 and later.

Declared in `UIEvent.h`.

Discussion
You can obtain the subtype of an event from the `subtype` (page 263) property.

Availability
Available in iOS 3.0 and later.

Declared In
`UIEvent.h`

UIFont Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIFont.h UIInterface.h
Related sample code	aurioTouch BonjourWeb ScrollViewSuite WiTap

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 275)
Creates and returns a font object for the specified font name and size.

`- fontWithSize:` (page 278)
Returns a font object that is the same as the receiver but which has the specified size instead.

Creating System Fonts

`+ systemFontOfSize:` (page 277)
Returns the font object used for standard interface items in the specified size.

`+ boldSystemFontOfSize:` (page 274)
Returns the font object used for standard interface items that are rendered in boldface type in the specified size.

`+ italicSystemFontOfSize:` (page 276)
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 274)
Returns an array of font family names available on the system.

`+ fontNamesForFamilyName:` (page 275)
Returns an array of font names available in a particular font family.

Getting Font Name Attributes

`familyName` (page 272) *property*
The font family name. (read-only)

`fontName` (page 272) *property*
The font face name. (read-only)

Getting Font Metrics

`pointSize` (page 273) *property*
The receiver's point size, or the effective vertical point size for a font with a nonstandard matrix. (read-only)

`ascender` (page 271) *property*
The top y-coordinate, offset from the baseline, of the receiver's longest ascender. (read-only)

`descender` (page 272) *property*
The bottom y-coordinate, offset from the baseline, of the receiver's longest descender. (read-only)

`leading` (page 273) *property*
The receiver's leading information. (read-only) (Deprecated. Use the `lineHeight` (page 273) property instead.)

`capHeight` (page 271) *property*
The receiver's cap height information. (read-only)

`xHeight` (page 273) *property*
The x-height of the receiver. (read-only)

`lineHeight` (page 273) *property*
The height of text lines (measured in points). (read-only)

Getting System Font Information

`+ labelFontSize` (page 276)
Returns the standard font size used for labels.

`+ buttonFontSize` (page 274)
Returns the standard font size used for buttons.

`+ smallSystemFontSize` (page 277)
Returns the size of the standard small system font.

`+ systemFontSize` (page 277)
Returns the size of the standard system font.

Properties

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

ascender

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 iOS 2.0 and later.

See Also
`@property descender` (page 272)

Declared In
`UIFont.h`

capHeight

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 iOS 2.0 and later.

Declared In
`UIFont.h`

descender

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 iOS 2.0 and later.

See Also
`@property ascender` (page 271)

Declared In
`UIFont.h`

familyName

The font 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 iOS 2.0 and later.

Declared In
`UIFont.h`

fontName

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 iOS 2.0 and later.

Declared In
`UIFont.h`

leading

The receiver's leading information. (read-only) (Deprecated in iOS 4.0. Use the `lineHeight` (page 273) property instead.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 4.0.

Declared In
`UIFont.h`

lineHeight

The height of text lines (measured in points). (read-only)

```
@property(nonatomic,readonly) CGFloat lineHeight
```

Availability
Available in iOS 4.0 and later.

Declared In
`UIFont.h`

pointSize

The 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 iOS 2.0 and later.

Declared In
`UIFont.h`

xHeight

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 iOS 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 iOS 2.0 and later.

Related Sample Code
aurioTouch
ScrollViewSuite
WiTap

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 iOS 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 275) 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 iOS 2.0 and later.

See Also
`+ fontNamesForFamilyName:` (page 275)

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 275) method to retrieve an actual font object.

Availability
Available in iOS 2.0 and later.

See Also
`+ familyNames` (page 274)
`+ fontWithName:size:` (page 275)

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 275) method to retrieve the specific font names for a given font family.

Availability
Available in iOS 2.0 and later.

See Also
`+ familyNames` (page 274)
`+ fontNamesForFamilyName:` (page 275)

Related Sample Code
ScrollViewSuite

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
BonjourWeb

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 iOS 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 iOS 2.0 and later.

Declared In
`UIFont.h`

UIGestureRecognizer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIGestureRecognizer.h UIGestureRecognizerSubclass.h
Companion guide	Event Handling Guide for iOS
Related sample code	ScrollViewSuite SimpleGestureRecognizers

Overview

`UIGestureRecognizer` is an abstract base class for concrete gesture-recognizer classes. A gesture-recognizer object (or, simply, a gesture recognizer) decouples the logic for recognizing a gesture and acting on that recognition. When one of these objects recognizes a common gesture or, in some cases, a change in the gesture, it sends an action message to each designated target object.

The concrete subclasses of `UIGestureRecognizer` are the following:

```
UITapGestureRecognizer
UIPinchGestureRecognizer
UIRotationGestureRecognizer
UISwipeGestureRecognizer
UIPanGestureRecognizer
UILongPressGestureRecognizer
```

The `UIGestureRecognizer` class defines a set of common behaviors that can be configured for all concrete gesture recognizers. It can also communicate with its delegate (an object that adopts the `UIGestureRecognizerDelegate` protocol), thereby enabling finer-grained customization of some behaviors.

A gesture recognizer operates on touches hit-tested to a specific view and all of that view's subviews. It thus must be associated with that view. To make that association you must call the `UIView` method `addGestureRecognizer:` (page 721). A gesture recognizer does not participate in the view's responder chain.

A gesture recognizer has one or more target-action pairs associated with it. If there are multiple target-action pairs, they are discrete, and not cumulative. Recognition of a gesture results in the dispatch of an action message to a target for each of those pairs. The action methods invoked must conform to one of the following signatures:

```
- (void)handleGesture;
- (void)handleGesture:(UIGestureRecognizer *)gestureRecognizer;
```

Methods conforming to the latter signature permit the target in some cases to query the gesture recognizer sending the message for additional information. For example, the target could ask a `UIRotationGestureRecognizer` object for the angle of rotation (in radians) since the last invocation of the action method for this gesture. Clients of gesture recognizers can also ask for the location of a gesture by calling `locationInView:` (page 289) or `locationOfTouch:inView:` (page 290).

The gesture interpreted by a gesture recognizer can be either discrete or continuous. A discrete gesture, such as a double tap, occurs but once in a multi-touch sequence and results in a single action sent. However, when a gesture recognizer interprets a continuous gesture such as a rotation gesture, it sends an action message for each incremental change until the multi-touch sequence concludes.

A window delivers touch events to a gesture recognizer before it delivers them to the hit-tested view attached to the gesture recognizer. Generally, if a gesture recognizer analyzes the stream of touches in a multi-touch sequence and does not recognize its gesture, the view receives the full complement of touches. If a gesture recognizer recognizes its gesture, the remaining touches for the view are cancelled. The usual sequence of actions in gesture recognition follows a path determined by default values of the `cancelsTouchesInView` (page 283), `delaysTouchesBegan` (page 284), `delaysTouchesEnded` (page 284) properties:

- `cancelsTouchesInView`—If a gesture recognizer recognizes its gesture, it unbinds the remaining touches of that gesture from their view (so the window won't deliver them). The window cancels the previously delivered touches with a (`touchesCancelled:withEvent:` (page 467)) message. If a gesture recognizer doesn't recognize its gesture, the view receives all touches in the multi-touch sequence.
- `delaysTouchesBegan`—As long as a gesture recognizer, when analyzing touch events, has not failed recognition of its gesture, the window withholds delivery of touch objects in the `UITouchPhaseBegan` (page 682) phase to the attached view. If the gesture recognizer subsequently recognizes its gesture, the view does not receive these touch objects. If the gesture recognizer does not recognize its gesture, the window delivers these objects in an invocation of the view's `touchesBegan:withEvent:` (page 467) method (and possibly a follow-up `touchesMoved:withEvent:` (page 468) invocation to inform it of the touches current location).
- `delaysTouchesEnded`—As long as a gesture recognizer, when analyzing touch events, has not failed recognition of its gesture, the window withholds delivery of touch objects in the `UITouchPhaseEnded` (page 682) phase to the attached view. If the gesture recognizer subsequently recognizes its gesture, the touches are cancelled (in a `touchesCancelled:withEvent:` (page 467) message). If the gesture recognizer does not recognize its gesture, the window delivers these objects in an invocation of the view's `touchesEnded:withEvent:` (page 468) method.

Note that "recognize" in the above descriptions does not necessarily equate to a transition to the Recognized state.

Subclassing Notes

You may create a subclass that UIGestureRecognizer that recognizes a distinctive gesture—for example, a "check mark" gesture. If you are going to create such a concrete gesture recognizer, be sure to import the `UIGestureRecognizerSubclass.h` header file. This header declares all the methods and properties a subclass must either override, call, or reset.

Gesture recognizers operate within a predefined state machine, transitioning to subsequent states as they handle multi-touch events. The states and their possible transitions differ for continuous and discrete gestures. All gesture recognizers begin a multi-touch sequence in the Possible state (`UIGestureRecognizerStatePossible` (page 296)). Discrete gestures transition from Possible to either Recognized (`UIGestureRecognizerStateRecognized` (page 297)) or Failed (`UIGestureRecognizerStateFailed` (page 297)), depending on whether they successfully interpret the gesture or not. If the gesture recognizer transitions to Recognized, it sends its action message to its target.

For continuous gestures, the state transitions a gesture recognizer might make are more numerous, as indicated in the following diagram:

Possible ----> Began ----> [Changed] ----> Cancelled
Possible ----> Began ----> [Changed] ----> Ended

The Changed state is optional and may occur multiple times before the Cancelled or Ended state is reached. The gesture recognizer sends action messages at each state transition. Thus for a continuous gesture such as a pinch, action messages are sent as the two fingers move toward or away from each other. The `enum` constants representing these states are of type `UIGestureRecognizerState` (page 296). (Note that the constants for Recognized and Ended states are synonymous.)

Subclasses must set the `state` (page 286) property to the appropriate value when they transition between states.

Methods to Override

The methods that subclass must override are described in "Methods For Subclasses" (page 283). They must also periodically reset the `state` (page 286) property (as described above) and may call the `ignoreTouch:forEvent:` (page 288) method.

Special Considerations

The `state` (page 286) property is declared in `UIGestureRecognizer.h` as being read-only. This property declaration is intended for clients of gesture recognizers. Subclasses of `UIGestureRecognizer` must import `UIGestureRecognizerSubclass.h`. This header file contains a redeclaration of `state` that makes it read-write.

Tasks

Initializing a Gesture Recognizer

- `initWithTarget:action:` (page 289)
  Initializes an allocated gesture-recognizer object with a target and an action selector.

Adding and Removing Targets and Actions

- `addTarget:action:` (page 287)
  Adds a target and an action to a gesture-recognizer object.
- `removeTarget:action:` (page 291)
  Removes a target and an action from a gesture-recognizer object.

Getting the Touches and Location of a Gesture

- `locationInView:` (page 289)
  Returns the point computed as the location in a given view of the gesture represented by the receiver.
- `locationOfTouch:inView:` (page 290)
  Returns the location of one of the gesture's touches in the local coordinate system of a given view.
- `numberOfTouches` (page 291)
  Returns the number of touches involved in the gesture represented by the receiver.

Getting the Recognizer's State and View

`state` (page 286) *property*
The current state of the gesture recognizer. (read-only)

`view` (page 286) *property*
The view the gesture recognizer is attached to. (read-only)

`enabled` (page 285) *property*
A Boolean property that indicates whether the gesture recognizer is enabled.

Canceling and Delaying Touches

`cancelsTouchesInView` (page 283) *property*
A Boolean value affecting whether touches are delivered to a view when a gesture is recognized.

`delaysTouchesBegan` (page 284) *property*
A Boolean value determining whether the receiver delays sending touches in a begin phase to its view.

`delaysTouchesEnded` (page 284) *property*
A Boolean value determining whether the receiver delays sending touches in a end phase to its view.

Specifying Dependencies Between Gesture Recognizers

- `requireGestureRecognizerToFail:` (page 291)
  Creates a dependency relationship between the receiver and another gesture recognizer.

Setting and Getting the Delegate

`delegate` (page 285) *property*
The delegate of the gesture recognizer.

Methods For Subclasses

The `UIGestureRecognizerSubclass.h` header file contains a class extension that declares methods intended to be called or overridden *only* by subclasses of `UIGestureRecognizer`. Clients that merely use concrete subclasses of `UIGestureRecognizer` must never call these methods (except for those noted).

- `touchesBegan:withEvent:` (page 292)
  Sent to the receiver when one or more fingers touch down in the associated view.
- `touchesMoved:withEvent:` (page 295)
  Sent to the receiver when one or more fingers move in the associated view.
- `touchesEnded:withEvent:` (page 294)
  Sent to the receiver when one or more fingers lift from the associated view.
- `touchesCancelled:withEvent:` (page 293)
  Sent to the receiver when a system event (such as a low-memory warning) cancels a touch event.
- `reset` (page 292)
  Overridden to reset internal state when a gesture is recognized.
- `ignoreTouch:forEvent:` (page 288)
  Tells the gesture recognizer to ignore a specific touch of the given event.
- `canBePreventedByGestureRecognizer:` (page 287)
  Overridden to indicate that the specified gesture recognizer can prevent the receiver from recognizing a gesture.
- `canPreventGestureRecognizer:` (page 288)
  Overridden to indicate that the receiver can prevent the specified gesture recognizer from recognizing its gesture.

Properties

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

cancelsTouchesInView

A Boolean value affecting whether touches are delivered to a view when a gesture is recognized.

```
@property(nonatomic) BOOL cancelsTouchesInView
```

Discussion
When this property is `YES` (the default) and the receiver recognizes its gesture, the touches of that gesture that are pending are not delivered to the view and previously delivered touches are cancelled through a `touchesCancelled:withEvent:` (page 467) message sent to the view. If a gesture recognizer doesn't recognize its gesture or if the value of this property is `NO`, the view receives all touches in the multi-touch sequence.

Availability
Available in iOS 3.2 and later.

See Also
`@property delaysTouchesBegan` (page 284)
`@property delaysTouchesEnded` (page 284)

Declared In
`UIGestureRecognizer.h`

delaysTouchesBegan

A Boolean value determining whether the receiver delays sending touches in a begin phase to its view.

```
@property(nonatomic) BOOL delaysTouchesBegan
```

Discussion
When the value of this property is `NO` (the default) and the receiver is analyzing touch events, the window suspends delivery of touch objects in the `UITouchPhaseBegan` (page 682) phase to the attached view. If the gesture recognizer subsequently recognizes its gesture, these touch objects are discarded. If the gesture recognizer, however, does not recognize its gesture, the window delivers these objects to the view in a `touchesBegan:withEvent:` (page 467) message (and possibly a follow-up `touchesMoved:withEvent:` (page 468) message to inform it of the touches current location). Set this property to `YES` to prevent views from processing any touches in the `UITouchPhaseBegan` (page 682) phase that may be recognized as part of this gesture.

Availability
Available in iOS 3.2 and later.

See Also
`@property cancelsTouchesInView` (page 283)
`@property delaysTouchesEnded` (page 284)

Declared In
`UIGestureRecognizer.h`

delaysTouchesEnded

A Boolean value determining whether the receiver delays sending touches in a end phase to its view.

```
@property(nonatomic) BOOL delaysTouchesEnded
```

Discussion
When the value of this property is `YES` (the default) and the receiver is analyzing touch events, the window suspends delivery of touch objects in the `UITouchPhaseEnded` (page 682) phase to the attached view. If the gesture recognizer subsequently recognizes its gesture, these touch objects are cancelled (via a `touchesCancelled:withEvent:` (page 467) message). If the gesture recognizer does not recognize its gesture, the window delivers these objects in an invocation of the view's `touchesEnded:withEvent:` (page 468) method. Set this property to `NO` to have touch objects in the `UITouchPhaseEnded` (page 682) delivered to the view while the gesture recognizer is analyzing the same touches.

Availability
Available in iOS 3.2 and later.

See Also
`@property cancelsTouchesInView` (page 283)
`@property delaysTouchesBegan` (page 284)

Declared In
`UIGestureRecognizer.h`

delegate

The delegate of the gesture recognizer.

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

Discussion
The gesture recognizer maintains a weak reference to its delegate. The delegate must adopt the `UIGestureRecognizerDelegate` protocol and implement one or more of its methods.

Availability
Available in iOS 3.2 and later.

Related Sample Code
SimpleGestureRecognizers

Declared In
`UIGestureRecognizer.h`

enabled

A Boolean property that indicates whether the gesture recognizer is enabled.

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

Discussion
Disables a gesture recognizers so it does not receive touches. The default value is `YES`. If you change this property to `NO` while a gesture recognizer is currently recognizing a gesture, the gesture recognizer transitions to a cancelled state.

Availability
Available in iOS 3.2 and later.

Declared In
UIGestureRecognizer.h

state

The current state of the gesture recognizer. (read-only)

```
@property(nonatomic, readonly) UIGestureRecognizerState state
```

Discussion
The possible states a gesture recognizer can be in are represented by the constants of type UIGestureRecognizerState (page 296). Some of these states are not applicable to discrete gestures. The read-only version of the state property is intended for clients of a gesture-recognizer class and not subclasses.

Special Considerations

Subclasses of UIGestureRecognizer must use a read-write version of the state property. They get this redeclaration when they import the UIGestureRecognizerSubclass.h header file:

```
@property(nonatomic,readwrite) UIGestureRecognizerState state;
```

Recognizers for discrete gestures transition from UIGestureRecognizerStatePossible (page 296) to UIGestureRecognizerStateFailed (page 297) or UIGestureRecognizerStateRecognized (page 297). Recognizers for continuous gesture transition from UIGestureRecognizerStatePossible to these phases in the given order: UIGestureRecognizerStateBegan (page 296), UIGestureRecognizerStateChanged (page 296), and UIGestureRecognizerStateEnded (page 296). If, however, they receive a cancellation touch, they should transition to UIGestureRecognizerStateCancelled (page 296). If recognizers for continuous gestures cannot interpret a multi-touch sequence as their gesture, they transition to UIGestureRecognizerStateFailed (page 297).

Availability
Available in iOS 3.2 and later.

Declared In
UIGestureRecognizer.h

view

The view the gesture recognizer is attached to. (read-only)

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

Discussion
You attach (or add) a gesture recognizer to a UIView object using the addGestureRecognizer: (page 721) method.

Availability
Available in iOS 3.2 and later.

Declared In
UIGestureRecognizer.h

Instance Methods

addTarget:action:

Adds a target and an action to a gesture-recognizer object.

```
- (void)addTarget:(id)target action:(SEL)action
```

Parameters

target

An object that is a recipient of action messages sent by the receiver when the represented gesture occurs. `nil` is not a valid value.

action

A selector identifying a method of a target to be invoked by the action message. `NULL` is not a valid value.

Discussion
You may call this method multiple times to specify multiple target-action pairs.

Availability
Available in iOS 3.2 and later.

See Also
- `removeTarget:action:` (page 291)
- `initWithTarget:action:` (page 289)

Declared In
`UIGestureRecognizer.h`

canBePreventedByGestureRecognizer:

Overridden to indicate that the specified gesture recognizer can prevent the receiver from recognizing a gesture.

```
- (BOOL)canBePreventedByGestureRecognizer:(UIGestureRecognizer
    *)preventingGestureRecognizer
```

Parameters

preventingGestureRecognizer

An instance of a subclass of `UIGestureRecognizer`.

Return Value
`YES` to indicate that *preventingGestureRecognizer* can block the receiver from recognizing its gesture, otherwise `NO`.

Discussion
Overriding these methods enables the same behavior as implementing the `UIGestureRecognizerDelegate` methods `gestureRecognizerShouldBegin:` (page 861) and `gestureRecognizer:shouldReceiveTouch:` (page 860). However, by overriding them, subclasses can define class-wide prevention rules. For example, a `UITapGestureRecognizer` object never prevents another `UITapGestureRecognizer` object with a higher tap count.

Availability
Available in iOS 3.2 and later.

See Also
- `canPreventGestureRecognizer:` (page 288)

Declared In
`UIGestureRecognizerSubclass.h`

canPreventGestureRecognizer:

Overridden to indicate that the receiver can prevent the specified gesture recognizer from recognizing its gesture.

```
- (BOOL)canPreventGestureRecognizer:(UIGestureRecognizer *)preventedGestureRecognizer
```

Parameters

preventedGestureRecognizer
An instance of a subclass of UIGestureRecognizer.

Return Value
`YES` to indicate that the receiver can block *preventedGestureRecognizer* from recognizing its gesture, otherwise `NO`.

Discussion
Overriding these methods enables the same behavior as implementing the `UIGestureRecognizerDelegate` methods `gestureRecognizerShouldBegin:` (page 861) and `gestureRecognizer:shouldReceiveTouch:` (page 860). However, by overriding them, subclasses can define class-wide prevention rules. For example, a `UITapGestureRecognizer` object never prevents another `UITapGestureRecognizer` object with a higher tap count.

Availability
Available in iOS 3.2 and later.

See Also
- `canBePreventedByGestureRecognizer:` (page 287)

Declared In
`UIGestureRecognizerSubclass.h`

ignoreTouch:forEvent:

Tells the gesture recognizer to ignore a specific touch of the given event.

```
- (void)ignoreTouch:(UITouch *)touch forEvent:(UIEvent *)event
```

Parameters

touch
A `UITouch` object that is part of the current multi-touch sequence and associated with *event*.

event
A `UIEvent` object that includes a reference to *touch*.

Discussion
If a touch isn't part of this gesture you may pass it to this method, causing it to be ignored. `UIGestureRecognizer` does not cancel ignored touches on the associated view even if `cancelsTouchesInView` (page 283) is `YES`. This method is intended to be called, not overridden.

Availability
Available in iOS 3.2 and later.

Declared In
`UIGestureRecognizerSubclass.h`

initWithTarget:action:

Initializes an allocated gesture-recognizer object with a target and an action selector.

```
- (id)initWithTarget:(id)target action:(SEL)action
```

Parameters

target
An object that is the recipient of action messages sent by the receiver when it recognizes a gesture. `nil` is not a valid value.

action
A selector that identifies the method implemented by the target to handle the gesture recognized by the receiver. The action selector must conform to the signature described in the class overview. `NULL` is not a valid value.

Return Value
An initialized instance of a concrete `UIGestureRecognizer` subclass or `nil` if an error occurred in the attempt to initialize the object.

Discussion
This method is the designated initializer. After creating the gesture recognizer, you may associate other target-action pairs with it by calling `addTarget:action:` (page 287).

Availability
Available in iOS 3.2 and later.

See Also
- `addTarget:action:` (page 287)
- `removeTarget:action:` (page 291)

Related Sample Code
ScrollViewSuite
SimpleGestureRecognizers

Declared In
`UIGestureRecognizer.h`

locationInView:

Returns the point computed as the location in a given view of the gesture represented by the receiver.

```
- (CGPoint)locationInView:(UIView *)view
```

Parameters

view

A `UIView` object on which the gesture took place. Specify `nil` to indicate the window.

Return Value

A point in the local coordinate system of *view* that identifies the location of the gesture. If `nil` is specified for *view*, the method returns the gesture location in the window's base coordinate system.

Discussion

The returned value is a generic single-point location for the gesture computed by the UIKit framework. It is usually the centroid of the touches involved in the gesture. For objects of the `UISwipeGestureRecognizer` and `UITapGestureRecognizer` classes, the location returned by this method has a significance special to the gesture. This significance is documented in the reference for those classes.

Availability

Available in iOS 3.2 and later.

See Also

`- locationOfTouch:inView:` (page 290)

Related Sample Code

ScrollViewSuite

SimpleGestureRecognizers

Declared In

`UIGestureRecognizer.h`

locationOfTouch:inView:

Returns the location of one of the gesture's touches in the local coordinate system of a given view.

```
- (CGPoint)locationOfTouch:(NSUInteger)touchIndex inView:(UIView *)view
```

Parameters

touchIndex

The index of a `UITouch` object in a private array maintained by the receiver. This touch object represents a touch of the current gesture.

view

A `UIView` object on which the gesture took place. Specify `nil` to indicate the window.

Return Value

A point in the local coordinate system of *view* that identifies the location of the touch. If `nil` is specified for *view*, the method returns the touch location in the window's base coordinate system.

Availability

Available in iOS 3.2 and later.

See Also

`- locationInView:` (page 289)

Declared In

`UIGestureRecognizer.h`

numberOfTouches

Returns the number of touches involved in the gesture represented by the receiver.

```
- (NSUInteger)numberOfTouches
```

Return Value
The number of `UITouch` objects in a private array maintained by the receiver. Each of these objects represents a touch in the current gesture.

Discussion
Using the value returned by this method in a loop, you can ask for the location of individual touches using the `locationOfTouch:inView:` (page 290) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIGestureRecognizer.h`

removeTarget:action:

Removes a target and an action from a gesture-recognizer object.

```
- (void)removeTarget:(id)target action:(SEL)action
```

Parameters

target
An object that currently is a recipient of action messages sent by the receiver when the represented gesture occurs. Specify `nil` if you want to remove all targets from the receiver.

action
A selector identifying a method of a target to be invoked by the action message. Specify `NULL` if you want to remove all actions from the receiver.

Discussion
Calling this method removes the specified target-action pair. Passing `nil` for *target* matches all targets and passing `NULL` for *action* matches all actions.

Availability
Available in iOS 3.2 and later.

See Also
- `addTarget:action:` (page 287)
- `initWithTarget:action:` (page 289)

Declared In
`UIGestureRecognizer.h`

requireGestureRecognizerToFail:

Creates a dependency relationship between the receiver and another gesture recognizer.

```
- (void)requireGestureRecognizerToFail:(UIGestureRecognizer *)otherGestureRecognizer
```

Parameters

otherGestureRecognizer

Another gesture-recognizer object (an instance of a subclass of UIGestureRecognizer).

Discussion

This method creates a relationship with another gesture recognizer that delays the receiver's transition out of `UIGestureRecognizerStatePossible` (page 296). The state that the receiver transitions to depends on what happens with *otherGestureRecognizer*:

- If *otherGestureRecognizer* transitions to `UIGestureRecognizerStateFailed` (page 297), the receiver transitions to its normal next state.
- if *otherGestureRecognizer* transitions to `UIGestureRecognizerStateRecognized` (page 297) or `UIGestureRecognizerStateBegan` (page 296), the receiver transitions to `UIGestureRecognizerStateFailed` (page 297).

An example where this method might be called is when you want a single-tap gesture require that a double-tap gesture fail.

Availability

Available in iOS 3.2 and later.

Declared In

`UIGestureRecognizer.h`

reset

Overridden to reset internal state when a gesture is recognized.

```
- (void)reset
```

Discussion

The runtime calls this method after the gesture-recognizer state has been set to `UIGestureRecognizerStateEnded` (page 296) or `UIGestureRecognizerStateRecognized` (page 297). Subclasses should reset any internal state in preparation for a new attempt at gesture recognition. After this method is called, the runtime ignores all remaining active touches; that is, the gesture recognizer receives no further updates for touches that have begun but haven't ended.

Availability

Available in iOS 3.2 and later.

See Also

`@property state` (page 286)

Declared In

`UIGestureRecognizerSubclass.h`

touchesBegan:withEvent:

Sent to the receiver when one or more fingers touch down in the associated view.

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
```

Parameters

touches

A set of `UITouch` instances in the event represented by *event* that represent the touches in the `UITouchPhaseBegan` (page 682) phase.

event

A `UIEvent` object representing the event to which the touches belong.

Discussion

This method has the same exact signature as the corresponding one declared by `UIResponder`. Through this method a gesture recognizer receives touch objects (in their `UITouchPhaseBegan` (page 682) phase) before the view attached to the gesture recognizer receives them. `UIGestureRecognizer` objects are not in the responder chain, yet observe touches hit-tested to their view and their view's subviews. After observation, the delivery of touch objects to the attached view, or their disposition otherwise, is affected by the `cancelsTouchesInView` (page 283), `delaysTouchesBegan` (page 284), and `delaysTouchesEnded` (page 284) properties.

If the gesture recognizer is interpreting a continuous gesture, it should set its state to `UIGestureRecognizerStateBegan` (page 296) upon receiving this message. If at any point in its handling of the touch objects the gesture recognizer determines that the multi-touch event sequence is not its gesture, it should set it state to `UIGestureRecognizerStateCancelled` (page 296).

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` (page 704) property of the attached view instance to `YES`.

Availability

Available in iOS 3.2 and later.

See Also

- `touchesMoved:withEvent:` (page 295)
- `touchesEnded:withEvent:` (page 294)
- `touchesCancelled:withEvent:` (page 293)

Declared In

`UIGestureRecognizerSubclass.h`

touchesCancelled:withEvent:

Sent to the receiver when a system event (such as a low-memory warning) cancels a touch event.

```
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
```

Parameters

touches

A set of `UITouch` instances in the event represented by *event* that represent the touches in the `UITouchPhaseCancelled` (page 683) phase.

event

A `UIEvent` object representing the event to which the touches belong.

Discussion

This method has the same exact signature as the corresponding one declared by `UIResponder`. Through this method a gesture recognizer receives touch objects (in their `UITouchPhaseCancelled` (page 683) phase) before the view attached to the gesture recognizer receives them. `UIGestureRecognizer` objects are not in the responder chain, yet observe touches hit-tested to their view and their view's subviews. After

observation, the delivery of touch objects to the attached view, or their disposition otherwise, is affected by the `cancelsTouchesInView` (page 283), `delaysTouchesBegan` (page 284), and `delaysTouchesEnded` (page 284) properties.

Upon receiving this message, the gesture recognizer for a continuous gesture should set its state to `UIGestureRecognizerStateCancelled` (page 296); a gesture recognizer for a discrete gesture should set its state to `UIGestureRecognizerStateFailed` (page 297).

Availability
Available in iOS 3.2 and later.

See Also
- `touchesBegan:withEvent:` (page 292)
- `touchesMoved:withEvent:` (page 295)
- `touchesEnded:withEvent:` (page 294)

Declared In
`UIGestureRecognizerSubclass.h`

touchesEnded:withEvent:

Sent to the receiver when one or more fingers lift from the associated view.

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
```

Parameters

touches
A set of `UITouch` instances in the event represented by *event* that represent the touches in the `UITouchPhaseEnded` (page 682) phase.

event
A `UIEvent` object representing the event to which the touches belong.

Discussion
This method has the same exact signature as the corresponding one declared by `UIResponder`. Through this method a gesture recognizer receives touch objects (in their `UITouchPhaseEnded` (page 682) phase) before the view attached to the gesture recognizer receives them. `UIGestureRecognizer` objects are not in the responder chain, yet observe touches hit-tested to their view and their view's subviews. After observation, the delivery of touch objects to the attached view, or their disposition otherwise, is affected by the `cancelsTouchesInView` (page 283), `delaysTouchesBegan` (page 284), and `delaysTouchesEnded` (page 284) properties.

If the gesture recognizer is interpreting a continuous gesture, it should set its state to `UIGestureRecognizerStateEnded` (page 296) upon receiving this message. If it is interpreting a discrete gesture, it should set its state to `UIGestureRecognizerStateRecognized` (page 297). If at any point in its handling of the touch objects the gesture recognizer determines that the multi-touch event sequence is not its gesture, it should set it state to `UIGestureRecognizerStateCancelled` (page 296).

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` (page 704) property of the attached view instance to `YES`.

Availability
Available in iOS 3.2 and later.

See Also
- `touchesBegan:withEvent:` (page 292)
- `touchesMoved:withEvent:` (page 295)
- `touchesCancelled:withEvent:` (page 293)

Declared In
`UIGestureRecognizerSubclass.h`

touchesMoved:withEvent:

Sent to the receiver when one or more fingers move in the associated view.

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
```

Parameters

touches

A set of `UITouch` instances in the event represented by *event* that represent touches in the `UITouchPhaseMoved` (page 682) phase.

event

A `UIEvent` object representing the event to which the touches belong.

Discussion
This method has the same exact signature as the corresponding one declared by `UIResponder`. Through this method a gesture recognizer receives touch objects (in their `UITouchPhaseMoved` (page 682) phase) before the view attached to the gesture recognizer receives them. `UIGestureRecognizer` objects are not in the responder chain, yet observe touches hit-tested to their view and their view's subviews. After observation, the delivery of touch objects to the attached view, or their disposition otherwise, is affected by the `cancelsTouchesInView` (page 283), `delaysTouchesBegan` (page 284), and `delaysTouchesEnded` (page 284) properties.

If the gesture recognizer is interpreting a continuous gesture, it should set its state to `UIGestureRecognizerStateChanged` (page 296) upon receiving this message. If at any point in its handling of the touch objects the gesture recognizer determines that the multi-touch event sequence is not its gesture, it should set it state to `UIGestureRecognizerStateCancelled` (page 296) .

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` (page 704) property of the attached view instance to `YES`.

Availability
Available in iOS 3.2 and later.

See Also
- `touchesBegan:withEvent:` (page 292)
- `touchesEnded:withEvent:` (page 294)
- `touchesCancelled:withEvent:` (page 293)

Declared In
`UIGestureRecognizerSubclass.h`

Constants

UIGestureRecognizerState

The current state a gesture recognizer is in.

```
typedef enum {
    UIGestureRecognizerStatePossible,

    UIGestureRecognizerStateBegan,
    UIGestureRecognizerStateChanged,
    UIGestureRecognizerStateEnded,
    UIGestureRecognizerStateCancelled,

    UIGestureRecognizerStateFailed,

    UIGestureRecognizerStateRecognized = UIGestureRecognizerStateEnded
} UIGestureRecognizerState;
```

Constants

`UIGestureRecognizerStatePossible`
The gesture recognizer has not yet recognized its gesture, but may be evaluating touch events. This is the default state.

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

`UIGestureRecognizerStateBegan`
The gesture recognizer has received touch objects recognized as a continuous gesture. It sends its action message (or messages) at the next cycle of the run loop.

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

`UIGestureRecognizerStateChanged`
The gesture recognizer has received touches recognized as a change to a continuous gesture. It sends its action message (or messages) at the next cycle of the run loop.

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

`UIGestureRecognizerStateEnded`
The gesture recognizer has received touches recognized as the end of a continuous gesture. It sends its action message (or messages) at the next cycle of the run loop and resets its state to `UIGestureRecognizerStatePossible` (page 296).

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

`UIGestureRecognizerStateCancelled`
The gesture recognizer has received touches resulting in the cancellation of a continuous gesture. It sends its action message (or messages) at the next cycle of the run loop and resets its state to `UIGestureRecognizerStatePossible` (page 296).

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

`UIGestureRecognizerStateFailed`

The gesture recognizer has received a multi-touch sequence that it cannot recognize as its gesture. No action message is sent and the gesture recognizer is reset to `UIGestureRecognizerStatePossible` (page 296).

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

`UIGestureRecognizerStateRecognized`

The gesture recognizer has received a multi-touch sequence that it recognizes as its gesture. It sends its action message (or messages) at the next cycle of the run loop and resets its state to `UIGestureRecognizerStatePossible` (page 296).

Available in iOS 3.2 and later.

Declared in `UIGestureRecognizer.h`.

Discussion
Gesture recognizers recognize a discrete event such as a tap or a swipe but do not report changes within the gesture. In other words, discrete gestures do not transition through the Began and Changed states and cannot fail or be cancelled.

Availability
Available in iOS 3.2 and later.

Declared In
`UIGestureRecognizer.h`

UIImage Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIKit/UIImage.h UIKit/UIInterface.h
Related sample code	AddMusic aurioTouch GKRocket GKTank ScrollViewSuite

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 1045) and `UIImageJPEGRepresentation` (page 1044) functions.

The system uses image objects to represent still pictures taken with the camera on supported devices. To take a picture, use the `UIImagePickerController` class. To save a picture to the Saved Photos album, use the `UIImageWriteToSavedPhotosAlbum` (page 1045) function.

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.

You should avoid creating `UIImage` objects that are greater than 1024 x 1024 in size. Besides the large amount of memory such an image would consume, you may run into problems when using the image as a texture in OpenGL ES or when drawing the image to a view or layer. This size restriction does not apply if you are performing code-based manipulations, such as resizing an image larger than 1024 x 1024 pixels by drawing it to a bitmap-backed graphics context. In fact, you may need to resize an image in this manner (or break it into several smaller images) in order to draw it to one of your views.

Supported Image Formats

Table 26-1 lists the file formats that can be read by the `UIImage` class.

Table 26-1 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 305)
Returns the image object associated with the specified filename.

Creating New Images

`+ imageWithContentsOfFile:` (page 307)
Creates and returns an image object by loading the image data from the file at the specified path.

`+ imageWithData:` (page 307)
Creates and returns an image object that uses the specified image data.

`+ imageWithCGImage:` (page 306)
Creates and returns an image object representing the specified Quartz image.

`+ imageWithCGImage:scale:orientation:` (page 306)
Creates and returns an image object with the specified scale and orientation factors.

`- stretchableImageWithLeftCapWidth:topCapHeight:` (page 312)
Creates and returns a new image object with the specified cap values.

Initializing Images

`- initWithContentsOfFile:` (page 311)
Initializes and returns the image object with the contents of the specified file.

`- initWithData:` (page 311)
Initializes and returns the image object with the specified data.

`- initWithCGImage:` (page 310)
Initializes and returns the image object with the specified Quartz image reference.

`- initWithCGImage:scale:orientation:` (page 310)
Initializes and returns an image object with the specified scale and orientation factors

Image Attributes

`imageOrientation` (page 303) *property*
The orientation of the receiver's image. (read-only)

`size` (page 304) *property*
The dimensions of the image, taking orientation into account. (read-only)

`scale` (page 303) *property*
The scale factor of the image. (read-only)

`CGImage` (page 302) *property*
The underlying Quartz image data. (read-only)

`leftCapWidth` (page 303) *property*
The horizontal end-cap size. (read-only)

`topCapHeight` (page 304) *property*
The vertical end-cap size. (read-only)

Drawing Images

`- drawAtPoint:` (page 308)
Draws the image at the specified point in the current context.

`- drawAtPoint:blendMode:alpha:` (page 309)
Draws the entire image at the specified point using the custom compositing options.

`- drawInRect:` (page 309)
Draws the entire image in the specified rectangle, scaling it as needed to fit.

`- drawInRect:blendMode:alpha:` (page 310)
Draws the entire image in the specified rectangle and using the specified compositing options.

`- drawAsPatternInRect:` (page 308)
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 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 iOS 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 313).

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`- stretchableImageWithLeftCapWidth:topCapHeight:` (page 312)

Declared In
`UIImage.h`

scale

The scale factor of the image. (read-only)

```
@property(nonatomic,readonly) CGFloat scale
```

Discussion
If you load an image from a file whose name includes the `@2x` modifier, the scale is set to `2.0`. If the filename does not include the modifier but is in the PNG or JPEG format and has an associated DPI value, a corresponding scale factor is computed and reflected in this property. You can also specify an explicit scale factor when initializing an image from a Core Graphics image. All other images are assumed to have a scale factor of `1.0`.

If you multiply the logical size of the image (stored in the `size` (page 304) property) by the value in this property, you get the dimensions of the image in pixels.

Availability
Available in iOS 4.0 and later.

Declared In
`UIImage.h`

size

The dimensions of the image, taking orientation into account. (read-only)

```
@property(nonatomic, readonly) CGSize size
```

Discussion
In iOS 4.0 and later, this value reflects the logical size of the image and is measured in points. In iOS 3.x and earlier, this value always reflects the dimensions of the image measured in pixels.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`- stretchableImageWithLeftCapWidth:topCapHeight:` (page 312)

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. 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.

On a device running iOS 4 or later, the behavior is identical if the device's screen has a scale of `1.0`. If the screen has a scale of `2.0`, this method first searches for an image file with the same filename with an `@2x` suffix appended to it. For example, if the file's name is `button`, it first searches for `button@2x`. If it finds a 2x, it loads that image and sets the `scale` property of the returned `UIImage` object to `2.0`. Otherwise, it loads the unmodified filename and sets the `scale` property to `1.0`. See *iOS Application Programming Guide* for more information on supporting images with different scale factors.

Special Considerations

On iOS 4 and later, the name of the file is not required to specify the filename extension. Prior to iOS 4, you must specify the filename extension.

Availability
Available in iOS 2.0 and later.

Related Sample Code
aurioTouch
GKRocket
GKTank
ScrollViewSuite
WiTap

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 iOS 2.0 and later.

Related Sample Code
aurioTouch

Declared In
UIImage.h

imageWithCGImage:scale:orientation:

Creates and returns an image object with the specified scale and orientation factors.

```
+ (UIImage *)imageWithCGImage:(CGImageRef)imageRef scale:(CGFloat)scale
    orientation:(UIImageOrientation)orientation
```

Parameters

imageRef
The Quartz image object.

scale
The scale factor to use when interpreting the image data. Specifying a scale factor of 1.0 results in an image whose size matches the pixel-based dimensions of the image. Applying a different scale factor changes the size of the image as reported by the size (page 304) property.

orientation
The orientation of the image data. You can use this parameter to specify any rotation factors applied to the image.

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 iOS 4.0 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In

`UIImage.h`

initWithCGImage:scale:orientation:

Initializes and returns an image object with the specified scale and orientation factors

```
- (id)initWithCGImage:(CGImageRef)imageRef scale:(CGFloat)scale
    orientation:(UIImageOrientation)orientation
```

Parameters

imageRef

The Quartz image object.

scale

The scale factor to assume when interpreting the image data. Applying a scale factor of 1.0 results in an image whose size matches the pixel-based dimensions of the image. Applying a different scale factor changes the size of the image as reported by the `size` (page 304) property.

orientation

The orientation of the image data. You can use this parameter to specify any rotation factors applied to the image.

Return Value

An initialized `UIImage` object, or `nil` if the method could not initialize the image from the specified data.

Availability

Available in iOS 4.0 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 not 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 iOS 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 iOS 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 want the entire image to be horizontally stretchable. For a discussion of how a non-zero value affects the image, see the `leftCapWidth` (page 303) property.

topCapHeight

The value to use for the top cap width. Specify `0` if you want the entire image to be vertically stretchable. For a discussion of how a non-zero value affects the image, see the `topCapHeight` (page 304) 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 iOS 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 iOS 2.0 and later.

Declared in `UIImage.h`.

`UIImageOrientationDown`

The image is rotated 180 degrees, as shown here. 

Available in iOS 2.0 and later.

Declared in `UIImage.h`.

`UIImageOrientationLeft`

The image is rotated 90 degrees counterclockwise, as shown here. 

Available in iOS 2.0 and later.

Declared in `UIImage.h`.

`UIImageOrientationRight`

The image is rotated 90 degrees clockwise, as shown here. 

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `UIImage.h`.

Declared In
`UIImage.h`

UIImagePickerController Class Reference

Inherits from	UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIImagePickerController.h

Overview

The `UIImagePickerController` class manages system-supplied user interfaces for choosing and taking pictures and movies on supported devices. The class manages user interactions and delivers the results of those interactions to the delegate object you've associated with the image picker.

To use the default image picker to handle user interactions, perform these steps:

1. Verify that the device is capable of picking content from the desired source. Do this calling the `isSourceTypeAvailable:` (page 326) class method.
2. Check which media types are available for the source type by calling the `availableMediaTypesForSourceType:` (page 324) class method. This lets you distinguish between a camera that can be used for video recording and one that can be used only for still images.
3. Tell the `UIImagePickerController` class which user interface to display. Do this by setting the `mediaTypes` (page 321) property.
4. Present the user interface.
5. When the user picks an image or movie, or cancels the operation, dismiss the image picker using your delegate object.

In addition to the default image picker, in iOS 3.1 and later you can manage user interactions yourself. To do this, provide an overlay view to display a custom picture-taking interface. This lets you initiate choosing and taking pictures and movies programmatically. Your custom overlay view can be displayed in addition to, or instead of, the default controls provided by the image picker interface.

To use this class, you must provide a delegate that conforms to the `UIImagePickerControllerDelegate` protocol. See *UIImagePickerControllerDelegate Protocol Reference*.

The default camera interface supports editing of previously-saved movies. Editing involves trimming from the start or end of the movie, then saving the trimmed movie.

In iOS 4.0 and later, you can provide custom controls to let the user adjust flash mode (on devices that have a flash LED), pick which camera to use (on devices that have a front and rear camera), and switch between still image and movie capture. You can also manage these settings programmatically.

In iOS 4.0 and later, on devices that have a flash LED, you can manipulate the flash directly to provide effects such as a strobe light. Present a picker interface set to use video capture mode. Then, turn the flash LED on or off by setting the `cameraFlashMode` (page 319) property to `UIImagePickerControllerCameraFlashModeOn` (page 331) or `UIImagePickerControllerCameraFlashModeOff` (page 331).

Movie capture has a default duration limit of 10 minutes but can be adjusted using the `videoMaximumDuration` (page 323) property. When a user taps the Share button to send a movie to MMS, MobileMe, YouTube, or another destination, an appropriate duration limit and an appropriate video quality are enforced.

To display an interface dedicated to movie editing, rather than one that also supports recording new movies, use the `UIVideoEditorController` class instead of this one. See *UIVideoEditorController Class Reference*.

Important: The `UIImagePickerController` class supports portrait mode only. This class is intended to be used as-is and does not support subclassing. The view hierarchy for this class is private and must not be modified, with one exception. In iOS 3.1 and later, you can assign a custom view to the `cameraOverlayView` (page 320) property and use that view to present additional information or manage the interactions between the camera interface and your code.

Tasks

Setting the Picker Source

`+ availableMediaTypesForSourceType:` (page 324)
Returns an array of the available media types for the specified source type.

`+ isSourceTypeAvailable:` (page 326)
Returns a Boolean value indicating whether the device supports picking media using the specified source type.

`sourceType` (page 322) *property*
The type of picker interface displayed by the controller.

Configuring the Picker

`allowsEditing` (page 318) *property*
A Boolean value indicating whether the user is allowed to edit a selected still image or movie.

`allowsImageEditing` (page 318) *property*
A Boolean value indicating whether the user is allowed to edit a selected image. (Deprecated. Use `allowsEditing` (page 318) instead.)

`delegate` (page 321) *property*
The image picker's delegate object.

`mediaTypes` (page 321) *property*
An array indicating the media types to be accessed by the picker interface.

Configuring the Video Capture Options

`videoQuality` (page 323) *property*
The video recording and transcoding quality.

`videoMaximumDuration` (page 323) *property*
The maximum duration, in seconds, for a video recording.

Customizing the Camera Controls

`showsCameraControls` (page 322) *property*
Indicates whether the image picker displays the default camera controls.

`cameraOverlayView` (page 320) *property*
The custom view to display on top of the default image picker interface.

`cameraViewTransform` (page 320) *property*
The transform to apply to the camera's preview image.

Capturing Still Images or Movies

- `takePicture` (page 327)
Captures a still image using the camera.

- `startVideoCapture` (page 326)
Starts video capture using the camera specified by the `UIImagePickerControllerCameraDevice` (page 330) property.

- `stopVideoCapture` (page 327)
Stops video capture.

Configuring the Camera

`cameraDevice` (page 319) *property*
The camera used by the image picker controller.

+ `isCameraDeviceAvailable:` (page 325)
Returns a Boolean value that indicates whether a given camera is available.

+ `availableCaptureModesForCameraDevice:` (page 324)
Returns an array of `NSNumber` objects indicating the capture modes supported by a given camera device.

`cameraCaptureMode` (page 319) *property*
The capture mode used by the camera.

`cameraFlashMode` (page 319) *property*
The flash mode used by the active camera.

`+ isFlashAvailableForCameraDevice:` (page 325)
Indicates whether a given camera has flash illumination capability.

Properties

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

allowsEditing

A Boolean value indicating whether the user is allowed to edit a selected still image or movie.

```
@property(nonatomic) BOOL allowsEditing
```

Discussion
If you allow the user to edit still images or movies, the delegate may receive a dictionary with information about the edits that were made. The protocol for the delegate is described in *UIImagePickerControllerDelegate Protocol Reference*.

This property is set to `NO` by default.

Availability
Available in iOS 3.1 and later.

Declared In
`UIImagePickerController.h`

allowsImageEditing

A Boolean value indicating whether the user is allowed to edit a selected image. (Deprecated in iOS 3.1. Use `allowsEditing` (page 318) instead.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 3.1.

Declared In
`UIImagePickerController.h`

cameraCaptureMode

The capture mode used by the camera.

```
@property(nonatomic) UIImagePickerControllerCameraCaptureMode cameraCaptureMode
```

Discussion
The various capture modes are listed in the "UIImagePickerControllerCameraCaptureMode" (page 330) enumeration. The default value is `UIImagePickerControllerCameraCaptureModePhoto` (page 331).

Availability
Available in iOS 4.0 and later.

See Also
`@property cameraDevice` (page 319)
`+ availableCaptureModesForCameraDevice:` (page 324)

Declared In
`UIImagePickerController.h`

cameraDevice

The camera used by the image picker controller.

```
@property(nonatomic) UIImagePickerControllerCameraDevice cameraDevice
```

Discussion
The default is `UIImagePickerControllerCameraDeviceRear` (page 330).

Availability
Available in iOS 4.0 and later.

See Also
`+ isCameraDeviceAvailable:` (page 325)
`+ isFlashAvailableForCameraDevice:` (page 325)
`+ availableCaptureModesForCameraDevice:` (page 324)

Declared In
`UIImagePickerController.h`

cameraFlashMode

The flash mode used by the active camera.

```
@property(nonatomic) UIImagePickerControllerCameraFlashMode cameraFlashMode
```

Discussion
The various flash modes are listed in the "UIImagePickerControllerCameraFlashMode" (page 331) enumeration. The default value is `UIImagePickerControllerCameraFlashModeAuto` (page 331).

The value of this property specifies the behavior of the still-image flash when the value of the `cameraCaptureMode` property is `UIImagePickerControllerCameraCaptureModePhoto` (page 331), and specifies the behavior of the video torch when `cameraCaptureMode` is `UIImagePickerControllerCameraCaptureModeVideo` (page 331).

Availability
Available in iOS 4.0 and later.

See Also
`@property cameraDevice` (page 319)
`@property cameraCaptureMode` (page 319)
`+ isFlashAvailableForCameraDevice:` (page 325)

Declared In
`UIImagePickerController.h`

cameraOverlayView

The custom view to display on top of the default image picker interface.

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

Discussion
You can use an overlay view to present a custom view hierarchy on top of the default image picker interface. The image picker layers your custom overlay view on top of the other image picker views and positions it relative to the screen coordinates. If you have the default camera controls set to be visible, incorporate transparency into your view, or position it to avoid obscuring the underlying content.

This property is set to `nil` by default.

You can access this property only when the source type of the image picker is set to `UIImagePickerControllerSourceTypeCamera`. Attempting to access this property for other source types results in the throwing of an `NSInvalidArgumentException` exception.

Availability
Available in iOS 3.1 and later.

See Also
`@property showsCameraControls` (page 322)

Declared In
`UIImagePickerController.h`

cameraViewTransform

The transform to apply to the camera's preview image.

```
@property(nonatomic) CGAffineTransform cameraViewTransform
```

Discussion
This transform affects the live preview image only and does not affect your custom overlay view or the default image picker controls. You can use this property in conjunction with custom controls to implement your own electronic zoom behaviors.

You can access this property only when the source type of the image picker is set to `UIImagePickerControllerSourceTypeCamera`. Attempting to access this property for other source types results in the throwing of an `NSInvalidArgumentException` exception.

Availability
Available in iOS 3.1 and later.

Declared In
`UIImagePickerController.h`

delegate

The image picker's delegate object.

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

Discussion
The delegate receives notifications when the user picks an image or movie, or exits the picker interface. The delegate also decides when to dismiss the picker interface, so you must provide a delegate to use a picker. If this property is `nil`, the picker is dismissed immediately if you try to show it.

For information about the methods you can implement for your delegate object, see *UIImagePickerControllerDelegate Protocol Reference*.

Availability
Available in iOS 2.0 and later.

Declared In
`UIImagePickerController.h`

mediaTypes

An array indicating the media types to be accessed by the picker interface.

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

Discussion
Depending on the media types you assign to this property, the picker displays the still camera or the movie camera interface, or a selection control that lets the user choose the picker interface. Before setting this property, check which media types are available by calling the `availableMediaTypesForSourceType:` (page 324) class method.

By default, this property is set to the single value `kUTTypeImage`, which designates the still camera interface.

If you set this property to an empty array, or to an array in which none of the media types is available for the current source, the system throws an exception.

Availability
Available in iOS 3.0 and later.

Declared In
`UIImagePickerController.h`

showsCameraControls

Indicates whether the image picker displays the default camera controls.

```
@property(nonatomic) BOOL showsCameraControls
```

Discussion
The default value of this property is `YES`, which specifies that the default camera controls are visible in the picker. Set it to `NO` to hide the default controls if you want to instead provide a custom overlay view using the `cameraOverlayView` property.

Note: In iOS 3.1.3 and earlier, hiding the default camera controls limits you to taking still pictures only, regardless of whether movie capture is available on the device.

If you set this property to `NO` and provide your own custom controls, you can take multiple pictures before dismissing the image picker interface. However, if you set this property to `YES`, your delegate must dismiss the image picker interface after the user takes one picture or cancels the operation.

You can access this property only when the source type of the image picker is set to `UIImagePickerControllerSourceTypeCamera`. Attempting to access this property for other source types results in the throwing of an `NSInvalidArgumentException` exception. Depending on the value you assign to the `mediaTypes` (page 321) property, the default controls display the still camera or movie camera interface, or a selection control that lets the user choose the picker interface.

Availability
Available in iOS 3.1 and later.

See Also
`@property cameraOverlayView` (page 320)
`- takePicture` (page 327)

Declared In
`UIImagePickerController.h`

sourceType

The type of picker interface displayed by the controller.

```
@property(nonatomic) UIImagePickerControllerSourceType sourceType
```

Discussion
Prior to running the picker interface, set this value to the desired source type. The specified source type must be available and an exception is thrown if it is not. If you change this property while the picker is visible, the picker interface changes to match the new value in this property.

The various source types are listed in the "UIImagePickerControllerSourceType" (page 328) enumeration. The default value is `UIImagePickerControllerSourceTypePhotoLibrary`.

Availability
Available in iOS 2.0 and later.

See Also
`+ isSourceTypeAvailable:` (page 326)

Declared In
UIImagePickerController.h

videoMaximumDuration

The maximum duration, in seconds, for a video recording.

```
@property(nonatomic) NSTimeInterval videoMaximumDuration
```

Discussion
The default value for this property is 10 minutes (600 seconds). When a user taps the Share button to send a movie to MMS, MobileMe, YouTube, or another destination, an appropriate duration limit and an appropriate video quality are enforced.

This property is available only if the `mediaTypes` (page 321) property's value array includes the `kUTTypeMovie` media type.

Availability
Available in iOS 3.1 and later.

See Also
`+ availableMediaTypesForSourceType:` (page 324)
`+ isSourceTypeAvailable:` (page 326)

Declared In
UIImagePickerController.h

videoQuality

The video recording and transcoding quality.

```
@property(nonatomic) UIImagePickerControllerQualityType videoQuality
```

Discussion
The video quality setting specified by this property is used during video recording. It is also used whenever picking a recorded movie. Specifically, if the video quality setting is lower than the video quality of an existing movie, displaying that movie in the picker results in transcoding the movie to the lower quality.

The various video qualities are listed in the "`UIImagePickerControllerQualityType`" (page 329) enumeration. The default value is `UIImagePickerControllerQualityTypeMedium` (page 329). To capture or transcode a movie using a video quality other than the default value, you must set the quality explicitly.

This property is available only if the `mediaTypes` (page 321) property's value array includes the `kUTTypeMovie` media type.

Availability
Available in iOS 3.1 and later.

See Also
`+ availableMediaTypesForSourceType:` (page 324)
`+ isSourceTypeAvailable:` (page 326)

Declared In
UIImagePickerController.h

Class Methods

availableCaptureModesForCameraDevice:

Returns an array of NSNumber objects indicating the capture modes supported by a given camera device.

```
+ (NSArray
    *)availableCaptureModesForCameraDevice:(UIImagePickerControllerCameraDevice)cameraDevice
```

Parameters

cameraDevice

A "UIImagePickerControllerCameraDevice" (page 330) constant indicating the camera you want to interrogate.

Return Value
An array of NSNumber objects indicating the capture modes supported by *cameraDevice*.

Discussion
See "UIImagePickerControllerCameraCaptureMode" (page 330) for possible values.

Availability
Available in iOS 4.0 and later.

See Also
@property cameraCaptureMode (page 319)
+ availableCaptureModesForCameraDevice: (page 324)

Declared In
UIImagePickerController.h

availableMediaTypesForSourceType:

Returns an array of the available media types for the specified source type.

```
+ (NSArray
    *)availableMediaTypesForSourceType:(UIImagePickerControllerSourceType)sourceType
```

Parameters

sourceType

The source to use to pick an image.

Return Value
An array whose elements identify the available media types for the specified source type.

Discussion
Some iOS devices support video recording. Use this method, along with the isSourceTypeAvailable: (page 326) method, to determine if video recording is available on a device. The availability of video recording is indicated by the presence of the kUTTypeMovie media type for the UIImagePickerControllerSourceTypeCamera (page 328) source type.

Availability
Available in iOS 3.0 and later.

See Also
`+ isSourceTypeAvailable:` (page 326)

Declared In
`UIImagePickerController.h`

isCameraDeviceAvailable:

Returns a Boolean value that indicates whether a given camera is available.

```
+ (BOOL)isCameraDeviceAvailable:(UIImagePickerControllerCameraDevice)cameraDevice
```

Parameters

cameraDevice

A "UIImagePickerControllerCameraDevice" (page 330) constant indicating the camera whose availability you want to check.

Return Value
`YES` if the camera indicated by *cameraDevice* is available, or `NO` if it is not available.

Availability
Available in iOS 4.0 and later.

See Also
`@property cameraDevice` (page 319)
`+ isFlashAvailableForCameraDevice:` (page 325)
`+ availableCaptureModesForCameraDevice:` (page 324)

Declared In
`UIImagePickerController.h`

isFlashAvailableForCameraDevice:

Indicates whether a given camera has flash illumination capability.

```
+
    (BOOL)isFlashAvailableForCameraDevice:(UIImagePickerControllerCameraDevice)cameraDevice
```

Parameters

cameraDevice

A "UIImagePickerControllerCameraDevice" (page 330) constant indicating the camera whose flash capability you want to know.

Return Value
`YES` if *cameraDevice* can use flash illumination, or `NO` if it cannot.

Availability
Available in iOS 4.0 and later.

See Also
`@property cameraDevice` (page 319)
`@property cameraFlashMode` (page 319)

Declared In
`UIImagePickerController.h`

isSourceTypeAvailable:

Returns a Boolean value indicating whether the device supports picking media using the specified source type.

```
+ (BOOL)isSourceTypeAvailable:(UIImagePickerControllerSourceType)sourceType
```

Parameters
sourceType
The source to use to pick an image or movie.

Return Value
`YES` if the device supports the specified source type; `NO` if the specified source type is not available.

Discussion
Because a media 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 iOS 2.0 and later.

See Also
`+ availableMediaTypesForSourceType:` (page 324)

Declared In
`UIImagePickerController.h`

Instance Methods

startVideoCapture

Starts video capture using the camera specified by the `UIImagePickerControllerCameraDevice` (page 330) property.

```
- (BOOL)startVideoCapture
```

Return Value
`YES` on success or `NO` on failure. This method may return a value of `NO` for various reasons, among them the following:

- Movie capture is already in progress
- The device does not support movie capture
- The device is out of disk space

Discussion
Use this method in conjunction with a custom overlay view to initiate the programmatic capture of a movie. You can take more than one movie without leaving the interface, but to do so requires you to hide the default image picker controls.

Calling this method while a movie is being captured has no effect. You must can the `stopVideoCapture` (page 327) method, and then wait until the associated delegate object receives an `imagePickerController:didFinishPickingMediaWithInfo:` (page 864) message, before you can capture another movie.

Calling this method when the source type of the image picker is set to a value other than `UIImagePickerControllerSourceTypeCamera` (page 328) results in the throwing of an `NSInvalidArgumentException` exception.

If you require additional options or more control over movie capture, use the movie capture methods in the AV Foundation framework. Refer to *AV Foundation Framework Reference*.

Availability
Available in iOS 4.0 and later.

Declared In
`UIImagePickerController.h`

stopVideoCapture

Stops video capture.

```
- (void)stopVideoCapture
```

Discussion
After you call this method to stop video capture, the system calls the image picker delegate's `imagePickerController:didFinishPickingMediaWithInfo:` (page 864) method.

Availability
Available in iOS 4.0 and later.

Declared In
`UIImagePickerController.h`

takePicture

Captures a still image using the camera.

```
- (void)takePicture
```

Discussion
Use this method in conjunction with a custom overlay view to initiate the programmatic capture of a still image. This supports taking more than one picture without leaving the interface, but requires that you hide the default image picker controls.

Calling this method while an image is being captured has no effect. You must wait until the associated delegate object receives an `imagePickerController:didFinishPickingMediaWithInfo:` (page 864) message before you can capture another picture.

Calling this method when the source type of the image picker is set to a value other than `UIImagePickerControllerSourceTypeCamera` results in the throwing of an `NSInvalidArgumentException` exception.

Availability
Available in iOS 3.1 and later.

See Also
`@property cameraOverlayView` (page 320)

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 or movie, as available, from the device's photo library.
Available in iOS 2.0 and later.
Declared in `UIImagePickerController.h`.

`UIImagePickerControllerSourceTypeCamera`
Take a new picture or movie, as available, using the device's specified, built-in camera. Specify the camera you want by using the `cameraDevice` (page 319) property.
Available in iOS 2.0 and later.
Declared in `UIImagePickerController.h`.

`UIImagePickerControllerSourceTypeSavedPhotosAlbum`
Pick an image or movie, as available, from the device's camera roll. If the device does not have a camera, pick an image or movie, as available, from the Saved Photos folder on the device.

Available in iOS 2.0 and later.

Declared in `UIImagePickerController.h`.

Discussion
A given source may not be available on a given device because the source is not physically present or because it cannot currently be accessed.

UIImagePickerControllerQualityType

Video quality settings for movies recorded with the built-in camera, or transcoded by displaying in the image picker.

```
enum {
    UIImagePickerControllerQualityTypeHigh    = 0,
    UIImagePickerControllerQualityType640x480 = 3,
    UIImagePickerControllerQualityTypeMedium  = 1,  // default value
    UIImagePickerControllerQualityTypeLow     = 2
};
typedef NSUInteger UIImagePickerControllerQualityType;
```

Constants
`UIImagePickerControllerQualityTypeHigh`
If recording, specifies that you want to use the highest-quality video recording supported for the active camera on the device.

Recorded files are suitable for on-device playback and for wired transfer to the Desktop using Image Capture; they are likely to be too large for transfer using Wi-Fi.

If displaying a recorded movie in the image picker, specifies that you do not want to reduce the video quality of the movie.

Available in iOS 3.1 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerQualityType640x480`
If recording, specifies that you want to use VGA-quality video recording (pixel dimensions of 640x480).

If displaying a recorded movie in the image picker, specifies that you want to transcode higher-quality movies to VGA video quality.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerQualityTypeMedium`
If recording, specifies that you want to use medium-quality video recording.

Recorded files can usually be transferred using Wi-Fi. This is the default video quality setting.

If displaying a recorded movie in the image picker, specifies that you want to transcode higher-quality movies to medium video quality.

Available in iOS 3.1 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerQualityTypeLow`
If recording, specifies that you want to use low-quality video recording.

Recorded files can usually be transferred over the cellular network.

If displaying a recorded movie in the image picker, specifies that you want to transcode higher-quality movies to low video quality.

Available in iOS 3.1 and later.

Declared in `UIImagePickerController.h`.

Discussion
The constants in this enumeration are for use as values of the `videoQuality` (page 323) property.

The video quality setting applies to transcoding as well as to recording. Specifically, if the video quality setting is lower than the video quality of an existing movie, displaying that movie in the picker results in transcoding the movie to the lower quality.

UIImagePickerControllerCameraDevice

The camera to use for image or movie capture.

```
enum {
    UIImagePickerControllerCameraDeviceRear,
    UIImagePickerControllerCameraDeviceFront
};
typedef NSUInteger UIImagePickerControllerCameraDevice;
```

Constants
`UIImagePickerControllerCameraDeviceRear`
Specifies the camera on the rear of the device.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerCameraDeviceFront`
Specifies the camera on the front of the device.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

Discussion
The constants in this enumeration are for use as values of the `cameraDevice` (page 319) property.

UIImagePickerControllerCameraCaptureMode

The category of media for the camera to capture.

```
enum {
    UIImagePickerControllerCameraCaptureModePhoto,
    UIImagePickerControllerCameraCaptureModeVideo
};
typedef NSUInteger UIImagePickerControllerCameraCaptureMode;
```

Constants

`UIImagePickerControllerCameraCaptureModePhoto`

Specifies that the camera captures still images.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerCameraCaptureModeVideo`

Specifies that the camera captures movies.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

Discussion

The constants in this enumeration are for use as values of the `cameraCaptureMode` (page 319) property.

UIImagePickerControllerCameraFlashMode

The flash mode to use with the active camera.

```
enum {
    UIImagePickerControllerCameraFlashModeOff  = -1,
    UIImagePickerControllerCameraFlashModeAuto = 0,
    UIImagePickerControllerCameraFlashModeOn   = 1
};
typedef NSInteger UIImagePickerControllerCameraFlashMode;
```

Constants

`UIImagePickerControllerCameraFlashModeOff`

Specifies that flash illumination is always off, no matter what the ambient light conditions are.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerCameraFlashModeAuto`

Specifies that the device should consider ambient light conditions to automatically determine whether or not to use flash illumination.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerCameraFlashModeOn`

Specifies that flash illumination is always on, no matter what the ambient light conditions are.

Available in iOS 4.0 and later.

Declared in `UIImagePickerController.h`.

Discussion

The constants in this enumeration are for use as values of the `cameraFlashMode` (page 319) property.

The behavior of the flash depends on the camera capture mode.

- For a `cameraCaptureMode` (page 319) value of `UIImagePickerControllerCameraCaptureModePhoto` (page 331), flash is used to transiently illuminate the subject during still image capture.
- For a `cameraCaptureMode` (page 319) value of `UIImagePickerControllerCameraCaptureModeVideo` (page 331), flash is used to continuously illuminate the subject during movie capture.

For a given camera on a device, flash may or may not be available. You specify the active camera by way of the `cameraDevice` (page 319) property. You can determine if the active camera has flash available by calling the `isFlashAvailableForCameraDevice:` (page 325) class method.

You can manipulate the flash directly to provide effects such as a strobe light. Present a picker interface set to use video capture mode. Then, turn the flash LED on or off by setting the `cameraFlashMode` property to `UIImagePickerControllerCameraFlashModeOn` or `UIImagePickerControllerCameraFlashModeOff`.

UIImageView Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIImageView.h
Related sample code	aurioTouch GKRocket GKTank ScrollViewSuite SimpleGestureRecognizers

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.

When a `UIImageView` object displays one of its images, the actual behavior is based on the properties of the image and the view. If either of the image's `leftCapWidth` or `topCapHeight` properties are non-zero, then the image is stretched according to the values in those properties. Otherwise, the image is scaled, sized to fit, or positioned in the image view 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.

All images associated with a `UIImageView` object should use the same `scale`. If your application uses images with different scales, they may render incorrectly.

Subclassing Notes

Special Considerations

The `UIImageView` class is optimized to draw its images to the display. `UIImageView` will not call `drawRect:` (page 726) a subclass. If your subclass needs custom drawing code, it is recommended you use `UIView` as the base class.

Tasks

Initializing a UIImageView Object

- `initWithImage:` (page 338)
  Returns an image view initialized with the specified image.
- `initWithImage:highlightedImage:` (page 339)
  Returns an image view initialized with the specified regular and highlighted images.

Image Data

`image` (page 337) *property*
The image displayed in the image view.

`highlightedImage` (page 337) *property*
The highlighted image displayed in the image view.

Animating Images

`animationImages` (page 335) *property*
An array of `UIImage` objects to use for an animation.

`highlightedAnimationImages` (page 336) *property*
An array of `UIImage` objects to use for an animation when the view is highlighted.

`animationDuration` (page 335) *property*
The amount of time it takes to go through one cycle of the images.

`animationRepeatCount` (page 336) *property*
Specifies the number of times to repeat the animation.

- `startAnimating` (page 339)
  Starts animating the images in the receiver.
- `stopAnimating` (page 340)
  Stops animating the images in the receiver.
- `isAnimating` (page 339)
  Returns a Boolean value indicating whether the animation is running.

Setting and Getting Attributes

`userInteractionEnabled` (page 338) *property*
A Boolean value that determines whether user events are ignored and removed from the event queue.

`highlighted` (page 336) *property*
A Boolean value that determines whether the image is highlighted.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

Related Sample Code
ScrollViewSuite

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.

Availability
Available in iOS 2.0 and later.

See Also
`@property image` (page 337)
`@property contentMode` (page 700) (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 iOS 2.0 and later.

Declared In
UIImageView.h

highlighted

A Boolean value that determines whether the image is highlighted.

```
@property(nonatomic, getter=isHighlighted) BOOL highlighted
```

Discussion
This property determines whether the regular or highlighted images are used. When `highlighted` is set to `YES`, a non-animated image will use the `highlightedImage` property and an animated image will use the `highlightedAnimationImages`. If both of those properties are set to `nil` or if `highlighted` is set to `NO`, it will use the `image` and `animationImages` properties.

Availability
Available in iOS 3.0 and later.

Declared In
UIImageView.h

highlightedAnimationImages

An array of `UIImage` objects to use for an animation when the view is highlighted.

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

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 `highlightedImage` property. The value of this property is `nil` by default.

Availability
Available in iOS 3.0 and later.

See Also
`@property highlightedImage` (page 337)

`@property contentMode` (page 700) (UIView)

Declared In
`UIImageView.h`

highlightedImage

The highlighted image displayed in the image view.

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

Discussion
The initial value of this property is the image passed into the `initWithImage:highlightedImage:` (page 339) method or `nil` if you initialized the receiver using a different method.

If the `highlightedAnimationImages` property contains a value other than `nil`, the contents of this property are not used.

Availability
Available in iOS 3.0 and later.

See Also
`@property highlightedAnimationImages` (page 336)

Declared In
`UIImageView.h`

image

The image displayed in the image view.

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

Discussion
The initial value of this property is the image passed into the `initWithImage:` (page 338) method or `nil` if you initialized the receiver using a different method.

If the `animationImages` property contains a value other than `nil`, the contents of this property are not used.

Setting the image property does not change the size of a `UIImageView`. Call `sizeToFit` (page 736) to adjust the size of the view to match the image.

Availability
Available in iOS 2.0 and later.

See Also
`@property animationImages` (page 335)

Related Sample Code
SimpleGestureRecognizers

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 iOS 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 adjusts the frame of the receiver to match the size of the specified image. It also disables user interactions for the image view by default.

Availability
Available in iOS 2.0 and later.

Related Sample Code
aurioTouch
GKRocket
GKTank
ScrollViewSuite
WiTap

Declared In
UIImageView.h

initWithImage:highlightedImage:

Returns an image view initialized with the specified regular and highlighted images.

```
- (id)initWithImage:(UIImage *)image highlightedImage:(UIImage *)highlightedImage
```

Parameters

image

The initial image to display in the image view.

highlightedImage

The image to display if the image view is highlighted.

Return Value
An initialized image view object.

Discussion
This method adjusts the frame of the receiver to match the size of the specified image. It also disables user interactions for the image view by default.

Availability
Available in iOS 3.0 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 iOS 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 iOS 2.0 and later.

Declared In
`UIImageView.h`

stopAnimating

Stops animating the images in the receiver.

```
- (void)stopAnimating
```

Availability
Available in iOS 2.0 and later.

Declared In
`UIImageView.h`

UILabel Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UILabel.h
Related sample code	CryptoExercise GKRocket GKTank ScrollViewSuite WiTap

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 740). 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 700) 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 347) *property*
The text displayed by the label.

`font` (page 344) *property*
The font of the text.

`textColor` (page 348) *property*
The color of the text.

`textAlignment` (page 348) *property*
The technique to use for aligning the text.

`lineBreakMode` (page 345) *property*
The technique to use for wrapping and truncating the label's text.

`enabled` (page 344) *property*
The enabled state to use when drawing the label's text.

Sizing the Label's Text

`adjustsFontSizeToFitWidth` (page 343) *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 343) *property*
Controls how text baselines are adjusted when text needs to shrink to fit in the label.

`minimumFontSize` (page 346) *property*
The size of the smallest permissible font with which to draw the label's text.

`numberOfLines` (page 346) *property*
The maximum number of lines to use for rendering text.

Managing Highlight Values

`highlightedTextColor` (page 345) *property*
The highlight color applied to the label's text.

`highlighted` (page 344) *property*
A Boolean value indicating whether the receiver should be drawn with a highlight.

Drawing a Shadow

`shadowColor` (page 347) *property*
The shadow color of the text.

`shadowOffset` (page 347) *property*
The shadow offset (measured in points) for the text.

Drawing and Positioning Overrides

`- textRectForBounds:limitedToNumberOfLines:` (page 350)
Returns the drawing rectangle for the label's text.

`- drawTextInRect:` (page 349)
Draws the receiver's text (or its shadow) in the specified rectangle.

Setting and Getting Attributes

`userInteractionEnabled` (page 349) *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 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 iOS 2.0 and later.

See Also
`@property font` (page 344)
`@property minimumFontSize` (page 346)

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 `adjustsFontSizeToFitWidth` 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 58).

Availability
Available in iOS 2.0 and later.

See Also
`@property adjustsFontSizeToFitWidth` (page 343)

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 iOS 2.0 and later.

See Also
`@property adjustsFontSizeToFitWidth` (page 343)

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 the system font at a size of 17 points (using the `systemFontOfSize:` class method of `UIFont`). The value for the property can only be set to a non-`nil` value; setting this property to `nil` raises an exception.

Availability
Available in iOS 2.0 and later.

Related Sample Code
aurioTouch

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 iOS 2.0 and later.

See Also
`@property highlightedTextColor` (page 345)

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 iOS 2.0 and later.

See Also
`@property highlighted` (page 344)

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 57) by default.

Availability
Available in iOS 2.0 and later.

See Also
`@property adjustsFontSizeToFitWidth` (page 343)

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 iOS 2.0 and later.

See Also
`@property adjustsFontSizeToFitWidth` (page 343)

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 736) method, resizing takes into account the value stored in this property. For example, if this property is set to 3, the `sizeToFit` (page 736) method resizes the receiver so that it is big enough to display three lines of text.

Availability
Available in iOS 2.0 and later.

See Also
`@property enabled` (page 344)
`@property adjustsFontSizeToFitWidth` (page 343)
`- sizeToFit` (page 736) (UIView)

Related Sample Code
WiTap

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 iOS 2.0 and later.

See Also
`@property shadowOffset` (page 347)

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 iOS 2.0 and later.

See Also
`@property shadowColor` (page 347)

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 iOS 2.0 and later.

Related Sample Code
aurioTouch
BonjourWeb
GKRocket
ToolbarSearch
WiTap

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 iOS 2.0 and later.

Related Sample Code
aurioTouch

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 a black color (set through the `blackColor` (page 196) class method of `UIColor`). The value for the property can only be set to a non-`nil` value; setting this property to `nil` raises an exception.

Availability
Available in iOS 2.0 and later.

Related Sample Code
aurioTouch
BonjourWeb

WiTap

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 iOS 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 iOS 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. For this method to be called, there must be a prior call to the sizeToFit or sizeThatFits: method. Note that labels in UITableViewCell objects are sized based on the cell dimensions, and not a requested size.

The default implementation of this method returns the original *bounds* rectangle.

Availability

Available in iOS 2.0 and later.

Declared In

UILabel.h

UILocalizedIndexedCollation Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	UILocalizedIndexedCollation.h

Overview

The `UILocalizedIndexedCollation` class is a convenience for organizing, sorting, and localizing the data for a table view that has a section index. The table view's data source then uses the collation object to provide the table view with input for section titles and section index titles.

Table views with section indexes are ideal for displaying and facilitating the access of data composed of many items organized by a sequential ordering scheme such as the alphabet. Users tap an index title to jump to the corresponding section. The initial table view of the Phone/Contacts application on the iPhone is an example. Note that the section titles can be different than the titles of the index.

To prepare the data for a section index, the `UITableViewController` object creates a indexed-collation object and then, for each model object that is to be indexed, calls `sectionForObject:collationStringSelector:` (page 354). This method determines the section in which each of these objects should appear and returns an integer that identifies the section. The table-view controller then puts each object in a local array for its section. For each section array, the controller calls the `sortedArrayFromArray:collationStringSelector:` (page 355) method to sort all of the objects in the section. The indexed-collation object is now the data store that the table-view controller uses to provide section-index data to the table view, as illustrated in Listing 30-1.

Listing 30-1 Data source using indexed-collation object to provide data to table view

```
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
{
    return [[[UILocalizedIndexedCollation currentCollation] sectionTitles]
objectAtIndex:section];
}

- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView
{
    return [[UILocalizedIndexedCollation currentCollation] sectionIndexTitles];
}
```

```
- (NSInteger)tableView:(UITableView *)tableView
sectionForSectionIndexTitle:(NSString *)title atIndex:(NSInteger)index
{
    return [[UILocalizedIndexedCollation currentCollation]
sectionForSectionIndexTitleAtIndex:index];
}
```

Tasks

Getting the Shared Instance

`+ currentCollation` (page 353)
Returns the shared indexed-collation instance.

Preparing the for Sections and Section Indexes

`- sectionForObject:collationStringSelector:` (page 354)
Returns an integer identifying the section in which a model object belongs.

`- sortedArrayFromArray:collationStringSelector:` (page 355)
Sorts the objects within a section by their localized titles.

Providing Section Index Data to the Table View

`sectionTitles` (page 353) *property*
Returns the list of section titles for the table view. (read-only)

`sectionIndexTitles` (page 352) *property*
Returns the list of section-index titles for the table view (read-only)

`- sectionForSectionIndexTitleAtIndex:` (page 354)
Returns the section that the table view should scroll to for the given index title.

Properties

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

sectionIndexTitles

Returns the list of section-index titles for the table view (read-only)

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

Discussion
This property contains the localized list of section-index titles sorted according to the specified ordering (for example, A through Z in US English). In its implementation of `sectionIndexTitlesForTableView:` (page 927), the data source can call this method on the indexed-collation object and pass back the result.

Availability
Available in iOS 3.0 and later.

See Also
`@property sectionTitles` (page 353)
`- sectionForSectionIndexTitleAtIndex:` (page 354)

Declared In
`UILocalizedIndexedCollation.h`

sectionTitles

Returns the list of section titles for the table view. (read-only)

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

Discussion
This property contains the localized list of section titles sorted according to the specified ordering (for example, A through Z in US English). In its implementation of `tableView:titleForHeaderInSection:` (page 932), the data source can call this method on the indexed-collation object, passing in the section index and returning the result.

Availability
Available in iOS 3.0 and later.

See Also
`@property sectionIndexTitles` (page 352)
`- sectionForSectionIndexTitleAtIndex:` (page 354)

Declared In
`UILocalizedIndexedCollation.h`

Class Methods

currentCollation

Returns the shared indexed-collation instance.

```
+ (id)currentCollation
```

Return Value
A `UILocalizedIndexedCollation` object or `nil` if there was a problem creating the object.

Availability
Available in iOS 3.0 and later.

Declared In
`UILocalizedIndexedCollation.h`

Instance Methods

sectionForObject:collationStringSelector:

Returns an integer identifying the section in which a model object belongs.

```
- (NSInteger)sectionForObject:(id)object collationStringSelector:(SEL)selector
```

Parameters

object
A model object of the application that is part of the data model for the table view.

selector
A selector that identifies a method returning an identifying string for `object` that is used in collation. The method should take no arguments and return an `NSString` object. For example, this could be a `name` property on the object.

Return Value
An integer that identifies the section in which the model object belongs. The numbers returned indicate a sequential ordering.

Discussion
The table-view controller should iterate through all model objects for the table view and call this method for each object. If the application provides a `Localizable.strings` file for the current language preference, the indexed-collation object localizes each string returned by the method identified by *selector*. It uses this localized name when collating titles. The controller should use the returned integer to identify a local "section" array in which it should insert *object*.

Availability
Available in iOS 3.0 and later.

See Also
`- sortedArrayFromArray:collationStringSelector:` (page 355)

Declared In
`UILocalizedIndexedCollation.h`

sectionForSectionIndexTitleAtIndex:

Returns the section that the table view should scroll to for the given index title.

```
- (NSInteger)sectionForSectionIndexTitleAtIndex:(NSInteger)indexTitleIndex
```

Parameters

indexTitleIndex

An integer identifying a section-index title by its position in the array of such titles.

Return Value

An integer identifying the table-view section associated with *indexTitleIndex*.

Discussion

This method allows the table view to map between a given item in the section index and a given section even when there isn't a one-to-one mapping. In its implementation of `tableView:sectionForSectionIndexTitle:atIndex:` (page 931), the data source can call this method on the indexed-collation object specifying as an argument the passed-in index integer; it then returns the result to the table view.

Availability

Available in iOS 3.0 and later.

See Also

`@property sectionTitles` (page 353)

`@property sectionIndexTitles` (page 352)

Declared In

`UILocalizedIndexedCollation.h`

sortedArrayFromArray:collationStringSelector:

Sorts the objects within a section by their localized titles.

```
- (NSArray *)sortedArrayFromArray:(NSArray *)array
    collationStringSelector:(SEL)selector
```

Parameters

array

An array containing the model objects for a section.

selector

A selector that identifies a method returning an identifying string for each object in *array*. The index-collation object uses this string for sorting the objects in the array. The method should take no arguments and return an `NSString` object. For example, this could be a `name` property on the object.

Return Value

A new array containing the items in *array*), sorted.

Discussion

The table-view controller creates the array of objects for a section (*array*) as part of iterating through its model objects with calls to the `sectionForObject:collationStringSelector:` (page 354) method. This method should be called on each local section array.

Availability

Available in iOS 3.0 and later.

Declared In

`UILocalizedIndexedCollation.h`

UILongPressGestureRecognizer Class Reference

Inherits from	UIGestureRecognizer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UILongPressGestureRecognizer.h
Companion guide	Event Handling Guide for iOS

Overview

`UILongPressGestureRecognizer` is a concrete subclass of `UIGestureRecognizer` that looks for long-press gestures. The user must press one or more fingers on a view for at least a specified period for the action message to be sent. In addition, the fingers may move only a specified distance for the gesture to be recognized; if they move beyond this limit the gesture fails.

Long-press gestures are continuous. The gesture begins (`UIGestureRecognizerStateBegan` (page 296)) when the number of allowable fingers (`numberOfTouchesRequired` (page 359)) have been pressed for the specified period (`minimumPressDuration` (page 358)) and the touches do not move beyond the allowable range of movement (`allowableMovement` (page 358)). The gesture recognizer transitions to the Change state whenever a finger moves, and it ends (`UIGestureRecognizerStateEnded` (page 296)) when any of the fingers are lifted.

Tasks

Configuring the Gesture Recognizer

`minimumPressDuration` (page 358) *property*
The minimum period fingers must press on the view for the gesture to be recognized.

`numberOfTouchesRequired` (page 359) *property*
The number of fingers that must be pressed on the view for the gesture to be recognized.

`numberOfTapsRequired` (page 358) *property*
The number of taps on the view required for the gesture to be recognized.

`allowableMovement` (page 358) *property*
The maximum movement of the fingers on the view before the gesture fails.

Properties

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

allowableMovement

The maximum movement of the fingers on the view before the gesture fails.

```
@property(nonatomic) CGFloat allowableMovement
```

Discussion
The allowable distance is in pixels. The default distance is 10 pixels.

Availability
Available in iOS 3.2 and later.

Declared In
`UILongPressGestureRecognizer.h`

minimumPressDuration

The minimum period fingers must press on the view for the gesture to be recognized.

```
@property(nonatomic) CFTimeInterval minimumPressDuration
```

Discussion
The time interval is in seconds. The default duration is is 0.4 seconds.

Availability
Available in iOS 3.2 and later.

Declared In
`UILongPressGestureRecognizer.h`

numberOfTapsRequired

The number of taps on the view required for the gesture to be recognized.

```
@property (nonatomic) NSUInteger  numberOfTapsRequired
```

Discussion
The default number of taps is 1.

Availability
Available in iOS 3.2 and later.

Declared In
`UILongPressGestureRecognizer.h`

numberOfTouchesRequired

The number of fingers that must be pressed on the view for the gesture to be recognized.

```
@property(nonatomic) NSInteger numberOfTouchesRequired
```

Discussion
The default number of fingers is 1.

Availability
Available in iOS 3.2 and later.

Declared In
`UILongPressGestureRecognizer.h`

UIMenuController Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	UIMenuController.h

Overview

The singleton `UIMenuController` instance presents the menu interface for the Cut, Copy, Paste, Select, Select All, and Delete commands.

This menu is referred to as the editing menu. When you make this menu visible, `UIMenuController` positions it relative to a target rectangle on the screen; this rectangle usually defines a selection. The menu appears above the target rectangle or, if there is not enough space for it, below it. The menu's pointer is placed at the center of the top or bottom of the target rectangle, as appropriate. Be sure to set the tracking rectangle before you make the menu visible. You are also responsible for detecting, tracking, and displaying selections.

The `UIResponderStandardEditActions` informal protocol declares methods that are invoked when the user taps a menu command. The `canPerformAction:withSender:` (page 461) method of `UIResponder` is also related to the editing menu. A responder implements this method to enable and disable commands of the editing menu just before the menu is displayed. You can force this updating of menu commands' enabled state by calling the `update` (page 365) method.

Note: iOS 3.2 introduced three changes to this class:

- You can add your own menu items to the editing menu via the `menuItems` (page 363) property.
- You can control the direction the arrow of the editing menu points through the `arrowDirection` (page 362) property.
- The Delete menu item was added to the set of system menu items. Tapping it invokes the `UIResponderStandardEditActions` action method `delete:`, also added in iOS 3.2.

Tasks

Getting the Menu Controller Instance

`+ sharedMenuController` (page 364)
Returns the menu controller.

Showing and Hiding the Menu

`menuVisible` (page 364) *property*
The visibility of the editing menu.

`- setMenuVisible:animated:` (page 364)
Shows or hides the editing menu, optionally animating the action.

Positioning the Menu

`- setTargetRect:inView:` (page 365)
Sets the area in a view above or below which the editing menu is positioned.

`menuFrame` (page 363) *property*
Returns the frame of the editing menu. (read-only)

`arrowDirection` (page 362) *property*
The direction the arrow of the editing menu is pointing.

Updating the Menu

`- update` (page 365)
Updates the enabled state of menu commands

Customizing Menu Items

`menuItems` (page 363) *property*
The custom menu items for the editing menu.

Properties

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

arrowDirection

The direction the arrow of the editing menu is pointing.

```
@property UIMenuControllerArrowDirection arrowDirection
```

Discussion
You can set the direction editing-menu arrow points by assigning a `UIMenuControllerArrowDirection` (page 366) enum constant to this property. The default behavior (`UIMenuControllerArrowDefault` (page 366)) is to point up or down at the object of focus based on its location on the screen.

Availability
Available in iOS 3.2 and later.

Declared In
`UIMenuController.h`

menuFrame

Returns the frame of the editing menu. (read-only)

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

Discussion
The property value is the bounding rectangle of the menu in screen coordinates. The property holds a value even if the menu is not visible. You can use this property to adjust any user-interface objects away from the menu.

Availability
Available in iOS 3.0 and later.

See Also
`- setTargetRect:inView:` (page 365)

Declared In
`UIMenuController.h`

menuItems

The custom menu items for the editing menu.

```
@property(copy) NSArray *menuItems
```

Discussion
The default value is `nil` (no custom menu items). Each menu item is an instance of the UIMenuItem class. You may create your own menu items, each with its own title and action selector, and add them to the editing menu through this property. Custom items appear in the menu after any system menu items.

Availability
Available in iOS 3.2 and later.

Declared In
`UIMenuController.h`

menuVisible

The visibility of the editing menu.

```
@property(nonatomic, getter=isMenuVisible) BOOL menuVisible
```

Discussion
Setting this property displays or hides the menu immediately, without animation. For animating the showing or hiding of the menu, use the `setMenuVisible:animated:` (page 364) method. Before showing the menu, be sure to position it relative to the selection.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

Class Methods

sharedMenuController

Returns the menu controller.

```
+ (UIMenuController *)sharedMenuController
```

Return Value
The shared `NSMenuController` instance.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

Instance Methods

setMenuVisible:animated:

Shows or hides the editing menu, optionally animating the action.

```
- (void)setMenuVisible:(BOOL)menuVisible animated:(BOOL)animated
```

Parameters

menuVisible
`YES` if the menu should be shown, `NO` if it should be hidden.

animated
`YES` if the showing or hiding of the menu should be animated, otherwise `NO`.

Discussion
Before showing the menu, be sure to position it relative to the selection. See `setTargetRect:inView:` (page 365) for details. If you do not set the target rect before displaying the menu, it appears at screen coordinates (0.0, 0.0).

Availability
Available in iOS 3.0 and later.

See Also
`@property menuVisible` (page 364)

Declared In
`UIMenuController.h`

setTargetRect:inView:

Sets the area in a view above or below which the editing menu is positioned.

```
- (void)setTargetRect:(CGRect)targetRect inView:(UIView *)targetView
```

Parameters

targetRect
A rectangle that defines the area that is to be the target of the menu commands.

targetView
The view in which *targetRect* appears.

Discussion
This target rectangle (*targetRect*) is usually the bounding rectangle of a selection. `UIMenuController` positions the editing menu above this rectangle; if there is not enough space for the menu there, it positions it below the rectangle. The menu's pointer is placed at the center of the top or bottom of the target rectangle as appropriate. Note that if you make the width or height of the target rectangle zero, `UIMenuController` treats the target area as a line or point for positioning (for example, an insertion caret or a single point).

Once it is set, the target rectangle does not track the view; if the view moves (such as would happen in a scroll view), you must update the target rectangle accordingly.

Availability
Available in iOS 3.0 and later.

See Also
`@property menuFrame` (page 363)

Declared In
`UIMenuController.h`

update

Updates the enabled state of menu commands

```
- (void)update
```

Discussion
By default, `UIMenuController` calls this method just before the editing menu is made visible and when touches occur in the menu. As a result, a responder object in the application enables or disables menu commands depending on the context; for example, if the pasteboard holds no data of a compatible type, the Paste command would be disabled. You can call this method to force an update of the editing menu. You may also override this method to add any custom behavior.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

Constants

UIMenuControllerArrowDirection

The direction the arrow of the editing menu is pointing.

```
typedef enum {
    UIMenuControllerArrowDefault,
    UIMenuControllerArrowUp,
    UIMenuControllerArrowDown,
    UIMenuControllerArrowLeft,
    UIMenuControllerArrowRight,
} UIMenuControllerArrowDirection;
```

Constants

`UIMenuControllerArrowDefault`
The arrow is pointing up or down at the object of focus based on its location in the screen.
Available in iOS 3.2 and later.
Declared in `UIMenuController.h`.

`UIMenuControllerArrowUp`
The arrow is pointing up at the object of focus.
Available in iOS 3.2 and later.
Declared in `UIMenuController.h`.

`UIMenuControllerArrowDown`
The arrow is pointing down at the object of focus.
Available in iOS 3.2 and later.
Declared in `UIMenuController.h`.

`UIMenuControllerArrowLeft`
The arrow is pointing left at the object of focus.
Available in iOS 3.2 and later.
Declared in `UIMenuController.h`.

`UIMenuControllerArrowRight`

The arrow is pointing right at the object of focus.

Available in iOS 3.2 and later.

Declared in `UIMenuController.h`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIMenuController.h`

Notifications

UIMenuControllerWillShowMenuNotification

Posted by the menu controller just before it shows the menu.

There is no `userInfo` dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

UIMenuControllerDidShowMenuNotification

Posted by the menu controller just after it shows the menu.

There is no `userInfo` dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

UIMenuControllerWillHideMenuNotification

Posted by the menu controller just before it hides the menu.

There is no `userInfo` dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

UIMenuControllerDidHideMenuNotification

Posted by the menu controller just after it hides the menu.

There is no `userInfo` dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

UIMenuControllerMenuFrameDidChangeNotification

Posted when the frame of a visible menu changes.

There is no `userInfo` dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`UIMenuController.h`

UINavigationBar Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UINavigationBar.h
Related sample code	AddMusic MultipleDetailViews

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.

You can use a navigation bar as a standalone object or in conjunction with a navigation controller object. To use a navigation bar as a standalone object, you create it and add it to your view hierarchy like you would any other view. Specifically, you can create it in Interface Builder and load it with the rest of your views or you can create it programmatically using the standard `alloc` and `initWithFrame:` (page 729) methods.

You can modify the appearance of the bar using the `barStyle` (page 372), `tintColor` (page 373), and `translucent` (page 374) properties. These properties affect the visual appearance of the bar itself but they also affect the way buttons are displayed in the bar. For example, if you set the `translucent` property to `YES`, any buttons in the bar are also made partially opaque.

For information about using a navigation bar with a navigation controller object, see "Using With a Navigation Controller" (page 370).

Adding Content to a Navigation Bar

When you use a navigation bar as a standalone object, you are responsible for providing its contents. Unlike other types of views, you do not add subviews to a navigation bar directly. Instead, you use a navigation item (an instance of the `UINavigationItem` class) to specify what buttons or custom views you want displayed. A navigation item has properties for specifying views on the left, right, and center of the navigation bar and for specifying a custom prompt string.

A navigation bar manages a stack of `UINavigationItem` objects. Although the stack is there mostly to support navigation controllers, you can use it as well to implement your own custom navigation interface. The topmost item in the stack represents the navigation item whose contents are currently displayed by the navigation bar. You push new navigation items onto the stack using the `pushNavigationItem:animated:` (page 375) method and pop items off the stack using the `popNavigationItemAnimated:` (page 374) method. Both of these changes can be animated for the benefit of the user.

In addition to pushing and popping items, you can also set the contents of the stack directly using either the `items` (page 373) property or the `setItems:animated:` (page 375) method. You might use these methods at launch time to restore your interface to its previous state or to push or pop more than one navigation item at a time.

If you are using a navigation bar as a standalone object, you should assign a custom delegate object to the `delegate` (page 372) property and use that object to intercept messages coming from the navigation bar. Delegate objects must conform to the `UINavigationBarDelegate` protocol. The delegate notifications let you know when the contents of responsible for deciding when items are pushed or popped from the stack—for example, it should display the previous view when the user clicks the back button.

For more information about creating navigation items, see *UINavigationItem Class Reference*. For more information about implementing a delegate object, see *UINavigationBarDelegate Protocol Reference*.

Using With a Navigation Controller

The most common way to use a navigation bar is in conjunction with a `UINavigationController` object. If you use a navigation controller to manage the navigation between different screens of content, the navigation controller creates the navigation bar automatically and pushes and pops navigation items when appropriate. You do not have to create the navigation bar and you do not have to manage the pushing and popping of navigation items yourself.

When used in conjunction with a navigation controller, there are only a handful of direct customizations you can make to the navigation bar. Specifically, it is alright to modify the `barStyle`, `tintColor`, and `translucent` properties of this class, but you must never directly change `UIView`-level properties such as the `frame` (page 701), `bounds` (page 698), `alpha` (page 696), or `hidden` (page 703) properties directly. In addition, you should let the navigation controller manage the stack of navigation items and not attempt to modify these items yourself.

A navigation controller automatically assigns itself as the delegate of its navigation bar object. Therefore, when using a navigation controller, you must not attempt to assign a custom delegate object to the corresponding navigation bar.

Tasks

Configuring Navigation Bars

`barStyle` (page 372) *property*
The appearance of the navigation bar.

`tintColor` (page 373) *property*
The color used to tint the bar.

`translucent` (page 374) *property*
A Boolean value indicating whether the navigation bar is only partially opaque.

Assigning the Delegate

`delegate` (page 372) *property*
The navigation bar's delegate object.

Pushing and Popping Items

- `pushNavigationItem:animated:` (page 375)
Pushes the given navigation item onto the receiver's stack and updates the navigation bar.

- `popNavigationItemAnimated:` (page 374)
Pops the top item from the receiver's stack and updates the navigation bar.

- `setItems:animated:` (page 375)
Replaces the navigation items currently managed by the navigation bar with the specified items.

`items` (page 373) *property*
An array of navigation items managed by the navigation bar.

`topItem` (page 373) *property*
The navigation item at the top of the navigation bar's stack. (read-only)

`backItem` (page 371) *property*
The navigation item that is immediately below the topmost item on navigation bar's stack. (read-only)

Properties

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

backItem

The navigation item that is immediately below the topmost item on navigation bar's stack. (read-only)

```
@property(nonatomic, readonly, retain) UINavigationItem *backItem
```

Discussion
If the `leftBarButtonItem` (page 395) property of the topmost navigation item is `nil`, the navigation bar displays a back button whose title is derived from the item in this property.

If there is only one item on the navigation bar's stack, the value of this property is `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`@property topItem` (page 373)
`@property items` (page 373)

Declared In
`UINavigationBar.h`

barStyle

The appearance of the navigation bar.

```
@property(nonatomic, assign) UIBarStyle barStyle
```

Discussion
See `UIBarStyle` (page 1011) for possible values. The default value is `UIBarStyleDefault` (page 1011).

It is permissible to set the value of this property when the navigation bar is being managed by a navigation controller object.

Availability
Available in iOS 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`.

If the navigation bar was created by a navigation controller and is being managed by that object, you must not change the value of this property. Navigation controllers act as the delegate for any navigation bars they create.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property backItem` (page 371)
`@property topItem` (page 373)

Declared In
`UINavigationBar.h`

tintColor

The color used to tint the bar.

```
@property(nonatomic, retain) UIColor *tintColor
```

Discussion
The default value is `nil`.

It is permissible to set the value of this property when the navigation bar is being managed by a navigation controller object.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property backItem` (page 371)
`@property items` (page 373)

Related Sample Code
AddMusic

Declared In
UINavigationBar.h

translucent

A Boolean value indicating whether the navigation bar is only partially opaque.

```
@property(nonatomic,assign,getter=isTranslucent) BOOL translucent
```

Discussion
Always set to YES if the barStyle property contains the value UIBarStyleBlackTranslucent (page 1012). When YES, the navigation bar is drawn with partial opacity, regardless of the bar style. The amount of opacity is fixed and cannot be changed.

It is permissible to set the value of this property when the navigation bar is being managed by a navigation controller object.

Availability
Available in iOS 3.0 and later.

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 iOS 2.0 and later.

See Also
- pushNavigationItem:animated: (page 375)

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 iOS 2.0 and later.

See Also

`- popNavigationItemAnimated:` (page 374)

Declared In

`UINavigationBar.h`

setItems:animated:

Replaces the navigation items currently managed by the navigation bar with the specified items.

```
- (void)setItems:(NSArray *)items  animated:(BOOL)animated
```

Parameters

items

The `UINavigationItem` objects to place in the stack. The front-to-back order of the items in this array represents the new bottom-to-top order of the items in the navigation stack. Thus, the last item added to the array becomes the top item of the navigation stack.

animated

If `YES`, animate the pushing or popping of the top stack item. If `NO`, replace the stack items without any animations.

Discussion

You can use this method to update or replace the navigation items in the stack without pushing or popping each item explicitly. In addition, this method lets you update the stack without animating the changes, which might be appropriate at launch time when you want to restore the state of the navigation stack to some previous state.

If animations are enabled, this method decides which type of transition to perform based on whether the last item in the *items* array is already on the current navigation stack. If the item is currently on the stack, but is not the topmost item, this method uses a pop transition; if it is the topmost item, no transition is performed. If the item is not on the stack, this method uses a push transition. Only one transition is performed, but when that transition finishes, the entire contents of the stack are replaced with the new items. For example, if items A, B, and C are on the stack and you set items D, A, and B, this method uses a pop transition and the resulting stack contains the items D, A, and B.

Availability
Available in iOS 3.0 and later.

Declared In
`UINavigationBar.h`

UINavigationController Class Reference

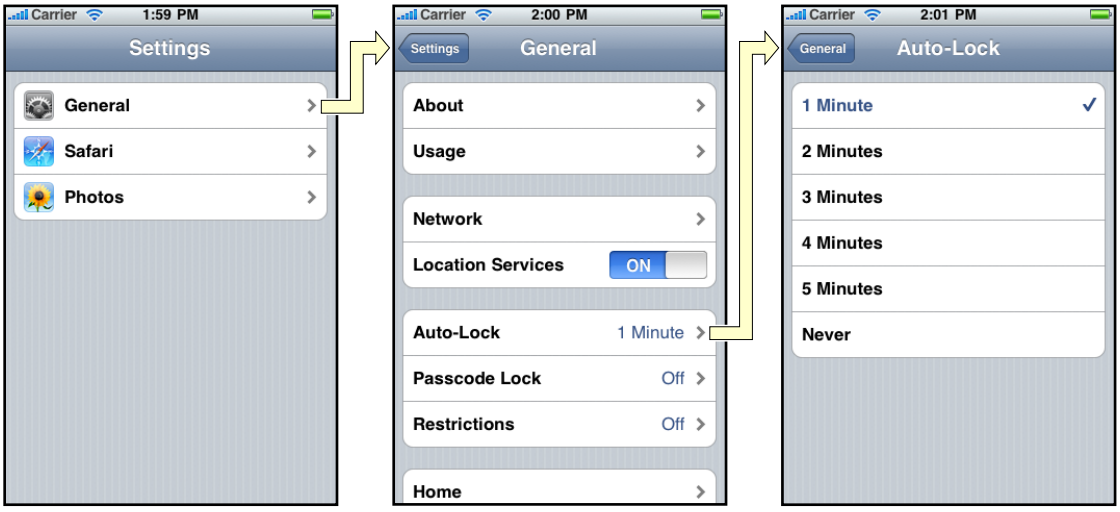
Inherits from	UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UINavigationController.h
Companion guide	View Controller Programming Guide for iOS
Related sample code	BonjourWeb CryptoExercise GKRocket ToolbarSearch

Overview

The `UINavigationController` class implements a specialized view controller that manages the navigation of hierarchical content. This class is not intended for subclassing. Instead, you use instances of it as-is in situations where you want your application's user interface to reflect the hierarchical nature of your content. This **navigation interface** makes it possible to present your data efficiently and also makes it easier for the user to navigate that content.

The screens presented by a navigation interface typically mimic the hierarchical organization of your data. At each level of the hierarchy, you provide an appropriate screen (managed by a custom view controller) to display the content at that level. Figure 34-1 shows an example of the navigation interface presented by the Settings application in iPhone Simulator. The first screen presents the user with the list of applications that contain preferences. Selecting an application reveals individual settings and groups of settings for that application. Selecting a group yields more settings and so on. For all but the root view, the navigation controller provides a back button to allow the user to move back up the hierarchy.

Figure 34-1 A sample navigation interface



A navigation controller object manages the currently displayed screens using the **navigation stack**. At the bottom of this stack is the root view controller and at the top of the stack is the view controller currently being displayed. You use the methods of your navigation controller object to modify the stack at runtime. The most common operation is to push new view controllers onto the stack using the `pushViewController:animated:` (page 388) method. Pushing a new view controller object onto the stack causes the view of that view controller to be displayed and the navigation controls to be updated to reflect the change. You typically push view controllers in response to the user selecting an item that leads to the next level in your information hierarchy.

In addition to pushing view controllers onto the navigation stack, you can also pop them using the `popViewControllerAnimated:` (page 387) method. Although you can pop view controllers yourself, the navigation controller also provides a back button (when appropriate) that pops the top view controller in response to user interactions.

A navigation controller object notifies its delegate object in response to changes in the active view controller. The delegate object is a custom object provided by your application that conforms to the `UINavigationControllerDelegate` protocol. You can use the methods of this protocol to respond to the change and perform additional setup or cleanup tasks.

For more information about how to integrate navigation controllers into your application, see *View Controller Programming Guide for iOS*.

Navigation Controller Views

Because the `UINavigationController` class inherits from the `UIViewController` class, navigation controllers have their own view that is accessible through the `view` (page 761) property. When deploying a navigation interface, you must install this view as the root of whatever view hierarchy you are creating. For example, if you are deploying the navigation interface by itself, you would make this view the main subview of your window. To install a navigation interface inside a tab bar interface, you would install the navigation controller's view as the root view of the appropriate tab.

The view for a navigation controller is just a container for several other views, including a navigation bar, an optional toolbar, and the view containing your custom content. Figure 34-2 shows how these views are assembled to present the overall navigation interface. Although the content of the navigation bar and toolbar views changes, the views themselves do not. Only the custom content view changes to reflect the view controller that is at the top of the navigation stack.

Figure 34-2 The views of a navigation controller

Note: Because the amount of space available for the custom view can vary (depending on the size of the other navigation views), your custom view's `autoresizingMask` (page 697) property should be set to have a flexible width and height. Before displaying your view, the navigation controller automatically positions and sizes it to fit the available space.

The navigation controller is responsible for managing the configuration and display of the navigation bar and navigation toolbar. You must never modify these views directly. Instead, you should manipulate them through the methods and properties of the `UINavigationController` class. You can hide and show the navigation bar using the `navigationBarHidden` (page 383) property or `setNavigationBarHidden:animated:` (page 389) method. To specify custom items for the navigation bar, you configure the displayed view controller as described in "Updating the Navigation Bar" (page 380). For information on how to specify items for the navigation toolbar, see "Displaying a Toolbar" (page 381).

With only a few exceptions, you should never modify the navigation bar object directly. It is permissible to modify the `barStyle` (page 372) or `translucent` (page 374) properties of the navigation bar but you must never change its `frame` (page 701), `bounds` (page 698), or `alpha` (page 696) values directly. In addition, the

navigation controller object builds the contents of the navigation bar dynamically using the navigation items (instances of the `UINavigationItem` class) associated with the view controllers on the navigation stack. To change the contents of the navigation bar, you must therefore configure the navigation items for your custom view controllers. For more information about navigation items, see *UINavigationItem Class Reference*.

Updating the Navigation Bar

When the user changes the top-level view controller, whether by pushing or popping a view controller or changing the contents of the navigation stack directly, the navigation controller updates the navigation bar accordingly. Specifically, the navigation controller updates the bar button items displayed in each of the three navigation bar positions: left, middle, and right. Bar button items are instances of the `UIBarButtonItem` class. You can create items with custom content or create standard system items depending on your needs. For more information about how to create bar button items, see *UIBarButtonItem Class Reference*.

The bar button item on the left side of the navigation bar allows for navigation back to the previous view controller on the navigation stack. The navigation controller updates the left side of the navigation bar as follows:

- If the new top-level view controller has a custom left bar button item, that item is displayed. To specify a custom left bar button item, set the `leftBarButtonItem` (page 395) property of the view controller's navigation item.
- If the top-level view controller does not have a custom left bar button item, but the navigation item of the previous view controller has a valid item in its `backBarButtonItem` (page 394) property, the navigation bar displays that item.
- If a custom bar button item is not specified by either of the view controllers, a default back button is used and its title is set to the value of the `title` (page 760) property of the previous view controller—that is, the view controller one level down on the stack. (If there is only one view controller on the navigation stack, no back button is displayed.)

The navigation controller updates the middle of the navigation bar as follows:

- If the new top-level view controller has a custom title view, the navigation bar displays that view in place of the default title view. To specify a custom title view, set the `titleView` (page 397) property of the view controller's navigation item.
- If no custom title view is set, the navigation bar displays a label containing the view controller's default title. The string for this label is usually obtained from the `title` (page 760) property of the view controller itself. If you want to display a different title than the one associated with the view controller, set the `title` property of the view controller's navigation item instead.

The navigation controller updates the right side of the navigation bar as follows:

- If the new top-level view controller has a custom right bar button item, that item is displayed. To specify a custom right bar button item, set the `rightBarButtonItem` (page 396) property of the view controller's navigation item.
- If no custom right bar button item is specified, the navigation bar displays nothing on the right side of the bar.

The navigation controller updates the navigation bar each time the top view controller changes. Thus, these changes occur each time a view controller is pushed onto the stack or popped from it. When you animate a push or pop operation, the navigation controller similarly animates the change in navigation bar content.

Displaying a Toolbar

In iOS 3.0 and later, navigation controller objects make it easy to provide a custom toolbar for each screen of a navigation interface. The navigation controller object now manages an optional toolbar in its view hierarchy. When displayed, this toolbar obtains its current set of items from the `toolbarItems` (page 761) property of the active view controller. When the active view controller changes, the navigation controller updates the toolbar items to match the new view controller, animating the new items into position when appropriate.

The navigation toolbar is hidden by default but you can show it for your navigation interface by calling the `setToolbarHidden:animated:` (page 389) method of your navigation controller object. If not all of your view controllers support toolbar items, your delegate object can call this method to toggle the visibility of the toolbar during subsequent push and pop operations.

Tasks

Creating Navigation Controllers

- `initWithRootViewController:` (page 385)
  Initializes and returns a newly created navigation controller.

Accessing Items on the Navigation Stack

`topViewController` (page 384) *property*
The view controller at the top of the navigation stack. (read-only)

`visibleViewController` (page 385) *property*
The view controller associated with the currently visible view in the navigation interface. (read-only)

`viewControllers` (page 384) *property*
The view controllers currently on the navigation stack.

- `setViewControllers:animated:` (page 390)
  Replaces the view controllers currently managed by the navigation controller with the specified items.

Pushing and Popping Stack Items

- `pushViewController:animated:` (page 388)
  Pushes a view controller onto the receiver's stack and updates the display.
- `popViewControllerAnimated:` (page 387)
  Pops the top view controller from the navigation stack and updates the display.

- `popToRootViewControllerAnimated:` (page 386)
  Pops all the view controllers on the stack except the root view controller and updates the display.
- `popToViewController:animated:` (page 387)
  Pops view controllers until the specified view controller is at the top of the navigation stack.

Configuring Navigation Bars

`navigationBar` (page 383) *property*
The navigation bar managed by the navigation controller. (read-only)

`navigationBarHidden` (page 383) *property*
A Boolean value that determines whether the navigation bar is hidden.

- `setNavigationBarHidden:animated:` (page 389)
  Sets whether the navigation bar is hidden.

Accessing the Delegate

`delegate` (page 382) *property*
The receiver's delegate or `nil` if it doesn't have a delegate.

Configuring Custom Toolbars

`toolbar` (page 383) *property*
The custom toolbar associated with the navigation controller. (read-only)

- `setToolbarHidden:animated:` (page 389)
  Changes the visibility of the navigation controller's built-in toolbar.

`toolbarHidden` (page 384) *property*
A Boolean indicating whether the navigation controller's built-in toolbar is visible.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

Declared In
UINavigationController.h

navigationBar

The navigation bar managed by the navigation controller. (read-only)

```
@property(nonatomic, readonly) UINavigationBar *navigationBar
```

Discussion
It is permissible to modify the barStyle (page 372) or translucent (page 374) properties of the navigation bar but you must never change its frame (page 701), bounds (page 698), or alpha (page 696) values directly. To show or hide the navigation bar, you should always do so through the navigation controller by changing its navigationBarHidden (page 383) property or calling the setNavigationBarHidden:animated: (page 389) method.

Availability
Available in iOS 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. Setting this property does not animate the hiding or showing of the navigation bar; use setNavigationBarHidden:animated: (page 389) for that purpose.

Availability
Available in iOS 2.0 and later.

Declared In
UINavigationController.h

toolbar

The custom toolbar associated with the navigation controller. (read-only)

```
@property(nonatomic,readonly) UIToolbar *toolbar
```

Discussion
This property contains a reference to the built-in toolbar managed by the navigation controller. Access to this toolbar is provided solely for clients that want to present an action sheet from the toolbar. You should not modify the UIToolbar object directly.

Management of this toolbar's contents is done through the custom view controllers associated with this navigation controller. For each view controller on the navigation stack, you can assign a custom set of toolbar items using the setToolbarItems:animated: (page 770) method of UIViewController.

The visibility of this toolbar is controlled by the `toolbarHidden` property. The toolbar also obeys the `hidesBottomBarWhenPushed` (page 755) property of the currently visible view controller and hides and shows itself automatically as needed.

Availability
Available in iOS 3.0 and later.

See Also
`@property toolbarHidden` (page 384)

Declared In
`UINavigationController.h`

toolbarHidden

A Boolean indicating whether the navigation controller's built-in toolbar is visible.

```
@property(nonatomic,getter=isToolbarHidden) BOOL toolbarHidden
```

Discussion
If this property is set to `YES`, the toolbar is not visible. The default value of this property is `YES`.

Availability
Available in iOS 3.0 and later.

See Also
`@property toolbar` (page 383)
`- setToolbarHidden:animated:` (page 389)

Declared In
`UINavigationController.h`

topViewController

The view controller at the top of the navigation stack. (read-only)

```
@property(nonatomic, readonly, retain) UIViewController *topViewController
```

Availability
Available in iOS 2.0 and later.

See Also
`@property visibleViewController` (page 385)
`@property viewControllers` (page 384)

Declared In
`UINavigationController.h`

viewControllers

The view controllers currently on the navigation stack.

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

Discussion
The root 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.

Assigning a new array of view controllers to this property is equivalent to calling the `setViewControllers:animated:` method with the *`animated`* parameter set to `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`@property topViewController` (page 384)
`@property visibleViewController` (page 385)
`- setViewControllers:animated:` (page 390)

Declared In
`UINavigationController.h`

visibleViewController

The view controller associated with the currently visible view in the navigation interface. (read-only)

```
@property(nonatomic, readonly, retain) UIViewController *visibleViewController
```

Discussion
The currently visible view can belong either to the view controller at the top of the navigation stack or to a view controller that was presented modally.

Availability
Available in iOS 2.0 and later.

See Also
`@property topViewController` (page 384)
`@property viewControllers` (page 384)

Declared In
`UINavigationController.h`

Instance Methods

initWithRootViewController:

Initializes and returns a newly created navigation controller.

```
- (id)initWithRootViewController:(UIViewController *)rootViewController
```

Parameters

rootViewController

The view controller that resides at the bottom of the navigation stack. This object cannot be an instance of the `UITabBarController` class.

Return Value

The initialized navigation controller object or `nil` if there was a problem initializing the object.

Discussion

This is a convenience method for initializing the receiver and pushing a root view controller onto the navigation stack. Every navigation stack must have at least one view controller to act as the root.

Availability

Available in iOS 2.0 and later.

Related Sample Code

BonjourWeb

ToolbarSearch

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. For information on how the navigation bar is updated, see "Updating the Navigation Bar" (page 380).

Availability

Available in iOS 2.0 and later.

See Also

- `pushViewController:animated:` (page 388)
- `popViewControllerAnimated:` (page 387)
- `popToViewController:animated:` (page 387)

Declared In

`UINavigationController.h`

popToViewController:animated:

Pops view controllers until the specified view controller is at the top of the navigation stack.

```
- (NSArray *)popToViewController:(UIViewController *)viewController
    animated:(BOOL)animated
```

Parameters

viewController

The view controller that you want to be at the top of 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 containing the view controllers that were popped from the stack.

Discussion
For information on how the navigation bar is updated, see "Updating the Navigation Bar" (page 380).

Availability
Available in iOS 2.0 and later.

See Also
- `pushViewController:animated:` (page 388)
- `popViewControllerAnimated:` (page 387)
- `popToRootViewControllerAnimated:` (page 386)

Declared In
`UINavigationController.h`

popViewControllerAnimated:

Pops the top view controller from the navigation 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 new top of the stack the active view controller. If the view controller at the top of the stack is the root view controller, this method does nothing. In other words, you cannot pop the last item on the stack.

In addition to displaying the view associated with the new view controller at the top of the stack, this method also updates the navigation bar and tool bar accordingly. In iOS 3.0 and later, the contents of the built-in navigation toolbar are updated to reflect the toolbar items of the new view controller. For information on how the navigation bar is updated, see "Updating the Navigation Bar" (page 380).

Availability
Available in iOS 2.0 and later.

See Also
- `pushViewController:animated:` (page 388)
- `popToRootViewControllerAnimated:` (page 386)
- `popToViewController:animated:` (page 387)

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. This object cannot be an instance of tab bar controller and it must not already be on the navigation stack.

animated
Specify `YES` to animate the transition or `NO` if you do not want the transition to be animated. You might specify `NO` if you are setting up the navigation controller at launch time.

Discussion
The object in the *viewController* parameter becomes the top view controller on the navigation stack. Pushing a view controller results in the display of the view it manages. How that view is displayed is determined by the animated parameter. If the *animated* parameter is `YES`, the view is animated into position; otherwise, the view is simply displayed in place. The view is automatically resized to fit between the navigation bar and toolbar (if present) before it is displayed.

In addition to displaying the view associated with the new view controller at the top of the stack, this method also updates the navigation bar and tool bar accordingly. In iOS 3.0 and later, the contents of the built-in navigation toolbar are updated to reflect the toolbar items of the new view controller. For information on how the navigation bar is updated, see "Updating the Navigation Bar" (page 380).

Important: In iOS 2.2 and later, if the object in the *viewController* parameter is already on the navigation stack, this method throws an exception. In earlier versions of iOS, the method simply does nothing.

Availability
Available in iOS 2.0 and later.

See Also
- `popViewControllerAnimated:` (page 387)
- `popToRootViewControllerAnimated:` (page 386)
- `popToViewController:animated:` (page 387)

Related Sample Code
BonjourWeb

Declared In
UINavigationController.h

setNavigationBarHidden:animated:

Sets whether the navigation bar is hidden.

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

Parameters

hidden
Specify YES to hide the navigation bar or NO to show it.

animated
Specify YES if you want to animate the change in visibility or NO if you want the navigation bar to appear immediately.

Discussion
For animated transitions, the duration of the animation is specified by the value in the UINavigationControllerHideShowBarDuration (page 391) constant.

Availability
Available in iOS 2.0 and later.

See Also
@property navigationBarHidden (page 383)

Declared In
UINavigationController.h

setToolbarHidden:animated:

Changes the visibility of the navigation controller's built-in toolbar.

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

Parameters

hidden
Specify YES to hide the toolbar or NO to show it.

animated
Specify YES if you want the toolbar to be animated on or off the screen.

Discussion
You can use this method to animate changes to the visibility of the built-in toolbar.

Calling this method with the *animated* parameter set to NO is equivalent to setting the value of the toolbarHidden property directly. The toolbar simply appears or disappears depending on the value in the *hidden* parameter.

Availability
Available in iOS 3.0 and later.

See Also
@property toolbarHidden (page 384)

Declared In
`UINavigationController.h`

setViewControllers:animated:

Replaces the view controllers currently managed by the navigation controller with the specified items.

```
- (void)setViewControllers:(NSArray *)viewControllers animated:(BOOL)animated
```

Parameters

viewControllers

The view controllers to place in the stack. The front-to-back order of the controllers in this array represents the new bottom-to-top order of the controllers in the navigation stack. Thus, the last item added to the array becomes the top item of the navigation stack.

animated

If `YES`, animate the pushing or popping of the top view controller. If `NO`, replace the view controllers without any animations.

Discussion
You can use this method to update or replace the current view controller stack without pushing or popping each controller explicitly. In addition, this method lets you update the set of controllers without animating the changes, which might be appropriate at launch time when you want to return the navigation controller to a previous state.

If animations are enabled, this method decides which type of transition to perform based on whether the last item in the *items* array is already in the navigation stack. If the view controller is currently in the stack, but is not the topmost item, this method uses a pop transition; if it is the topmost item, no transition is performed. If the view controller is not on the stack, this method uses a push transition. Only one transition is performed, but when that transition finishes, the entire contents of the stack are replaced with the new view controllers. For example, if controllers A, B, and C are on the stack and you set controllers D, A, and B, this method uses a pop transition and the resulting stack contains the controllers D, A, and B.

Availability
Available in iOS 3.0 and later.

Declared In
`UINavigationController.h`

Constants

UINavigationControllerHideShowBarDuration

A global variable that specifies a preferred duration when animating the navigation bar.

```
extern const CGFloat UINavigationControllerHideShowBarDuration
```

Constants

`UINavigationControllerHideShowBarDuration`

This variable specifies the duration when animating the navigation bar. Note that this is a constant value, so it cannot be set.

Available in iOS 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 iOS 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 397) 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 394) property if you want to use a different title when this item is the back item. The `backBarButtonItem` (page 394) 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 398) and `setRightBarButtonItem:animated:` (page 399) methods to change the left and right views; you can specify that the change be animated. Use the `titleView` (page 397) 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 397)
Returns a navigation item initialized with the specified title.

Getting and Setting Properties

`title` (page 396) *property*
The navigation item's title displayed in the center of the navigation bar.

`prompt` (page 396) *property*
A single line of text displayed at the top of the navigation bar.

`backBarButtonItem` (page 394) *property*
The bar button item to use when this item is represented by a back button on the navigation bar.

`hidesBackButton` (page 395) *property*
A Boolean value that determines whether the back button is hidden.

- `setHidesBackButton:animated:` (page 398)
Sets whether the back button is hidden, optionally animating the transition.

Customizing Views

`titleView` (page 397) *property*
A custom view displayed in the center of the navigation bar when this item is the top item.

`leftBarButtonItem` (page 395) *property*
A custom bar item displayed on the left of the navigation bar when this item is the top item.

`rightBarButtonItem` (page 396) *property*
A custom bar item displayed on the right of the navigation bar when this item is the top item.

- `setLeftBarButtonItem:animated:` (page 398)
Sets the custom bar item, optionally animating the transition to the view.

- `setRightBarButtonItem:animated:` (page 399)
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 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 iOS 2.0 and later.

See Also
`@property backItem` (page 371)
`@property hidesBackButton` (page 395)
`- setHidesBackButton:animated:` (page 398)

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 iOS 2.0 and later.

See Also
`@property backItem` (page 371)
`@property backBarButtonItem` (page 394)
`- setHidesBackButton:animated:` (page 398)

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 iOS 2.0 and later.

See Also
`@property rightBarButtonItem` (page 396)
`- setLeftBarButtonItem:animated:` (page 398)
`- setRightBarButtonItem:animated:` (page 399)

Related Sample Code
AddMusic

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 iOS 2.0 and later.

Related Sample Code
BonjourWeb

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 iOS 2.0 and later.

See Also
`@property leftBarButtonItem` (page 395)
`- setLeftBarButtonItem:animated:` (page 398)
`- setRightBarButtonItem:animated:` (page 399)

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 iOS 2.0 and later.

See Also
`- initWithTitle:` (page 397)
`UINavigationItem` (page 393)
`@property titleView` (page 397)

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 395) is not `nil`.

Custom views can contain buttons. Use the `buttonWithType:` (page 184) 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 iOS 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 iOS 2.0 and later.

See Also
`@property title` (page 396)

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 iOS 2.0 and later.

See Also

@property backItem (page 371)
@property backBarButtonItem (page 394)
@property hidesBackButton (page 395)

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 iOS 2.0 and later.

See Also

@property leftBarButtonItem (page 395)
@property rightBarButtonItem (page 396)
- setRightBarButtonItem:animated: (page 399)

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 iOS 2.0 and later.

See Also

`@property leftBarButtonItem` (page 395)

`@property rightBarButtonItem` (page 396)

`- setLeftBarButtonItem:animated:` (page 398)

Declared In

`UINavigationBar.h`

UINib Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 4.0 and later.
Declared in	UINib.h

Overview

Instances of the `UINib` class serve as object wrappers, or containers, for Interface Builder nib files. An `UINib` object caches the contents of a nib file in memory, ready for unarchiving and instantiation. When your application needs to instantiate the contents of the nib file it can do so without having to load the data from the nib file first, improving performance. The `UINib` object can automatically release this cached nib data to free up memory for your application under low-memory conditions, reloading that data the next time your application instantiates the nib. Your application should use `UINib` objects whenever it needs to repeatedly instantiate the same nib data. For example, if your table view uses a nib file to instantiate table view cells, caching the nib in a `UINib` object can provide a significant performance improvement.

When you create an `UINib` object using the contents of a nib file, the object loads the object graph in the referenced nib file, but it does not yet unarchive it. To unarchive all of the nib data and thus truly instantiate the nib your application calls the `instantiateWithOwner:options:` method on the `UINib` object. The steps that the `UINib` object follows to instantiate the nib's object graph are described in detail in *Resource Programming Guide*.

Tasks

Creating a Nib Object

`+ nibWithNibName:bundle:` (page 402)
Returns an `UINib` object initialized to the nib file in the specified bundle.

`+ nibWithData:bundle:` (page 402)
Creates an `UINib` object from nib data stored in memory.

Instantiating a Nib

`- instantiateWithOwner:options:` (page 403)
Unarchives and instantiates the in-memory contents of the receiver's nib file, creating a distinct object tree and set of top level objects.

Class Methods

nibWithData:bundle:

Creates an `UINib` object from nib data stored in memory.

```
+ (UINib *)nibWithData:(NSData *)data bundle:(NSBundle *)bundleOrNil
```

Parameters

data
A block of memory that contains nib data.

bundleOrNil
The bundle in which to search for resources referenced by the nib. If you specify `nil`, this method looks for the nib file in the main bundle.

Return Value
The initialized `NSNib` object or `nil` if there were errors during initialization.

Discussion
The `UINib` object looks for the nib file in the bundle's language-specific project directories first, followed by the `Resources` directory.

The preferred mechanism for instantiating `UINib` objects is with the `nibWithNibName:bundle:` class method. A `UINib` object instantiated using the `nibWithData:bundle:` class method cannot release the cached data under low memory conditions. Your application should be prepared to release the `UINib` object and the data under low memory conditions, recreating both the next time the application needs to instantiate the nib.

Availability
Available in iOS 4.0 and later.

Declared In
`UINib.h`

nibWithNibName:bundle:

Returns an `UINib` object initialized to the nib file in the specified bundle.

```
+ (UINib *)nibWithNibName:(NSString *)name bundle:(NSBundle *)bundleOrNil
```

Parameters

name
The name of the nib file, without any leading path information.

bundleOrNil

The bundle in which to search for the nib file. If you specify `nil`, this method looks for the nib file in the main bundle.

Return Value
The initialized `UINib` object or `nil` if there were errors during initialization or the nib file could not be located.

Discussion
The `UINib` object looks for the nib file in the bundle's language-specific project directories first, followed by the `Resources` directory.

Availability
Available in iOS 4.0 and later.

Declared In
`UINib.h`

Instance Methods

instantiateWithOwner:options:

Unarchives and instantiates the in-memory contents of the receiver's nib file, creating a distinct object tree and set of top level objects.

```
- (NSArray *)instantiateWithOwner:(id)ownerOrNil options:(NSDictionary *)optionsOrNil
```

Parameters

ownerOrNil

The object to use as the owner of the nib file. If the nib file has an owner, you must specify a valid object for this parameter.

optionsOrNil

A dictionary containing the options to use when opening the nib file. For a list of available keys for this dictionary, see *NSBundle UIKit Additions Reference*.

Return Value
An autoreleased `NSArray` object containing the top-level objects from the nib file.

Discussion
You can use this method to instantiate the objects in a nib and provide them to your code. This method unarchives each object, initializes it, sets its properties to their configured values, and reestablishes any connections to other objects. 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 `UINibExternalObjects` 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 iOS 4.0 and later.

Declared In
`UINib.h`

UIPageControl Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 224) event for handling by the delegate. The delegate can then evaluate the `currentPage` (page 406) 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 406) *property*
The current page, shown by the receiver as a white dot.

`numberOfPages` (page 407) *property*
The number of pages the receiver shows (as dots).

`hidesForSinglePage` (page 407) *property*
A Boolean value that controls whether the page indicator is hidden when there is only one page.

Updating the Page Display

`defersCurrentPageDisplay` (page 406) *property*
A Boolean value that controls when the current page is displayed.

- `updateCurrentPageDisplay` (page 408)
Updates the page indicator to the current page.

Resizing the Control

- `sizeForNumberOfPages:` (page 407)
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 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 407) minus 1.

Availability
Available in iOS 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 408). Set the value to `NO` (the default) to have the page indicator updated immediately.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 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 406). The class ignores this method if the value of `defersPageIndicatorUpdate` (page 406) is `NO`. Setting the `currentPage` value directly updates the indicator immediately.

Availability
Available in iOS 2.0 and later.

Declared In
`UIPageControl.h`

UIPanGestureRecognizer Class Reference

Inherits from	UIGestureRecognizer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIPanGestureRecognizer.h
Companion guide	Event Handling Guide for iOS

Overview

`UIPanGestureRecognizer` is a concrete subclass of `UIGestureRecognizer` that looks for panning (dragging) gestures. The user must be pressing one or more fingers on a view while they pan it. Clients implementing the action method for this gesture recognizer can ask it for the current translation and velocity of the gesture.

A panning gesture is continuous. It begins (`UIGestureRecognizerStateBegan` (page 296)) when the minimum number of fingers allowed (`minimumNumberOfTouches` (page 410)) has moved enough to be considered a pan. It changes (`UIGestureRecognizerStateChanged` (page 296)) when a finger moves while at least the minimum number of fingers are pressed down. It ends (`UIGestureRecognizerStateEnded` (page 296)) when all fingers are lifted.

Clients of this class can, in their action methods, query the `UIPanGestureRecognizer` object for the current translation of the gesture (`translationInView:` (page 411)) and the velocity of the translation (`velocityInView:` (page 412)). They can specify the view whose coordinate system should be used for the translation and velocity values. Clients may also reset the translation to a desired value.

Tasks

Configuring the Gesture Recognizer

`maximumNumberOfTouches` (page 410) *property*
The maximum number of fingers that can be touching the view for this gesture to be recognized.

`minimumNumberOfTouches` (page 410) *property*
The minimum number of fingers that can be touching the view for this gesture to be recognized.

Tracking the Location and Velocity of the Gesture

- `translationInView:` (page 411)
  The translation of the pan gesture in the coordinate system of the specified view.
- `setTranslation:inView:` (page 411)
  Sets the translation value in the coordinate system of the specified view.
- `velocityInView:` (page 412)
  The velocity of the pan gesture in the coordinate system of the specified view.

Properties

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

maximumNumberOfTouches

The maximum number of fingers that can be touching the view for this gesture to be recognized.

```
@property(nonatomic) NSUInteger maximumNumberOfTouches
```

Discussion
The default value is `NSUIntegerMax`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPanGestureRecognizer.h`

minimumNumberOfTouches

The minimum number of fingers that can be touching the view for this gesture to be recognized.

```
@property(nonatomic) NSUInteger minimumNumberOfTouches
```

Discussion
The default value is 1.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPanGestureRecognizer.h`

Instance Methods

setTranslation:inView:

Sets the translation value in the coordinate system of the specified view.

```
- (void)setTranslation:(CGPoint)translation inView:(UIView *)view
```

Parameters

translation

A point that identifies the new translation value.

view

A view in whose coordinate system the translation is to occur.

Discussion

Changing the translation value resets the velocity of the pan.

Availability

Available in iOS 3.2 and later.

See Also

- translationInView: (page 411)

Declared In

UIPanGestureRecognizer.h

translationInView:

The translation of the pan gesture in the coordinate system of the specified view.

```
- (CGPoint)translationInView:(UIView *)view
```

Parameters

view

The view in whose coordinate system the translation of the pan gesture should be computed. If you want to adjust a view's location to keep it under the user's finger, request the translation in that view's superview's coordinate system.

Return Value

A point identifying the new location of a view in the coordinate system of its designated superview.

Availability

Available in iOS 3.2 and later.

See Also

- setTranslation:inView: (page 411)

Declared In

UIPanGestureRecognizer.h

velocityInView:

The velocity of the pan gesture in the coordinate system of the specified view.

```
- (CGPoint)velocityInView:(UIView *)view
```

Parameters

view

The view in whose coordinate system the velocity of the pan gesture is computed.

Return Value

The velocity of the pan gesture, which is expressed in points per second. The velocity is broken into horizontal and vertical components.

Availability

Available in iOS 3.2 and later.

See Also

`- translationInView:` (page 411)

Declared In

`UIPanGestureRecognizer.h`

UIPasteboard Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	UIPasteboard.h
Companion guide	Device Features Programming Guide

Overview

The `UIPasteboard` class enables an application to share data within the application or with another application using system-wide or application-specific pasteboards.

Typically, an object in the application writes data to a pasteboard when the user requests a copy or cut operation on a selection in the user interface. Another object in the same or different application then reads that data from the pasteboard and presents it to the user at a new location; this usually happens when the user requests a paste operation.

A pasteboard is a named region of memory where data can be shared. There are two system pasteboards: the General pasteboard (`UIPasteboardNameGeneral` (page 430)) and the Find pasteboard (`UIPasteboardNameFind` (page 430). You can use the General pasteboard for copy-paste operations involving any kind of data; the Find pasteboard, which is used in search operations, holds the most recent string value in the search bar. Applications can also create pasteboards for their own use or for use by a family of related applications. Pasteboards must be identified by a unique names. You may also mark an application pasteboard as persistent, so that it continues to exist past the termination of the application and past system reboots. System pasteboards are persistent by default.

When you write an object to a pasteboard, it is stored as a pasteboard item. A pasteboard item is one or more key-value pairs where the key is a string that identifies the representation type of the value. Having multiple representation types per pasteboard item makes it more possible for one application to share data with another application without having to know specific capabilities of that application. For example, the source application could write the same image to the pasteboard in PNG, JPEG, and GIF data formats. If the receiving application can only handle GIF images, it can still obtain the pasteboard data.

A Uniform Type Identifier (UTI) is frequently used for a representation type (sometimes called a pasteboard type). For example, you could use `kUTTypeJPEG` (a constant for `public.jpeg`) as a representation type for JPEG data. However, applications are free to use any string they want for a representation type; however, for application-specific data types, it is recommended that you use reverse-DNS notation to ensure the uniqueness of the type (for example, `com.myCompany.myApp.myType`).

Note: For a discussion of Uniform Type Identifiers and a list of common ones, see *Uniform Type Identifiers Overview*.

`UIPasteboard` provides methods for reading and writing single pasteboard items at a time as well as multiple pasteboard items. The data written and read can be in two general forms. If the data to be written is a property-list objects or can be converted to such an object, use a method such as `setValue:forPasteboardType:` (page 428) to write it to the pasteboard. If the data is binary (say, image data) or can't be converted to a property-list type, you would use the `setData:forPasteboardType:` (page 427) to write it to the pasteboard. For `UIPasteboard`, the classes of the property-list objects are `NSString`, `NSArray`, `NSDictionary`, `NSDate`, `NSNumber` and `NSURL`. The class also provides convenience methods for writing and reading strings, images, URLs, and colors to and from single or multiple pasteboard items.

Although `UIPasteboard` is central to copy-paste operations, several other UIKit classes and protocols are used in these operations as well:

- `UIMenuController` —Displays a menu with Copy, Cut, Paste, Select, and Select All commands above or below the selection.
- `UIResponder` — Responders implement the `canPerformAction:withSender:` (page 461) to enable or disable commands in the above-mentioned menu based on the current context.
- `UIResponderStandardEditActions` — Responders implement methods declared in this informal protocol to handle the chosen menu commands (for example, `copy:` and `paste:`).

An application that implements copy-paste usually has to handle the management and presentation of selections in its user interface. It must also coordinate the addition and removal of items via paste and cut operations with its data model.

Tasks

Getting and Removing Pasteboards

`+ generalPasteboard` (page 421)
Returns the general pasteboard, which is used for general copy-paste operations

`+ pasteboardWithName:create:` (page 421)
Returns a pasteboard identified by name, optionally creating it if it doesn't exist.

`+ pasteboardWithUniqueName` (page 422)
Returns an application pasteboard identified by a unique system-generated name.

`+ removePasteboardWithName:` (page 423)
Invalidates the designated application pasteboard.

Getting and Setting Pasteboard Attributes

`name` (page 418) *property*
The name of the pasteboard. (read-only)

`persistent` (page 419) *property*
A Boolean value that indicates whether the pasteboard is persistent.

`changeCount` (page 416) *property*
The number of times the pasteboard's contents have changed. (read-only)

Determining Types of Single Pasteboard Items

- `pasteboardTypes` (page 426)
Returns the types of the first item on the pasteboard.
- `containsPasteboardTypes:` (page 424)
Returns whether the pasteboard holds data of the specified representation type.

Getting and Setting Single Pasteboard Items

- `dataForPasteboardType:` (page 425)
Returns the data in the pasteboard for the given representation type.
- `valueForPasteboardType:` (page 428)
Returns an object in the pasteboard for the given representation type.
- `setData:forPasteboardType:` (page 427)
Puts data in the pasteboard for the specified representation type.
- `setValue:forPasteboardType:` (page 428)
Puts a property-list object in the pasteboard for the specified representation type.

Determining the Types of Multiple Pasteboard Items

`numberOfItems` (page 419) *property*
Returns the number of items in the pasteboard (read-only)

- `pasteboardTypesForItemSet:` (page 427)
Returns an array of representation types for each specified pasteboard item.
- `itemSetWithPasteboardTypes:` (page 426)
Returns an index set identifying pasteboard items having the specified representation types.
- `containsPasteboardTypes:inItemSet:` (page 424)
Returns whether the specified pasteboard items contain data of the given representation types.

Getting and Setting Multiple Pasteboard Items

`items` (page 418) *property*
The pasteboard items on the pasteboard.

- `dataForPasteboardType:inItemSet:` (page 425)
Returns the data objects in the indicated pasteboard items that have the given representation type.
- `valuesForPasteboardType:inItemSet:` (page 429)
Returns the property-list objects in the indicated pasteboard items that have the given representation type.

- `addItems:` (page 423)
  Appends pasteboard items to the current contents of the pasteboard.

Getting and Setting Pasteboard Items of Standard Data Types

`string` (page 419) *property*
The string value of the first pasteboard item.

`strings` (page 420) *property*
An array of strings in all pasteboard items.

`image` (page 417) *property*
The image object of the first pasteboard item.

`images` (page 418) *property*
An array of image objects in all pasteboard items.

`URL` (page 420) *property*
The URL object of the first pasteboard item.

`URLs` (page 421) *property*
An array of URL objects in all pasteboard items.

`color` (page 417) *property*
The color object of the first pasteboard item.

`colors` (page 417) *property*
An array of color objects in all pasteboard items.

Properties

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

changeCount

The number of times the pasteboard's contents have changed. (read-only)

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

Discussion
Whenever the contents of a pasteboard changes—specifically, when pasteboard items are added, modified, or removed—`UIPasteboard` increments the value of this property. After it increments the change count, UIPasteboard posts the notifications named `UIPasteboardChangedNotification` (page 432) (for additions and modifications) and `UIPasteboardRemovedNotification` (page 432) (for removals). These notifications include (in the `userInfo` dictionary) the types of the pasteboard items added or removed. Because `UIPasteboard` waits until the end of the current event loop before incrementing the change count, notifications can be batched. The class also updates the change count when an application reactivates and another application has changed the pasteboard contents. When users restart a device, the change count is reset to zero.

Availability
Available in iOS 3.0 and later.

Declared In
UIPasteboard.h

color

The color object of the first pasteboard item.

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

Discussion
The value stored in this property is a UIColor object. The associated array of representation types is UIPasteboardTypeListColor (page 431), which includes type . Setting this property replaces all current items in the pasteboard with the new item. If the first item has no value of the indicated type, nil is returned.

Availability
Available in iOS 3.0 and later.

Declared In
UIPasteboard.h

colors

An array of color objects in all pasteboard items.

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

Discussion
The value stored in this property is an array of UIColor objects. The associated array of representation types is UIPasteboardTypeListColor (page 431), which includes type . Setting this property replaces all current items in the pasteboard with the new items. The returned array may have fewer objects than the number of pasteboard items; this happens if a pasteboard item does not have a value of the indicated type.

Availability
Available in iOS 3.0 and later.

Declared In
UIPasteboard.h

image

The image object of the first pasteboard item.

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

Discussion
The value stored in this property is a UIImage object. The associated array of representation types is UIPasteboardTypeListImage (page 431), which includes types kUTTypePNG and kUTTypeJPEG. Setting this property replaces all current items in the pasteboard with the new item. If the first item has no value of the indicated type, nil is returned.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

images

An array of image objects in all pasteboard items.

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

Discussion
The value stored in this property is an array of `UIImage` objects. The associated array of representation types is `UIPasteboardTypeListImage` (page 431), which includes types `kUTTypePNG` and `kUTTypeJPEG`. Setting this property replaces all current items in the pasteboard with the new items. The returned array may have fewer objects than the number of pasteboard items; this happens if a pasteboard item does not have a value of the indicated type.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

items

The pasteboard items on the pasteboard.

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

Discussion
The value of the property is an array of dictionaries. Each dictionary represents a pasteboard item, with the key being the representation type and the value the data object or property-list object associated with that type. Setting this property replaces all of the current pasteboard items.

Availability
Available in iOS 3.0 and later.

See Also
`- addItems:` (page 423)

Declared In
`UIPasteboard.h`

name

The name of the pasteboard. (read-only)

```
@property(readonly, nonatomic) NSString *name
```

Discussion
Names of application pasteboard objects should be unique across installed applications. If the object is a system pasteboard, this property returns one of the constants described in "Pasteboard Names" (page 430).

Availability
Available in iOS 3.0 and later.

See Also
`+ pasteboardWithName:create:` (page 421)
`+ pasteboardWithUniqueName` (page 422)

Declared In
`UIPasteboard.h`

numberOfItems

Returns the number of items in the pasteboard (read-only)

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

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

persistent

A Boolean value that indicates whether the pasteboard is persistent.

```
@property(getter=isPersistent, nonatomic) BOOL persistent
```

Discussion
When a pasteboard is persistent, it continues to exist past application terminations and across system reboots. Application pasteboards that are not persistent only last until the owning (creating) application quits. System pasteboards are persistent. Application pasteboards by default are not persistent. A persistent application pasteboard is removed when the application that created it is uninstalled.

Availability
Available in iOS 3.0 and later.

See Also
`+ pasteboardWithName:create:` (page 421)
`+ pasteboardWithUniqueName` (page 422)

Declared In
`UIPasteboard.h`

string

The string value of the first pasteboard item.

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

Discussion
The value stored in this property is an `NSString` object. The associated array of representation types is `UIPasteboardTypeListString` (page 430), which includes type `kUTTypeUTF8PlainText`. Setting this property replaces all current items in the pasteboard with the new item. If the first item has no value of the indicated type, `nil` is returned.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

strings

An array of strings in all pasteboard items.

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

Discussion
The value stored in this property is an array of `NSString` objects. The associated array of representation types is `UIPasteboardTypeListString` (page 430), which includes type `kUTTypeUTF8PlainText`.Setting this property replaces all current items in the pasteboard with the new items. The returned array may have fewer objects than the number of pasteboard items; this happens if a pasteboard item does not have a value of the indicated type.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

URL

The URL object of the first pasteboard item.

```
@property(nonatomic, copy) NSURL *URL
```

Discussion
The value stored in this property is an `NSURL` object. The associated array of representation types is `UIPasteboardTypeListURL` (page 431), which includes type `kUTTypeURL`. Setting this property replaces all current items in the pasteboard with the new item. If the first item has no value of the indicated type, `nil` is returned.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

URLs

An array of URL objects in all pasteboard items.

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

Discussion
The value stored in this property is an array of `NSURL` objects. The associated array of representation types is `UIPasteboardTypeListURL` (page 431), which includes type `kUTTypeURL`. Setting this property replaces all current items in the pasteboard with the new items. The returned array may have fewer objects than the number of pasteboard items; this happens if a pasteboard item does not have a value of the indicated type.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

Class Methods

generalPasteboard

Returns the general pasteboard, which is used for general copy-paste operations

```
+ (UIPasteboard *)generalPasteboard
```

Return Value
A shared system pasteboard object with the name of `UIPasteboardNameGeneral` (page 430).

Discussion
You may use the general pasteboard for copying and pasting text, images, URLs, colors, and other data within an application or between applications. The general pasteboard is persistent across device restarts and application uninstalls.

Availability
Available in iOS 3.0 and later.

See Also
`+ pasteboardWithName:create:` (page 421)
`+ pasteboardWithUniqueName` (page 422)
`+ removePasteboardWithName:` (page 423)

Declared In
`UIPasteboard.h`

pasteboardWithName:create:

Returns a pasteboard identified by name, optionally creating it if it doesn't exist.

```
+ (UIPasteboard *)pasteboardWithName:(NSString *)pasteboardName create:(BOOL)create
```

Parameters

pasteboardName

A string or string constant that identifies (or should identify) the pasteboard. Specify `nil` if you want `UIPasteboard` to create a pasteboard with a unique name.

create

A Boolean value that indicates whether the pasteboard should be created if it doesn't already exist. Specify `NO` for system pasteboards or if you want an existing application pasteboard.

Return Value
A pasteboard object that can be used for transferring data within and application or between applications.

Discussion
You call this method to obtain the `UIPasteboardNameFind` (page 430) pasteboard and to create custom application pasteboards. (You may also use it to obtain the general pasteboard, but `generalPasteboard` (page 421) exists for that purpose.) If you create a pasteboard for your application, the name should a unique string to prevent possible name collisions with other applications' pasteboards; for this, use of reverse DNS notation (for example, `com.mycompany.myapp.pboard`) is recommended. Application pasteboards returned by this method are not persistent, existing only until the application quits. To make them persistent, set the `persistent` (page 419) property to `YES`.

Availability
Available in iOS 3.0 and later.

See Also
`+ pasteboardWithUniqueName` (page 422)
`+ removePasteboardWithName:` (page 423)

Declared In
`UIPasteboard.h`

pasteboardWithUniqueName

Returns an application pasteboard identified by a unique system-generated name.

```
+ (UIPasteboard *)pasteboardWithUniqueName
```

Return Value
An application pasteboard object with a unique name.

Discussion
Obtain the value of the `name` (page 418) property to discover the name of the returned pasteboard. Application pasteboards returned by this method are not persistent, existing only until the application quits. To make them persistent, set the `persistent` (page 419) property to `YES`. Calling this method is equivalent to calling `pasteboardWithName:create:` (page 421) with the first parameter set to `nil` and the second set to `YES`.

Availability
Available in iOS 3.0 and later.

See Also
`+ generalPasteboard` (page 421)
`+ removePasteboardWithName:` (page 423)

Declared In
UIPasteboard.h

removePasteboardWithName:

Invalidates the designated application pasteboard.

```
+ (void)removePasteboardWithName:(NSString *)pasteboardName
```

Parameters

pasteboardName
The name of the pasteboard to be invalidated.

Discussion
Invalidation of an application pasteboard frees up all resources used by it. Once a pasteboard is invalidated, you cannot use the it; `UIPasteboard` ignores any calls to it. The method has no effect if called with the name of a system pasteboard.

Availability
Available in iOS 3.0 and later.

See Also
`+ pasteboardWithName:create:` (page 421)
`+ pasteboardWithUniqueName` (page 422)

Declared In
UIPasteboard.h

Instance Methods

addItems:

Appends pasteboard items to the current contents of the pasteboard.

```
- (void)addItems:(NSArray *)items
```

Parameters

items
An array of dictionaries. Each dictionary represents a pasteboard item, with the key being the representation type and the value the data object or property-list object associated with that type.

Availability
Available in iOS 3.0 and later.

See Also
`@property items` (page 418)

Declared In
UIPasteboard.h

containsPasteboardTypes:

Returns whether the pasteboard holds data of the specified representation type.

```
- (BOOL)containsPasteboardTypes:(NSArray *)pasteboardTypes
```

Parameters

pasteboardTypes

An array of strings. Each string should identify a representation of the pasteboard item that the pasteboard reader can handle. These string are frequently UTIs. See the class description for more information about pasteboard item types.

Return Value

`YES` if the pasteboard item holds data of the indicated representation type, otherwise `NO`.

Discussion

This method works on the first item in the pasteboard. If there are other items, it ignores them. You can use this method when enabling or disabling the Paste menu command.

Availability

Available in iOS 3.0 and later.

See Also

`- pasteboardTypes` (page 426)

Declared In

`UIPasteboard.h`

containsPasteboardTypes:inItemSet:

Returns whether the specified pasteboard items contain data of the given representation types.

```
- (BOOL)containsPasteboardTypes:(NSArray *)pasteboardTypes inItemSet:(NSIndexSet
    *)itemSet
```

Parameters

pasteboardTypes

An array of strings, with each string identifying a representation type. Typically you use UTIs as pasteboard types.

itemSet

An index set with each integer value identifying a pasteboard item positionally in the pasteboard. Pass in `nil` to request all pasteboard items.

Return Value

`YES` if the pasteboard items identified by *itemSet* have data corresponding to the representation types specified by *pasteboardTypes*; otherwise, returns `NO`.

Availability

Available in iOS 3.0 and later.

See Also

`@property numberOfItems` (page 419)

`- pasteboardTypesForItemSet:` (page 427)

`- itemSetWithPasteboardTypes:` (page 426)

Declared In
UIPasteboard.h

dataForPasteboardType:

Returns the data in the pasteboard for the given representation type.

```
- (NSData *)dataForPasteboardType:(NSString *)pasteboardType
```

Parameters

pasteboardType
A string identifying a representation type of a pasteboard item.

Return Value
A data object or nil if there is no data in the pasteboard of the given type.

Discussion
The returned object often holds raw (binary) data, such as image data. This method works on the first item in the pasteboard. If there are other items, it ignores them.

Availability
Available in iOS 3.0 and later.

See Also
- valueForPasteboardType: (page 428)
- setData:forPasteboardType: (page 427)

Declared In
UIPasteboard.h

dataForPasteboardType:inItemSet:

Returns the data objects in the indicated pasteboard items that have the given representation type.

```
- (NSArray *)dataForPasteboardType:(NSString *)pasteboardType inItemSet:(NSIndexSet
    *)itemSet
```

Parameters

pasteboardType
A string identifying a representation type. Typically this is a UTI.

itemSet
An index set with each integer value identifying a pasteboard item positionally in the pasteboard. Pass in nil to request all pasteboard items.

Return Value
An array of NSData objects or, if a requested pasteboard item has no data of the the type indicated by *pasteboardType*, a NSNull object.

Availability
Available in iOS 3.0 and later.

See Also
- valuesForPasteboardType:inItemSet: (page 429)

`@property items` (page 418)

Declared In
`UIPasteboard.h`

itemSetWithPasteboardTypes:

Returns an index set identifying pasteboard items having the specified representation types.

```
- (NSIndexSet *)itemSetWithPasteboardTypes:(NSArray *)pasteboardTypes
```

Parameters

pasteboardTypes

An array of strings, with each string identifying a representation type. Typically you use UTIs as pasteboard types.

Return Value
An index set with each integer positionally identifying a pasteboard item that has one of the representation types specified in *pasteboardTypes*.

Discussion
You can pass the index set returned in this method in a call to `dataForPasteboardType:inItemSet:` (page 425) or `valuesForPasteboardType:inItemSet:` (page 429) to get the data in the indicated pasteboard items.

Availability
Available in iOS 3.0 and later.

See Also
`@property numberOfItems` (page 419)
`- pasteboardTypesForItemSet:` (page 427)
`- containsPasteboardTypes:inItemSet:` (page 424)

Declared In
`UIPasteboard.h`

pasteboardTypes

Returns the types of the first item on the pasteboard.

```
- (NSArray *)pasteboardTypes
```

Return Value
An array of strings indicating the representation types of the first item on the pasteboard.

Discussion
A type is frequently, but not necessarily, a UTI (Uniform Type Identifier). It identifies a representation of the data on the pasteboard. For example, a pasteboard item could hold image data under `public.png` and `public.tiff` representations. Applications can define their own types for custom data such as `com.mycompany.myapp.mytype`; however, in this case, only those applications that know of the type could understand the data written to the pasteboard.

With this method, you can determine if the pasteboard holds data of a particular representation type by a line of code such as this:

```
BOOL pngOnPasteboard = [[pasteboard pasteboardTypes]
containsObject:@"public.png"];
```

Availability
Available in iOS 3.0 and later.

See Also
`- containsPasteboardTypes:` (page 424)

Declared In
`UIPasteboard.h`

pasteboardTypesForItemSet:

Returns an array of representation types for each specified pasteboard item.

```
- (NSArray *)pasteboardTypesForItemSet:(NSIndexSet *)itemSet
```

Parameters

itemSet
An index set with each integer value identifying a pasteboard item positionally in the pasteboard. Pass in `nil` to request all pasteboard items.

Return Value
An array of arrays, with each inner array holding the representation types for a particular pasteboard item.

Availability
Available in iOS 3.0 and later.

See Also
`@property numberOfItems` (page 419)
`- itemSetWithPasteboardTypes:` (page 426)
`- containsPasteboardTypes:inItemSet:` (page 424)

Declared In
`UIPasteboard.h`

setData:forPasteboardType:

Puts data in the pasteboard for the specified representation type.

```
- (void)setData:(NSData *)data forPasteboardType:(NSString *)pasteboardType
```

Parameters

data
The data object to be written to the pasteboard.

pasteboardType
A string identifying the representation type of the pasteboard item. This is typically a UTI.

Discussion
Use this method to put data on the pasteboard when the data is not a standard property-list object—that is, an object of the `NSString`, `NSArray`, `NSDictionary`, `NSDate`, `NSNumber`, or `NSURL` class. (For property-list objects, use the `setValue:forPasteboardType:` (page 428) method.) For example, you could archive a graph of model objects and pass the resulting `NSData` object to a related application via a pasteboard using a custom pasteboard type. This method writes data for the first item in the pasteboard. Calling this method replaces any items currently in the pasteboard.

Availability
Available in iOS 3.0 and later.

See Also
`- dataForPasteboardType:` (page 425)

Declared In
`UIPasteboard.h`

setValue:forPasteboardType:

Puts a property-list object in the pasteboard for the specified representation type.

```
- (void)setValue:(id)value forPasteboardType:(NSString *)pasteboardType
```

Parameters

value

The property-list object to be written to the pasteboard.

pasteboardType

A string identifying the representation type of the pasteboard item. If the type is a UTI, it must be compatible with the class of *value*; otherwise, nothing is written to the pasteboard.

Discussion
Use this method to put an object on the pasteboard that is a standard property-list object—that is an object of the `NSString`, `NSArray`, `NSDictionary`, `NSDate`, `NSNumber`, or `NSURL` class. (For all other data, such as raw binary data, use the `setData:forPasteboardType:` (page 427) method.) This method writes the object as the first item in the pasteboard. Calling this method replaces any items currently in the pasteboard.

Availability
Available in iOS 3.0 and later.

See Also
`- valueForPasteboardType:` (page 428)

Declared In
`UIPasteboard.h`

valueForPasteboardType:

Returns an object in the pasteboard for the given representation type.

```
- (id)valueForPasteboardType:(NSString *)pasteboardType
```

Parameters

pasteboardType

A string identifying a representation type of a pasteboard item.

Return Value

An object that is an instance of the appropriate class based on *pasteboardType*, a property-list object, or a `NSData` object containing "raw" data.

Discussion

This method attempts to return an object that is of a class type appropriate to the representation type, which typically is a UTI. For example, if the representation type is `kUTTypePlainText` (`public.plain-text`), the method returns an `NSString` object. If the method cannot determine the class type from the representation type, it returns the object as a generic property-list object. Property-list objects include `NSString`, `NSArray`, `NSDictionary`, `NSDate`, or `NSNumber` objects, with `NSURL` objects also as a possibility. If the method cannot decode the value as a property-list object, it returns the pasteboard item as an `NSData` object. This method works on the first item in the pasteboard. If there are other items, it ignores them.

Availability

Available in iOS 3.0 and later.

See Also

- `dataForPasteboardType:` (page 425)
- `setValue:forPasteboardType:` (page 428)

Declared In

`UIPasteboard.h`

valuesForPasteboardType:inItemSet:

Returns the property-list objects in the indicated pasteboard items that have the given representation type.

```
- (NSArray *)valuesForPasteboardType:(NSString *)pasteboardType inItemSet:(NSIndexSet
    *)itemSet
```

Parameters

pasteboardType

A string identifying a representation type. Typically this is a UTI.

itemSet

An index set with each integer value identifying a pasteboard item positionally in the pasteboard. Pass in `nil` to request all pasteboard items.

Return Value

An array of `NSData` objects or, if a requested pasteboard item has no data of the the type indicated by *pasteboardType*, a `NSNull` object.

Discussion

Returned objects are of one of the following classes: `NSString`, `NSArray`, `NSDictionary`, `NSDate`, `NSNumber`, or `NSURL`.

Availability

Available in iOS 3.0 and later.

See Also

- `dataForPasteboardType:inItemSet:` (page 425)

`@property items` (page 418)

Declared In
`UIPasteboard.h`

Constants

Pasteboard Names

Names identifying the system pasteboards.

```
UIKIT_EXTERN NSString *const UIPasteboardNameGeneral;
UIKIT_EXTERN NSString *const UIPasteboardNameFind;
```

Constants

`UIPasteboardNameGeneral`

The name identifying the General pasteboard, which is used for general copy-cut-paste operations.

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

`UIPasteboardNameFind`

The name identifying the Find pasteboard, which is used in search operations. In such operations, the most recent search string in the search bar is put in the Find pasteboard.

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

Discussion
You can access both system pasteboards by calling the class method `pasteboardWithName:create:` (page 421), specifying one of these constants as the first argument. You may also access the general pasteboard by calling the `generalPasteboard` (page 421) class method. Both system pasteboards are persistent across device restarts, application uninstalls, and restores.

Declared In
`UIPasteboard.h`

Data Type Extensions

Pasteboard-item representation types for a given object value.

```
UIKIT_EXTERN NSArray *UIPasteboardTypeListString;
UIKIT_EXTERN NSArray *UIPasteboardTypeListURL;
UIKIT_EXTERN NSArray *UIPasteboardTypeListImage;
UIKIT_EXTERN NSArray *UIPasteboardTypeListColor;
```

Constants

`UIPasteboardTypeListString`

An array of pasteboard-item representation types for strings, including `kUTTypeUTF8PlainText`. Related properties are `string` (page 419) and `strings` (page 420).

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

`UIPasteboardTypeListURL`
An array of pasteboard-item representation types for URLs, including `kUTTypeURL`. Related properties are `URL` (page 420) and `URLs` (page 421).

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

`UIPasteboardTypeListImage`
An array of pasteboard-item representation types for images, including `kUTTypePNG` and `kUTTypeJPEG`. Related properties are `image` (page 417) and `images` (page 418).

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

`UIPasteboardTypeListColor`
An array of pasteboard-item representation types for colors. Related properties are `color` (page 417) and `colors` (page 417).

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

Declared In
`UIPasteboard.h`

UserInfo Dictionary Keys

You use the following keys to access the representation types of pasteboard items that have been added to or removed from a pasteboard.

```
UIKIT_EXTERN NSString *const UIPasteboardChangedTypesAddedKey;
UIKIT_EXTERN NSString *const UIPasteboardChangedTypesRemovedKey;
```

Constants
`UIPasteboardChangedTypesAddedKey`
With the notification named `UIPasteboardChangedNotification` (page 432), use this key to access the added representation types. These types are stored as an array in the notification's `userInfo` dictionary.

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

`UIPasteboardChangedTypesRemovedKey`
With the notification named `UIPasteboardChangedNotification` (page 432), use this key to access the removed representation types. These types are stored as an array in the notification's `userInfo` dictionary.

Available in iOS 3.0 and later.

Declared in `UIPasteboard.h`.

Declared In
`UIPasteboard.h`

Notifications

UIPasteboardChangedNotification

Posted by a pasteboard object when its contents change. This happens at the same time the pasteboard's change count (`changeCount` (page 416) property) is incremented. Changes include the addition, removal, and modification of pasteboard items. The `userInfo` dictionary may contain the representation types of pasteboard items that have been added to or removed from the pasteboard. See "UserInfo Dictionary Keys" (page 431) for the keys used to access these representation types. If pasteboard items have been modified but not added or removed, the `userInfo` dictionary is `nil`.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

UIPasteboardRemovedNotification

Posted by a pasteboard object just before an application removes it. The removal class method is `removePasteboardWithName:` (page 423). There is no `userInfo` dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`UIPasteboard.h`

UIPickerView Class Reference

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

Overview

The `UIPickerView` class implements objects, called picker views, that use a spinning-wheel or slot-machine metaphor to show one or more sets of values. Users select values by rotating the wheels so that the desired row of values aligns 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.

The user interface provided by a picker view consists of components and rows. 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 numbers of components and rows. 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 the number of components and the number of rows in each component.

You can dynamically change the rows of a component by calling the `reloadComponent:` (page 437) method, or dynamically change the rows of all components by calling the `reloadAllComponents` (page 437) method. When you call either of these methods, the picker view asks the delegate for new component and row data, and asks the data source for new component and row counts. Reload a picker view when a selected value in one component should change the set of values in another component. For example, changing a row value from February to March in one component should change a related component representing the days of the month.

Tasks

Getting the Dimensions of the View Picker

`numberOfComponents` (page 435) *property*
Gets the number of components for the picker view. (read-only)

- `numberOfRowsInComponent:` (page 436)
Returns the number of rows for a component.

- `rowSizeForComponent:` (page 438)
Returns the size of a row for a component.

Reloading the View Picker

- `reloadAllComponents` (page 437)
Reloads all components of the picker view.

- `reloadComponent:` (page 437)
Reloads a particular component of the picker view.

Selecting Rows in the View Picker

- `selectRow:inComponent:animated:` (page 438)
Selects a row in a specified component of the picker view.

- `selectedRowInComponent:` (page 438)
Returns the index of the selected row in a given component.

Returning the View for a Row and Component

- `viewForRow:forComponent:` (page 439)
Returns the view used by the picker view for a given row and component.

Specifying the Delegate

`delegate` (page 435) *property*
The delegate for the picker view.

Specifying the Data Source

`dataSource` (page 435) *property*
The data source for the picker view.

Managing the Appearance of the Picker View

`showsSelectionIndicator` (page 436) *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 Programming Language*.

dataSource

The data source for the picker view.

```
@property(nonatomic, assign) id<UIPickerViewDataSource> dataSource
```

Discussion
The data source must adopt the `UIPickerViewDataSource` protocol and implement the required methods to return the number of components and the number of rows in each component.

Availability
Available in iOS 2.0 and later.

Declared In
`UIPickerView.h`

delegate

The delegate for the picker view.

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

Discussion
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.

Availability
Available in iOS 2.0 and later.

Declared In
`UIPickerView.h`

numberOfComponents

Gets the number of components for the picker view. (read-only)

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

Discussion
A `UIPickerView` object fetches the value of this property from the data source and and caches it. The default value is zero.

Availability
Available in iOS 2.0 and later.

See Also
- `numberOfRowsInComponent:` (page 436)
- `rowSizeForComponent:` (page 438)

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 iOS 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
A zero-indexed number identifying a component.

Return Value
The number of rows in the given component.

Discussion
A picker view fetches the value of this property from the data source and and caches it. The default value is zero.

Availability
Available in iOS 2.0 and later.

See Also
`@property numberOfComponents` (page 435)
`- rowSizeForComponent:` (page 438)

Declared In
`UIPickerView.h`

reloadAllComponents

Reloads all components of the picker view.

```
- (void)reloadAllComponents
```

Discussion
Calling this method causes the picker view to query the delegate for new data for all components.

Availability
Available in iOS 2.0 and later.

See Also
`- reloadComponent:` (page 437)

Declared In
`UIPickerView.h`

reloadComponent:

Reloads a particular component of the picker view.

```
- (void)reloadComponent:(NSInteger)component
```

Parameters
component
A zero-indexed number identifying a component of the picker view.

Discussion
Calling this method causes the picker view to query the delegate for new data for the given component.

Availability
Available in iOS 2.0 and later.

See Also
`- reloadAllComponents` (page 437)

Declared In
`UIPickerView.h`

rowSizeForComponent:

Returns the size of a row for a component.

```
- (CGSize)rowSizeForComponent:(NSInteger)component
```

Parameters

component

A zero-indexed 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

A picker view fetches the value of this property by calling the `pickerView:widthForComponent:` (page 882) and `pickerView:rowHeightForComponent:` (page 880) delegate methods, and caches it. The default value is (0, 0).

Availability

Available in iOS 2.0 and later.

See Also

`@property numberOfComponents` (page 435)

`- numberOfRowsInComponent:` (page 436)

Declared In

`UIPickerView.h`

selectedRowInComponent:

Returns the index of the selected row in a given component.

```
- (NSInteger)selectedRowInComponent:(NSInteger)component
```

Parameters

component

A zero-indexed number identifying a component of the picker view.

Return Value

A zero-indexed number identifying the selected row, or `-1` if no row is selected.

Availability

Available in iOS 2.0 and later.

See Also

`- selectRow:inComponent:animated:` (page 438)

Declared In

`UIPickerView.h`

selectRow:inComponent:animated:

Selects a row in a specified component of the picker view.

```
- (void)selectRow:(NSInteger)row inComponent:(NSInteger)component
    animated:(BOOL)animated
```

Parameters

row

A zero-indexed number identifying a row of *component*.

component

A zero-indexed 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 iOS 2.0 and later.

See Also
`- selectedRowInComponent:` (page 438)

Declared In
`UIPickerView.h`

viewForRow:forComponent:

Returns the view used by the picker view for a given row and component.

```
- (UIView *)viewForRow:(NSInteger)row forComponent:(NSInteger)component
```

Parameters

row

The zero-indexed number of a row of the picker view.

component

The zero-indexed number of a component of the picker view.

Return Value
The view provided by the delegate in the `pickerView:viewForRow:forComponent:reusingView:` (page 881) 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 iOS 2.0 and later.

Declared In
`UIPickerView.h`

UIPinchGestureRecognizer Class Reference

Inherits from	UIGestureRecognizer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIPinchGestureRecognizer.h
Companion guide	Event Handling Guide for iOS

Overview

`UIPinchGestureRecognizer` is a concrete subclass of `UIGestureRecognizer` that looks for pinching gestures involving two touches. When the user moves the two fingers toward each other, the conventional meaning is zoom-out; when the user moves the two fingers away from each other, the conventional meaning is zoom-in.

Pinching is a continuous gesture. The gesture begins (`UIGestureRecognizerStateBegan` (page 296)) when the two touches have moved enough to be considered a pinch gesture. The gesture changes (`UIGestureRecognizerStateChanged` (page 296)) when a finger moves (with both fingers remaining pressed). The gesture ends (`UIGestureRecognizerStateEnded` (page 296)) when both fingers lift from the view.

Tasks

Interpreting the Pinching Gesture

`scale` (page 442) *property*
The scale factor relative to the points of the two touches in screen coordinates.

`velocity` (page 442) *property*
The velocity of the pinch in scale factor per second. (read-only)

Properties

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

scale

The scale factor relative to the points of the two touches in screen coordinates.

```
@property(nonatomic) CGFloat scale
```

Discussion
You may set the scale factor, but doing so resets the velocity.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPinchGestureRecognizer.h`

velocity

The velocity of the pinch in scale factor per second. (read-only)

```
@property(nonatomic, readonly) CGFloat velocity
```

Availability
Available in iOS 3.2 and later.

Declared In
`UIPinchGestureRecognizer.h`

UIPopoverController Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIPopoverController.h
Companion guide	iPad Programming Guide
Related sample code	MultipleDetailViews ToolbarSearch

Overview

The `UIPopoverController` class is used to manage the presentation of content in a popover. You use popovers to present information temporarily but in a way that does not take over the entire screen like a modal view does. The popover content is layered on top of your existing content in a special type of window. The popover remains visible until the user taps outside of the popover window or you explicitly dismiss it. Popover controllers are for use exclusively on iPad devices. Attempting to create one on other devices results in an exception.

To display a popover, create an instance of this class and present it using one of the appropriate methods. When initializing an instance of this class, you must provide the view controller that provides the content for the popover. Popovers normally derive their size from the view controller they present. However, you can change the size of the popover by modifying the value in the `popoverContentSize` (page 446) property or by calling the `setPopoverContentSize:animated:` (page 450) method. The latter approach is particularly effective if you need to animate changes to the popover's size. The size you specify is just the preferred size for the popover's view. The actual size may be altered to ensure that the popover fits on the screen and does not collide with the keyboard.

When displayed, taps outside of the popover window cause the popover to be dismissed automatically. To allow the user to interact with the specified views and not dismiss the popover, you can assign one or more views to the `passthroughViews` (page 446) property. Taps inside the popover window do not automatically cause the popover to be dismissed. Your view and view controller code must handle actions and events inside the popover explicitly and call the `dismissPopoverAnimated:` (page 447) method as needed.

If the user rotates the device while a popover is visible, the popover controller hides the popover and then shows it again at the end of the rotation. The popover controller attempts to position the popover appropriately for you but you may have to present it again or hide it altogether in some cases. For example, when displayed from a bar button item, the popover controller automatically adjusts the position (and

potentially the size) of the popover to account for changes to the position of the bar button item. However, if you remove the bar button item during the rotation, or if you presented the popover from a target rectangle in a view, the popover controller does not attempt to reposition the popover. In those cases, you must manually hide the popover or present it again from an appropriate new position. You can do this in the `didRotateFromInterfaceOrientation:` (page 764) method of the view controller that you used to present the popover.

You can assign a delegate to the popover to manage interactions with the popover and receive notifications about its dismissal. For information about the methods of the delegate object, see *UIPopoverControllerDelegate Protocol Reference*.

Tasks

Initializing the Popover

- `initWithContentViewController:` (page 448)
  Returns an initialized popover controller object.

Configuring the Popover Attributes

`contentViewController` (page 445) *property*
The view controller responsible for the content portion of the popover.

- `setContentViewController:animated:` (page 449)
  Sets the view controller responsible for the content portion of the popover.

`popoverContentSize` (page 446) *property*
The size of the popover's content view.

- `setPopoverContentSize:animated:` (page 450)
  Changes the size of the popover's content view.

`passthroughViews` (page 446) *property*
An array of views that the user can interact with while the popover is visible.

`delegate` (page 445) *property*
The delegate you want to receive popover controller messages.

Getting the Popover Attributes

`popoverVisible` (page 447) *property*
A Boolean value indicating whether the popover is currently visible. (read-only)

`popoverArrowDirection` (page 446) *property*
The direction of the popover's arrow. (read-only)

Presenting and Dismissing the Popover

- `presentPopoverFromRect:inView:permittedArrowDirections:animated:` (page 449)
  Displays the popover and anchors it to the specified location in the view.
- `presentPopoverFromBarButtonItem:permittedArrowDirections:animated:` (page 448)
  Displays the popover and anchors it to the specified bar button item.
- `dismissPopoverAnimated:` (page 447)
  Dismisses the popover programmatically.

Properties

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

contentViewController

The view controller responsible for the content portion of the popover.

```
@property (nonatomic, retain) UIViewController *contentViewController
```

Discussion
This property is initially set to the view controller passed to the `initWithContentViewController:` (page 448) method. You can change the value of this property later to reflect a new set of content. Changing the value of this property swaps the new view controller in for the old one immediately and does not trigger an animation. If you want to animate the change, use the `setContentViewController:animated:` method instead.

Availability
Available in iOS 3.2 and later.

See Also
- `setContentViewController:animated:` (page 449)

Declared In
`UIPopoverController.h`

delegate

The delegate you want to receive popover controller messages.

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

Discussion
The popover controller uses its delegate to determine whether it should dismiss the popover and provides a notification when such an event occurs. For more information about the methods you can implement in your delegate, see *UIPopoverControllerDelegate Protocol Reference*.

Availability
Available in iOS 3.2 and later.

Related Sample Code
ToolbarSearch

Declared In
`UIPopoverController.h`

passthroughViews

An array of views that the user can interact with while the popover is visible.

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

Discussion
When a popover is active, interactions with other views are normally disabled until the popover is dismissed. Assigning an array of views to this property allows taps outside of the popover to be handled by the corresponding views.

Availability
Available in iOS 3.2 and later.

Related Sample Code
ToolbarSearch

Declared In
`UIPopoverController.h`

popoverArrowDirection

The direction of the popover's arrow. (read-only)

```
@property (nonatomic, readonly) UIPopoverArrowDirection popoverArrowDirection
```

Discussion
The default value of this property is `UIPopoverArrowDirectionUnknown` (page 451). When you present the popover, the value changes to reflect the actual direction of the arrow being used by the popover. When the popover is subsequently dismissed, the value of this property returns to `UIPopoverArrowDirectionUnknown`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

popoverContentSize

The size of the popover's content view.

```
@property (nonatomic) CGSize popoverContentSize
```

Discussion
This property represents the size of the content view that is managed by the view controller in the `contentViewController` (page 445) property. The initial value of this property is set to value in the view controller's `contentSizeForViewInPopover` (page 754) property. Changing the value of this property overrides the default value of the current view controller. The overridden value persists until you assign a new content view controller to the receiver. Thus, if you want to keep your overridden value, you must reassign it after changing the content view controller.

When changing the value of this property, the width value you specify must be at least 320 points and no more than 600 points. There are no restrictions on the height value. However, both the width and height values you specify may be adjusted to ensure the popup fits on screen and is not covered by the keyboard. If you change the value of this property while the popover is visible, the size change is animated.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

popoverVisible

A Boolean value indicating whether the popover is currently visible. (read-only)

```
@property (nonatomic, readonly, getter=isPopoverVisible) BOOL popoverVisible
```

Discussion
You must present the popover to make it visible.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

Instance Methods

dismissPopoverAnimated:

Dismisses the popover programmatically.

```
- (void)dismissPopoverAnimated:(BOOL)animated
```

Parameters
animated
Specify `YES` to animate the dismissal of the popover or `NO` to dismiss it immediately.

Discussion
You can use this method to dismiss the popover programmatically in response to taps inside the popover window. Taps outside of the popover's contents automatically dismiss the popover.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

initWithContentViewController:

Returns an initialized popover controller object.

```
- (id)initWithContentViewController:(UIViewController *)viewController
```

Parameters

viewController
The view controller for managing the popover's content. This parameter must not be `nil`.

Return Value
An initialized popover controller object.

Discussion
When initializing a popover controller, you must specify the view controller object whose content is to be displayed in the popover. You can change this view controller later by modifying the `contentViewController` (page 445) property.

Availability
Available in iOS 3.2 and later.

Related Sample Code
ToolbarSearch

Declared In
`UIPopoverController.h`

presentPopoverFromBarButtonItem:permittedArrowDirections:animated:

Displays the popover and anchors it to the specified bar button item.

```
- (void)presentPopoverFromBarButtonItem:(UIBarButtonItem *)item
    permittedArrowDirections:(UIPopoverArrowDirection)arrowDirections
    animated:(BOOL)animated
```

Parameters

item
The bar button item on which to anchor the popover.

arrowDirections
The arrow directions the popover is permitted to use. You can use this value to force the popover to be positioned on a specific side of the bar button item. However, it is generally better to specify `UIPopoverArrowDirectionAny` (page 451) and let the popover decide the best placement. You must not specify `UIPopoverArrowDirectionUnknown` (page 451) for this parameter.

animated
Specify `YES` to animate the presentation of the popover or `NO` to display it immediately.

Discussion
When presenting the popover, this method adds the toolbar that owns the button to the popover's list of passthrough views. Thus, taps in the toolbar result in the action methods of the corresponding toolbar items being called. If you want the popover to be dismissed when a different toolbar item is tapped, you must implement that behavior in your action handler methods.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

presentPopoverFromRect:inView:permittedArrowDirections:animated:

Displays the popover and anchors it to the specified location in the view.

```
- (void)presentPopoverFromRect:(CGRect)rect inView:(UIView *)view
    permittedArrowDirections:(UIPopoverArrowDirection)arrowDirections
    animated:(BOOL)animated
```

Parameters

rect
The rectangle in view at which to anchor the popover window.

view
The view containing the anchor rectangle for the popover.

arrowDirections
The arrow directions the popover is permitted to use. You can use this value to force the popover to be positioned on a specific side of the rectangle. However, it is generally better to specify `UIPopoverArrowDirectionAny` (page 451) and let the popover decide the best placement. You must not specify `UIPopoverArrowDirectionUnknown` (page 451) for this parameter.

animated
Specify `YES` to animate the presentation of the popover or `NO` to display it immediately.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

setContentViewController:animated:

Sets the view controller responsible for the content portion of the popover.

```
- (void)setContentViewController:(UIViewController *)viewController
    animated:(BOOL)animated
```

Parameters

viewController
The new view controller whose content should be displayed by the popover.

animated

Specify YES if the change of view controllers should be animated or NO if the change should occur immediately.

Availability
Available in iOS 3.2 and later.

Declared In
UIPopoverController.h

setPopoverContentSize:animated:

Changes the size of the popover's content view.

```
- (void)setPopoverContentSize:(CGSize)size animated:(BOOL)animated
```

Parameters

size

The new size to apply to the content view.

animated

Specify YES if you want the change in size to be animated or NO if you want the change to appear immediately.

Discussion
When changing the size of the popover's content, the width value you specify must be at least 320 points and no more than 600 points. There are no restrictions on the height value. However, both the width and height values you specify may be adjusted to ensure the popup fits on screen and is not covered by the keyboard.

Availability
Available in iOS 3.2 and later.

Declared In
UIPopoverController.h

Constants

UIPopoverArrowDirection

Constants for specifying the direction of the popover arrow.

```
enum {
    UIPopoverArrowDirectionUp = 1UL << 0,
    UIPopoverArrowDirectionDown = 1UL << 1,
    UIPopoverArrowDirectionLeft = 1UL << 2,
    UIPopoverArrowDirectionRight = 1UL << 3,
    UIPopoverArrowDirectionAny = UIPopoverArrowDirectionUp |
UIPopoverArrowDirectionDown |
                                 UIPopoverArrowDirectionLeft |
UIPopoverArrowDirectionRight,
    UIPopoverArrowDirectionUnknown = NSUIntegerMax
};
typedef NSUInteger UIPopoverArrowDirection;
```

Constants

`UIPopoverArrowDirectionUp`
An arrow that points upward.
Available in iOS 3.2 and later.
Declared in `UIPopoverController.h`.

`UIPopoverArrowDirectionDown`
An arrow that points downward.
Available in iOS 3.2 and later.
Declared in `UIPopoverController.h`.

`UIPopoverArrowDirectionLeft`
An arrow that points toward the left.
Available in iOS 3.2 and later.
Declared in `UIPopoverController.h`.

`UIPopoverArrowDirectionRight`
An arrow that points toward the right.
Available in iOS 3.2 and later.
Declared in `UIPopoverController.h`.

`UIPopoverArrowDirectionAny`
An arrow that points in any direction.
Available in iOS 3.2 and later.
Declared in `UIPopoverController.h`.

`UIPopoverArrowDirectionUnknown`
The status of the arrow is currently unknown.
Available in iOS 3.2 and later.
Declared in `UIPopoverController.h`.

UIProgressView Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 455)
Initializes and returns an progress-view object.

Managing the Progress Bar

`progress` (page 454) *property*
The current progress shown by the receiver.

Configuring the Bar Style

`progressViewStyle` (page 454) *property*
The current graphical style of the receiver.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 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 455). For more on these constants, see `UIProgressViewStyle` (page 455).

Availability
Available in iOS 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 455) 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 454) property.

Availability

Available in iOS 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 iOS 2.0 and later.

Declared in `UIProgressView.h`.

`UIProgressViewStyleBar`

The style of progress view that is used in a toolbar.

Available in iOS 2.0 and later.

Declared in `UIProgressView.h`.

Discussion
You can set and retrieve the current style of progress view through the `progressViewStyle` (page 454) property.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	UIResponder.h
Companion guide	Event Handling Guide for iOS

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 or, simply, responders.

There are two general kinds of events: touch events and motion events. The primary event-handling methods for touches are `touchesBegan:withEvent:` (page 467), `touchesMoved:withEvent:` (page 468), `touchesEnded:withEvent:` (page 468), and `touchesCancelled:withEvent:` (page 467). The parameters of these methods associate touches with their events—specifically 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.

iOS 3.0 introduced system capabilities for generating motion events, specifically the motion of shaking the device. The event-handling methods for these kinds of events are `motionBegan:withEvent:` (page 463), `motionEnded:withEvent:` (page 464), and `motionCancelled:withEvent:` (page 464). Additionally for iOS 3.0, the `canPerformAction:withSender:` (page 461) method allows responders to validate commands in the user interface while the `undoManager` (page 460) property returns the nearest `NSUndoManager` object in the responder chain.

In iOS 4.0, `UIResponder` added the `remoteControlReceivedWithEvent:` (page 466) method for handling remote-control events.

Tasks

Managing the Responder Chain

- `nextResponder` (page 465)
  Returns the receiver's next responder, or `nil` if it has none.
- `isFirstResponder` (page 463)
  Returns a Boolean value indicating whether the receiver is the first responder.
- `canBecomeFirstResponder` (page 461)
  Returns a Boolean value indicating whether the receiver can become first responder.
- `becomeFirstResponder` (page 460)
  Notifies the receiver that it is about to become first responder in its window.
- `canResignFirstResponder` (page 462)
  Returns a Boolean value indicating whether the receiver is willing to relinquish first-responder status.
- `resignFirstResponder` (page 466)
  Notifies the receiver that it has been asked to relinquish its status as first responder in its window.

Managing Input Views

`inputView` (page 460) *property*
The custom input view to display when the object becomes the first responder. (read-only)

`inputAccessoryView` (page 459) *property*
The custom accessory view to display when the object becomes the first responder. (read-only)

- `reloadInputViews` (page 465)
  Updates the custom input and accessory views when the object is the first responder.

Responding to Touch Events

- `touchesBegan:withEvent:` (page 467)
  Tells the receiver when one or more fingers touch down in a view or window.
- `touchesMoved:withEvent:` (page 468)
  Tells the receiver when one or more fingers associated with an event move within a view or window.
- `touchesEnded:withEvent:` (page 468)
  Tells the receiver when one or more fingers are raised from a view or window.
- `touchesCancelled:withEvent:` (page 467)
  Sent to the receiver when a system event (such as a low-memory warning) cancels a touch event.

Responding to Motion Events

- `motionBegan:withEvent:` (page 463)
  Tells the receiver that a motion event has begun.

- `motionEnded:withEvent:` (page 464)
  Tells the receiver that a motion event has ended.
- `motionCancelled:withEvent:` (page 464)
  Tells the receiver that a motion event has been cancelled.

Responding to Remote-Control Events

- `remoteControlReceivedWithEvent:` (page 466)
  Sent to the receiver when a remote-control event is received.

Getting the Undo Manager

`undoManager` (page 460) *property*
Returns the nearest shared undo manager in the responder chain.

Validating Commands

- `canPerformAction:withSender:` (page 461)
  Requests the receiving responder to enable or disable the specified command in the user interface.

Properties

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

inputAccessoryView

The custom accessory view to display when the object becomes the first responder. (read-only)

```
@property (readonly, retain) UIView *inputAccessoryView
```

Discussion
The default value of this property is `nil`. Subclasses that want to attach custom controls to either a system-supplied input view (such as the keyboard) or a custom input view (one you provide in the `inputView` (page 460) property) should redeclare this property as readwrite and use it to manage their custom accessory view. When the receiver subsequently becomes the first responder, the responder infrastructure attaches the view to the appropriate input view before displaying it.

This property is typically used to attach an accessory view to the system-supplied keyboard that is presented for `UITextField` and `UITextView` objects.

Availability
Available in iOS 3.2 and later.

Declared In
`UIResponder.h`

inputView

The custom input view to display when the object becomes the first responder. (read-only)

```
@property (readonly, retain) UIView *inputView
```

Discussion
The value of this property is `nil`. Responder objects that require a custom view to gather input from the user should redeclare this property as readwrite and use it to manage their custom input view. When the receiver subsequently becomes the first responder, the responder infrastructure presents the specified input view automatically. Similarly, when the view resigns its first responder status, the responder infrastructure automatically dismisses the specified view.

This property is typically used to replace the system-supplied keyboard that is presented for `UITextField` and `UITextView` objects.

Availability
Available in iOS 3.2 and later.

Declared In
`UIResponder.h`

undoManager

Returns the nearest shared undo manager in the responder chain.

```
@property(readonly) NSUndoManager *undoManager
```

Discussion
By default, every window of an application has an undo manager: a shared object for managing undo and redo operations. However, the class of any object in the responder chain can have their own custom undo manager. (For example, instances of `UITextField` have their own undo manager that is cleared when the text field resigns first-responder status.) When you request an undo manager, the request goes up the responder chain and the `UIWindow`object returns a usable instance.

You may add undo managers to your view controllers to perform undo and redo operations local to the managed view.

Availability
Available in iOS 3.0 and later.

Declared In
`UIResponder.h`

Instance Methods

becomeFirstResponder

Notifies the receiver that it is about to become first responder in its window.

```
- (BOOL)becomeFirstResponder
```

Return Value
`YES` if the receiver accepts first-responder status or `NO` if it refuses this status. The default implementation returns `YES`, accepting first responder status.

Discussion
Subclasses can override this method to update state or perform some action such as highlighting the selection.

A responder object only becomes the first responder if the current responder can resign first-responder status (`canResignFirstResponder` (page 462)) and the new responder can become first responder.

You may call this method to make a responder object such as a view the first responder. However, you should only call it on that view if it is part of a view hierarchy. If the view's `window` property holds a `UIWindow` object, it has been installed in a view hierarchy; if it returns `nil`, the view is detached from any hierarchy.

Availability
Available in iOS 2.0 and later.

See Also
- `isFirstResponder` (page 463)
- `canBecomeFirstResponder` (page 461)

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
Returns `NO` by default. If a responder object returns `YES` from this method, it becomes the first responder and can 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 iOS 2.0 and later.

See Also
- `becomeFirstResponder` (page 460)

Declared In
`UIResponder.h`

canPerformAction:withSender:

Requests the receiving responder to enable or disable the specified command in the user interface.

```
- (BOOL)canPerformAction:(SEL)action withSender:(id)sender
```

Parameters

action

A selector that identifies a method associated with a command. For the editing menu, this is one of the editing methods declared by the `UIResponderStandardEditActions` informal protocol (or example, `copy:`).

sender

The object calling this method. For the editing menu commands, this is the shared `UIApplication` object. Depending on the context, you can query the sender for information to help you determine whether a command should be enabled.

Return Value

`YES` if the the command identified by *action* should be enabled or `NO` if it should be disabled. Returning `YES` means that your class can handle the command in the current context.

Discussion

This default implementation of this method returns `YES` if the responder class implements the requested action and calls the next responder if it does not. Subclasses may override this method to enable menu commands based on the current state; for example, you would enable the Copy command if there is a selection or disable the Paste command if the pasteboard did not contain data with the correct pasteboard representation type. If no responder in the responder chain returns `YES`, the menu command is disabled. Note that if your class returns `NO` for a command, another responder further up the responder chain may still return `YES`, enabling the command.

This method might be called more than once for the same action but with a different sender each time. You should be prepared for any kind of sender including `nil`.

For information on the editing menu, see the description of the `UIMenuController` class.

Availability

Available in iOS 3.0 and later.

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

Returns `YES` by default. As an example, a text field in the middle of editing might want to implement this method to return `NO` to keep itself active during editing.

Availability

Available in iOS 2.0 and later.

See Also

`- resignFirstResponder` (page 466)

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 iOS 2.0 and later.

See Also
- becomeFirstResponder (page 460)
- resignFirstResponder (page 466)

Declared In
UIResponder.h

motionBegan:withEvent:

Tells the receiver that a motion event has begun.

```
- (void)motionBegan:(UIEventSubtype)motion withEvent:(UIEvent *)event
```

Parameters

motion
An event-subtype constant indicating the kind of motion. A common motion is shaking, which is indicated by UIEventSubtypeMotionShake (page 267).

event
An object representing the event associated with the motion.

Discussion
iOS informs the first responder only when a motion event starts and when it ends; for example, it doesn't report individual shakes. The receiving object must be the first responder to receive motion events.

The default implementation of this method does nothing. However immediate UIKit subclasses of UIResponder, particularly UIView, forward the message up the responder chain.

Availability
Available in iOS 3.0 and later.

See Also
- motionEnded:withEvent: (page 464)
- motionCancelled:withEvent: (page 464)

Declared In
UIResponder.h

motionCancelled:withEvent:

Tells the receiver that a motion event has been cancelled.

```
- (void)motionCancelled:(UIEventSubtype)motion withEvent:(UIEvent *)event
```

Parameters

motion

An event-subtype constant indicating the kind of motion associated with *event*. A common motion is shaking, which is indicated by `UIEventSubtypeMotionShake` (page 267).

event

An object representing the event associated with the motion.

Discussion

This method is invoked when the Cocoa Touch framework receives an interruption requiring cancellation of the motion event. This interruption is something that might cause the application to be no longer active or the view to be removed from the window. The method can also be invoked if the shaking goes on too long. All responders that handle motion events should implement this method; in it they should clean up any state information that was established in the `motionBegan:withEvent:` (page 463) implementation.

The default implementation of this method does nothing. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain.

Availability

Available in iOS 3.0 and later.

See Also

- `motionBegan:withEvent:` (page 463)
- `motionEnded:withEvent:` (page 464)

Declared In

`UIResponder.h`

motionEnded:withEvent:

Tells the receiver that a motion event has ended.

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event
```

Parameters

motion

An event-subtype constant indicating the kind of motion. A common motion is shaking, which is indicated by `UIEventSubtypeMotionShake` (page 267).

event

An object representing the event associated with the motion.

Discussion

iOS informs the responder only when a motion event starts and when it ends; for example, it doesn't report individual shakes.

The default implementation of this method does nothing. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain.

Availability
Available in iOS 3.0 and later.

See Also
- `motionBegan:withEvent:` (page 463)
- `motionCancelled:withEvent:` (page 464)

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 `UIViewController` object that manages it (if it has one) or its superview (if it doesn't); `UIViewController` implements the method by returning its view's superview; `UIWindow` returns the application object, and `UIApplication` returns `nil`.

Availability
Available in iOS 2.0 and later.

See Also
- `isFirstResponder` (page 463)

Declared In
`UIResponder.h`

reloadInputViews

Updates the custom input and accessory views when the object is the first responder.

```
- (void)reloadInputViews
```

Discussion
You can use this method to refresh the custom input view or input accessory view associated with the current object when it is the first responder. The views are replaced immediately—that is, without animating them into place. If the current object is not the first responder, this method has no effect.

Availability
Available in iOS 3.2 and later.

Declared In
`UIResponder.h`

remoteControlReceivedWithEvent:

Sent to the receiver when a remote-control event is received.

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

Parameters

event

An event object encapsulating a remote-control command. Remote-control events have a type of `UIEventTypeRemoteControl` (page 266).

Discussion

Remote-control events originate as commands from external accessories, including headsets. An application responds to these commands by controlling audio or video media presented to the user. The receiving responder object should examine the `subtype` (page 263) of *event* to determine the intended command—for example, play (UIEventSubtypeRemoteControlPlay)—and then proceed accordingly.

To allow delivery of remote-control events, you must call the `beginReceivingRemoteControlEvents` (page 117) method of `UIApplication`; to turn off delivery of remote-control events, call `endReceivingRemoteControlEvents` (page 120).

Availability

Available in iOS 4.0 and later.

Declared In

`UIResponder.h`

resignFirstResponder

Notifies the receiver that it has 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 iOS 2.0 and later.

See Also

- `isFirstResponder` (page 463)
- `canResignFirstResponder` (page 462)

Related Sample Code

KeyboardAccessory

MoviePlayer

ToolbarSearch

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

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 touches belong.

Discussion

The default implementation of this method does nothing. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain.

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` (page 704) property of the corresponding view instance to `YES`.

Availability

Available in iOS 2.0 and later.

See Also

- `touchesMoved:withEvent:` (page 468)
- `touchesEnded:withEvent:` (page 468)
- `touchesCancelled:withEvent:` (page 467)

Declared In

`UIResponder.h`

touchesCancelled:withEvent:

Sent to the receiver when a system event (such as a low-memory warning) cancels a touch event.

```
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
```

Parameters

touches

A 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 interruption requiring cancellation of the touch event; for this, it generates a `UITouch` object with a phase of `UITouchPhaseCancel`. The interruption is something that might cause the application to be no longer active or the view to be removed from the window

When an object receives a `touchesCancelled:withEvent:` message it should clean up any state information that was established in its `touchesBegan:withEvent:` (page 467) implementation.

The default implementation of this method does nothing. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain.

Availability
Available in iOS 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

A 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. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain.

When an object receives a `touchesEnded:withEvent:` message it should clean up any state information that was established in its `touchesBegan:withEvent:` (page 467) implementation.

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` (page 704) property of the corresponding view instance to `YES`.

Availability
Available in iOS 2.0 and later.

See Also
- `touchesBegan:withEvent:` (page 467)
- `touchesMoved:withEvent:` (page 468)
- `touchesCancelled:withEvent:` (page 467)

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

A 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. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain.

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` (page 704) property of the corresponding view instance to `YES`.

Availability

Available in iOS 2.0 and later.

See Also

- `touchesBegan:withEvent:` (page 467)
- `touchesEnded:withEvent:` (page 468)
- `touchesCancelled:withEvent:` (page 467)

Declared In

`UIResponder.h`

UIRotationGestureRecognizer Class

Inherits from	UIGestureRecognizer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIRotationGestureRecognizer.h
Companion guide	Event Handling Guide for iOS
Related sample code	SimpleGestureRecognizers

Overview

`UIRotationGestureRecognizer` is a concrete subclass of `UIGestureRecognizer` that looks for rotation gestures involving two touches. When the user moves the fingers opposite each other in a circular motion, the underlying view should rotate in a corresponding direction and speed.

Rotation is a continuous gesture. It begins when two touches have moved enough to be considered a rotation. The gesture changes when a finger moves while the two fingers are down. It ends when both fingers have lifted. At each stage in the gesture, the gesture recognizer sends its action message.

Tasks

Interpreting the Gesture

`rotation` (page 472) *property*
The rotation of the gesture in radians since its last change.

`velocity` (page 472) *property*
The velocity of the rotation gesture in radians per second. (read-only)

Properties

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

rotation

The rotation of the gesture in radians since its last change.

```
@property(nonatomic) CGFloat rotation
```

Discussion
You may set the rotation value to an arbitrary value; however, setting the rotation resets the velocity.

Availability
Available in iOS 3.2 and later.

Declared In
`UIRotationGestureRecognizer.h`

velocity

The velocity of the rotation gesture in radians per second. (read-only)

```
@property(nonatomic, readonly) CGFloat velocity
```

Availability
Available in iOS 3.2 and later.

Declared In
`UIRotationGestureRecognizer.h`

UIScreen Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIScreen.h
Related sample code	MoviePlayer WiTap

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 476)
Returns the screen object representing the device's screen.

`+ screens` (page 476)
Returns an array containing all of the screens attached to the device.

Getting the Bounds Information

`bounds` (page 475) *property*
Contains the bounding rectangle of the screen, measured in points. (read-only)

`applicationFrame` (page 474) *property*
The frame rectangle to use for your application's window. (read-only)

`scale` (page 475) *property*
The natural scale factor associated with the screen. (read-only)

Accessing the Screen Modes

`availableModes` (page 474) *property*
The display modes that can be associated with the receiver. (read-only)

`currentMode` (page 475) *property*
The current screen mode associated with the receiver.

Getting a Display Link

`- displayLinkWithTarget:selector:` (page 476)
Returns a display link object for the current screen.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

Related Sample Code
WiTap

Declared In
`UIScreen.h`

availableModes

The display modes that can be associated with the receiver. (read-only)

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

Discussion
The array contains one or more `UIScreenMode` objects, each of which represents a display mode supported by the screen.

Availability
Available in iOS 3.2 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 iOS 2.0 and later.

Related Sample Code
MoviePlayer
WiTap

Declared In
UIScreen.h

currentMode

The current screen mode associated with the receiver.

```
@property(nonatomic,retain) UIScreenMode *currentMode
```

Discussion
The default value of this property is the mode containing the highest resolution supported by the screen. You can change the value of this property to support different resolutions as needed. For example, you might want to lower the default resolution to one that your application supports more readily.

Availability
Available in iOS 3.2 and later.

Declared In
UIScreen.h

scale

The natural scale factor associated with the screen. (read-only)

```
@property(nonatomic,readonly) CGFloat scale
```

Discussion
This value reflects the scale factor needed to convert from the default logical coordinate space into the device coordinate space of this screen. The default logical coordinate space is measured using points, where one point is approximately equal to 1/160th of an inch. If a device's screen has a reasonably similar pixel density, the scale factor is typically set to 1.0 so that one point maps to one pixel. However, a screen with a significantly different pixel density may set this property to a higher value.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Related Sample Code
MoviePlayer
WiTap

Declared In
`UIScreen.h`

screens

Returns an array containing all of the screens attached to the device.

```
+ (NSArray *)screens
```

Return Value
An array of `UIScreen` objects.

Discussion
The returned array includes the main screen plus any additional screens connected to the device. The main screen is always at index `0`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIScreen.h`

Instance Methods

displayLinkWithTarget:selector:

Returns a display link object for the current screen.

```
- (CADisplayLink *)displayLinkWithTarget:(id)target selector:(SEL)sel
```

Parameters

target

An object to be notified when the screen should be updated.

sel

The method of *target* to call. This selector must have the following signature:

```
- (void)selector:(CADisplayLink *)sender;
```

Return Value
A newly constructed display link object.

Discussion
You use display link objects to synchronize your drawing code to the screen's refresh rate. The newly constructed display link retains the target.

Availability
Available in iOS 4.0 and later.

Declared In
`UIScreen.h`

Notifications

UIScreenDidConnectNotification

This notification is posted when a new screen is connected to the device. The object of the notification is the `UIScreen` object representing the new screen. There is no *userInfo* dictionary.

Connection notifications are not sent for screens that are already present when the application is launched. The application can instead use the `screens` (page 476) method to get the current set of screens at launch time.

Availability
Available in iOS 3.2 and later.

Declared In
`UIScreen.h`

UIScreenDidDisconnectNotification

This notification is posted when a screen is disconnected from the device. The object of the notification is the `UIScreen` object that represented the now disconnected screen. There is no *userInfo* dictionary.

Availability
Available in iOS 3.2 and later.

Declared In
`UIScreen.h`

UIScreenModeDidChangeNotification

This notification is posted when the current mode of a screen changes. The object of the notification is the `UIScreen` object whose `currentMode` (page 475) property changed. There is no *`userInfo`* dictionary.

Clients can use this notification to detect changes in the screen resolution.

Availability
Available in iOS 3.2 and later.

Declared In
`UIScreen.h`

UIScreenMode Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIScreenMode.h

Overview

A `UIScreenMode` object represents a possible set of attributes that can be applied to a `UIScreen` object. The object encapsulates information about the size of the screen's underlying display buffer and the aspect ratio it uses for individual pixels.

Most developers should never need to use the information provided by this class and should simply use the bounds provided by the `UIScreen` object for their drawing space. The bounds of screen and window objects automatically take the pixel aspect ratio and underlying drawing hardware into consideration. However, developers that work with pixel-level information more directly may use the information in the current screen mode object to tailor their code for the target screen.

You do not create instances of this class directly. Instead, you get the screen modes supported by a given screen from the corresponding `UIScreen` object.

Tasks

Accessing the Screen Mode Attributes

`size` (page 480) *property*
The screen size, measured in pixels. (read-only)

`pixelAspectRatio` (page 480) *property*
The aspect ratio of a single pixel. (read-only)

Properties

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

pixelAspectRatio

The aspect ratio of a single pixel. (read-only)

```
@property(readonly,nonatomic) CGFloat pixelAspectRatio
```

Discussion
The aspect ratio is defined as `x/y`, where `x` is the width of the pixel and `y` is the height of the pixel.

Availability
Available in iOS 3.2 and later.

Declared In
`UIScreenMode.h`

size

The screen size, measured in pixels. (read-only)

```
@property(readonly,nonatomic) CGSize size
```

Discussion
The value in this property represents the size of the underlying display buffer.

Availability
Available in iOS 3.2 and later.

Declared In
`UIScreenMode.h`

UIScrollView Class Reference

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

Overview

The UIScrollView class provides support for displaying 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 with 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 their 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 496), `pagingEnabled` (page 490), and `touchesShouldCancelInContentView:` (page 496) 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 897) and `scrollViewDidEndZooming:withView:atScale:` (page 893); in addition, the maximum (`maximumZoomScale` (page 490)) and minimum (`minimumZoomScale` (page 490)) zoom scale must be different.

Tasks

Managing the Display of Content

- `setContentOffset:animated:` (page 495)
  Sets the offset from the content view's origin that corresponds to the receiver's origin.

`contentOffset` (page 486) *property*
The point at which the origin of the content view is offset from the origin of the scroll view.

`contentSize` (page 487) *property*
The size of the content view.

`contentInset` (page 486) *property*
The distance that the content view is inset from the enclosing scroll view.

Managing Scrolling

`scrollEnabled` (page 491) *property*
A Boolean value that determines whether scrolling is enabled.

`directionalLockEnabled` (page 489) *property*
A Boolean value that determines whether scrolling is disabled in a particular direction

`scrollsToTop` (page 491) *property*
A Boolean value that controls whether the scroll-to-top gesture is effective

- `scrollRectToVisible:animated:` (page 494)
  Scrolls a specific area of the content so that it is visible in the receiver.

`pagingEnabled` (page 490) *property*
A Boolean value that determines whether paging is enabled for the scroll view.

`bounces` (page 485) *property*
A Boolean value that controls whether the scroll view bounces past the edge of content and back again.

`alwaysBounceVertical` (page 485) *property*
A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content.

`alwaysBounceHorizontal` (page 484) *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 496)
Overridden by subclasses to customize the default behavior when a finger touches down in displayed content.

- `touchesShouldCancelInContentView:` (page 496)
Returns whether to cancel touches related to the content subview and start dragging.

`canCancelContentTouches` (page 486) *property*
A Boolean value that controls whether touches in the content view always lead to tracking.

`delaysContentTouches` (page 488) *property*
A Boolean value that determines whether the scroll view delays the handling of touch-down gestures.

`decelerationRate` (page 488) *property*
A floating-point value that determines the rate of deceleration after the user lifts their finger.

`dragging` (page 489) *property*
A Boolean value that indicates whether the user has begun scrolling the content. (read-only)

`tracking` (page 492) *property*
Returns whether the user has touched the content to initiate scrolling. (read-only)

`decelerating` (page 487) *property*
Returns whether the content is moving in the scroll view after the user lifted their finger. (read-only)

Managing the Scroll Indicator

`indicatorStyle` (page 489) *property*
The style of the scroll indicators.

`scrollIndicatorInsets` (page 491) *property*
The distance the scroll indicators are inset from the edge of the scroll view.

`showsHorizontalScrollIndicator` (page 492) *property*
A Boolean value that controls whether the horizontal scroll indicator is visible.

`showsVerticalScrollIndicator` (page 492) *property*
A Boolean value that controls whether the vertical scroll indicator is visible.

- `flashScrollIndicators` (page 494)
Displays the scroll indicators momentarily.

Zooming and Panning

- `zoomToRect:animated:` (page 497)
Zooms to a specific area of the content so that it is visible in the receiver.

`zoomScale` (page 494) *property*
A floating-point value that specifies the current scale factor applied to the scroll view's content.

- `setZoomScale:animated:` (page 495)
  A floating-point value that specifies the current zoom scale.

`maximumZoomScale` (page 490) *property*
A floating-point value that specifies the maximum scale factor that can be applied to the scroll view's content.

`minimumZoomScale` (page 490) *property*
A floating-point value that specifies the minimum scale factor that can be applied to the scroll view's content.

`zoomBouncing` (page 493) *property*
A Boolean value that indicates that zooming has exceeded the scaling limits specified for the receiver. (read-only)

`zooming` (page 493) *property*
A Boolean value that indicates whether the content view is currently zooming in or out. (read-only)

`bouncesZoom` (page 485) *property*
A Boolean value that determines whether the scroll view animates the content scaling when the scaling exceeds the maximum or minimum limits.

Managing the Delegate

`delegate` (page 488) *property*
The delegate of the scroll-view object.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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
If this property is set to `YES` and `bounces` (page 485) is `YES`, horizontal dragging is allowed even if the content is smaller than the bounds of the scroll view. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`@property alwaysBounceVertical` (page 485)

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
If this property is set to `YES` and `bounces` (page 485) is `YES`, vertical dragging is allowed even if the content is smaller than the bounds of the scroll view. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`@property alwaysBounceHorizontal` (page 484)

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 this property is `YES`, 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. The default value is `YES`.

Availability
Available in iOS 2.0 and later.

See Also
`@property alwaysBounceVertical` (page 485)
`@property alwaysBounceHorizontal` (page 484)

Declared In
`UIScrollView.h`

bouncesZoom

A Boolean value that determines whether the scroll view animates the content scaling when the scaling exceeds the maximum or minimum limits.

```
@property(nonatomic) BOOL bouncesZoom
```

Discussion
If the value of this property is `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 this property is `NO`, zooming stops immediately at one a scaling limits. The default is `YES` .

Availability
Available in iOS 2.0 and later.

See Also
`@property maximumZoomScale` (page 490)
`@property minimumZoomScale` (page 490)
`@property zoomBouncing` (page 493)
`@property zooming` (page 493)

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 this property is `YES` and a view in the content has begun tracking a finger touching it, and if the user drags the finger enough to initiate a scroll, the view receives a `touchesCancelled:withEvent:` (page 467) message and the scroll view handles the touch as a scroll. If the value of this property is `NO`, the scroll view does not scroll regardless of finger movement once the content view starts tracking.

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

See Also
`- setContentOffset:animated:` (page 495)

Related Sample Code
ScrollViewSuite

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 iOS 2.0 and later.

Related Sample Code
ScrollViewSuite

Declared In
`UIScrollView.h`

decelerating

Returns whether the content is moving in the scroll view after the user lifted their 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 iOS 2.0 and later.

Declared In
`UIScrollView.h`

decelerationRate

A floating-point value that determines the rate of deceleration after the user lifts their finger.

```
@property(nonatomic) float decelerationRate
```

Discussion
Your application can use the `UIScrollViewDecelerationRateNormal` (page 498) and `UIScrollViewDecelerationRateFast` (page 498) constants as reference points for reasonable deceleration rates.

Availability
Available in iOS 3.0 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 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 496). The default value is `YES`.

See the class description for a fuller discussion.

Availability
Available in iOS 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 iOS 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
If this property is `NO`, scrolling is permitted in both horizontal and vertical directions. If this property is `YES` and the user begins dragging in one general direction (horizontally or vertically), the scroll view disables scrolling in the other direction. If the drag direction is diagonal, then scrolling will not be locked and the user can drag in any direction until the drag completes. The default value is `NO`

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property tracking` (page 492)

Declared In
`UIScrollView.h`

indicatorStyle

The style of the scroll indicators.

```
@property(nonatomic) UIScrollViewIndicatorStyle indicatorStyle
```

Discussion
The default style is `UIScrollViewIndicatorStyleDefault` (page 497). See "Scroll Indicator Style" (page 497) for descriptions of these constants.

Availability
Available in iOS 2.0 and later.

Declared In
`UIScrollView.h`

maximumZoomScale

A floating-point value that specifies the maximum scale factor that can be applied to the scroll view's content.

```
@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 iOS 2.0 and later.

See Also
- `@property minimumZoomScale` (page 490)
- `@property zoomBouncing` (page 493)
- `@property bouncesZoom` (page 485)
- `@property zoomBouncing` (page 493)
- `@property zooming` (page 493)

Declared In
`UIScrollView.h`

minimumZoomScale

A floating-point value that specifies the minimum scale factor that can be applied to the scroll view's content.

```
@property(nonatomic) float minimumZoomScale
```

Discussion
This value determines how small the content can be scaled. The default value is `1.0`

Availability
Available in iOS 2.0 and later.

See Also
- `@property maximumZoomScale` (page 490)
- `@property zoomBouncing` (page 493)
- `@property bouncesZoom` (page 485)
- `@property zoomBouncing` (page 493)
- `@property zooming` (page 493)

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 this 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 iOS 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 this property is `YES` , scrolling is enabled, and if it is `NO` , scrolling is disabled. The default is `YES`.

Availability
Available in iOS 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 iOS 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`.

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:`, UIScrollView ignores the request. After the scroll view scrolls to the top of the content view, it sends the delegate a `scrollViewDidScrollToTop:` (page 894) message.

Availability
Available in iOS 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 iOS 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 iOS 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 value of this property is `YES` if the user has touched the content view but might not have yet have started dragging it.

Availability
Available in iOS 2.0 and later.

See Also
`@property dragging` (page 489)

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 iOS 2.0 and later.

See Also
`@property maximumZoomScale` (page 490)
`@property minimumZoomScale` (page 490)
`@property zooming` (page 493)
`@property bouncesZoom` (page 485)

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 iOS 2.0 and later.

See Also
`@property maximumZoomScale` (page 490)
`@property minimumZoomScale` (page 490)
`@property zoomBouncing` (page 493)
`@property bouncesZoom` (page 485)

Declared In
`UIScrollView.h`

zoomScale

A floating-point value that specifies the current scale factor applied to the scroll view's content.

```
@property(nonatomic) float zoomScale
```

Discussion
This value determines how much the content is currently scaled. The default value is `1.0`.

Availability
Available in iOS 3.0 and later.

Related Sample Code
ScrollViewSuite

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 iOS 2.0 and later.

Declared In
`UIScrollView.h`

scrollRectToVisible:animated:

Scrolls a specific area of the content so that it is visible in the receiver.

```
- (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 iOS 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 iOS 2.0 and later.

See Also
@property contentOffset (page 486)

Declared In
UIScrollView.h

setZoomScale:animated:

A floating-point value that specifies the current zoom scale.

```
- (void)setZoomScale:(float)scale animated:(BOOL)animated
```

Parameters

scale
The new value to scale the content to.

animated
YES to animate the transition to the new scale, NO to make the transition immediate.

Discussion
The new scale value should be between the minimumZoomScale and the maximumZoomScale.

Availability
Available in iOS 3.0 and later.

Related Sample Code
ScrollViewSuite

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 iOS 2.0 and later.

See Also

`- touchesShouldCancelInContentView:` (page 496)

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 486) property is `NO`.

Availability

Available in iOS 2.0 and later.

See Also
- `touchesShouldBegin:withEvent:inContentView:` (page 496)

Declared In
`UIScrollView.h`

zoomToRect:animated:

Zooms to a specific area of the content so that it is visible in the receiver.

```
- (void)zoomToRect:(CGRect)rectanimated:(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 zooms so that the content view becomes the area defined by *rect*, adjusting the `zoomScale` as necessary.

Availability
Available in iOS 3.0 and later.

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 489) 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 iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `UIScrollView.h`.

Deceleration Constants

The rate of deceleration for a scrolling view.

```
const float UIScrollViewDecelerationRateNormal;
const float UIScrollViewDecelerationRateFast;
```

Constants

`UIScrollViewDecelerationRateNormal`
: The default deceleration rate for a scroll view.

  Available in iOS 3.0 and later.

  Declared in `UIScrollView.h`.

`UIScrollViewDecelerationRateFast`
: A fast deceleration rate for a scroll view.

  Available in iOS 3.0 and later.

  Declared in `UIScrollView.h`.

UISearchBar Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UISearchBar.h
Related sample code	ToolbarSearch

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. The `UISearchBar` object does not actually perform any searches. You use a delegate, an object conforming to the `UISearchBarDelegate` protocol, to implement the actions when text is entered and buttons are clicked.

Tasks

Text Content

`placeholder` (page 502) *property*
The string that is displayed when there is no other text in the text field.

`prompt` (page 503) *property*
A single line of text displayed at the top of the search bar.

`text` (page 505) *property*
The current or starting search text.

Display Attributes

`barStyle` (page 501) *property*
The style that specifies the receiver's appearance.

`tintColor` (page 506) *property*
The color used to tint the bar.

`translucent` (page 506) *property*
Specifies whether the receiver is translucent.

Text Input Properties

`autocapitalizationType` (page 501) *property*
The auto-capitalization style for the text object.

`autocorrectionType` (page 501) *property*
The auto-correction style for the text object.

`keyboardType` (page 502) *property*
The keyboard style associated with the text object.

Button Configuration

`showsBookmarkButton` (page 504) *property*
A Boolean value indicating whether the bookmark button is displayed.

`showsCancelButton` (page 504) *property*
A Boolean value indicating whether the cancel button is displayed.

- `setShowsCancelButton:animated:` (page 506)
Sets the display state of the cancel button optionally with animation.

`showsSearchResultsButton` (page 505) *property*
A Boolean value indicating whether the search results button is displayed.

`searchResultsButtonSelected` (page 503) *property*
A Boolean value indicating whether the search results button is selected.

Scope Buttons

`scopeButtonTitles` (page 503) *property*
An array of strings indicating the titles of the scope buttons.

`selectedScopeButtonIndex` (page 504) *property*
The index of the selected scope button.

`showsScopeBar` (page 505) *property*
Specifies whether the scope bar is displayed.

Delegate

`delegate` (page 502) *property*
The search bar's delegate object.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 988).

Availability
Available in iOS 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 989), which for most input methods results in auto-correction being enabled.

Availability
Available in iOS 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 1011) for possible values. The default value is `UIBarStyleDefault` (page 1011).

Availability
Available in iOS 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 iOS 2.0 and later.

Related Sample Code
ToolbarSearch

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 990).

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

Declared In
`UISearchBar.h`

scopeButtonTitles

An array of strings indicating the titles of the scope buttons.

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

Discussion
The order of the strings in the array indicates the order that the corresponding buttons will be displayed, from left to right. The index in the array corresponds to the index used in `selectedScopeButtonIndex` (page 504).

Availability
Available in iOS 3.0 and later.

See Also
`@property selectedScopeButtonIndex` (page 504)
`@property showsScopeBar` (page 505)

Declared In
`UISearchBar.h`

searchResultsButtonSelected

A Boolean value indicating whether the search results button is selected.

```
@property(nonatomic, getter=isSearchResultsButtonSelected) BOOL
    searchResultsButtonSelected
```

Discussion
The default value is `NO`.

Availability
Available in iOS 3.2 and later.

Declared In
`UISearchBar.h`

selectedScopeButtonIndex

The index of the selected scope button.

```
@property(nonatomic) NSInteger selectedScopeButtonIndex
```

Discussion
The indexes of the scope buttons are determined by the indexes of the strings in `scopeButtonTitles` (page 503).

Availability
Available in iOS 3.0 and later.

See Also
`@property scopeButtonTitles` (page 503)
`@property showsScopeBar` (page 505)

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 iOS 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 iOS 2.0 and later.

See Also
`@property showsCancelButton` (page 504)

Declared In
`UISearchBar.h`

showsScopeBar

Specifies whether the scope bar is displayed.

```
@property(nonatomic) BOOL showsScopeBar
```

Availability
Available in iOS 3.0 and later.

See Also
`@property scopeButtonTitles` (page 503)

Declared In
`UISearchBar.h`

showsSearchResultsButton

A Boolean value indicating whether the search results button is displayed.

```
@property(nonatomic) BOOL showsSearchResultsButton
```

Discussion
The default value is `NO`.

Availability
Available in iOS 3.2 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 iOS 2.0 and later.

Related Sample Code
ToolbarSearch

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 iOS 2.0 and later.

Declared In
UISearchBar.h

translucent

Specifies whether the receiver is translucent.

```
@property(nonatomic, assign, getter=isTranslucent) BOOL translucent
```

Availability
Available in iOS 3.0 and later.

Declared In
UISearchBar.h

Instance Methods

setShowsCancelButton:animated:

Sets the display state of the cancel button optionally with animation.

```
- (void)setShowsCancelButton:(BOOL)showsCancelButton animated:(BOOL)animated
```

Parameters

showsCancelButton
`YES` to display the cancel button, otherwise `NO`.

animated
`YES` to use animation to change the display state of the cancel button, otherwise `NO`.

Availability
Available in iOS 3.0 and later.

See Also
`- setShowsCancelButton:animated:` (page 506)

Declared In
`UISearchBar.h`

UISearchDisplayController Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	UIKit/UISearchDisplayController.h

Overview

A search display controller manages display of a search bar and a table view that displays the results of a search of data managed by another view controller.

You initialize a search display controller with a search bar and a view controller responsible for managing the original content to be searched. When the user starts a search, the search display controller is responsible for superimposing the search interface over the original view controller's view and showing the search results. The results are displayed in a table view that's created by the search display controller. In addition to the original view controller, there are logically four other roles. These are typically all be played by the same object, often the original view controller itself.

1. The search results table view's data source.

   This object is responsible for providing the data for the results table.

2. The search results table view's delegate.

   This object is responsible for, amongst other things, responding to the user's selection of an item in the results table.

3. The search display controller's delegate.

   The delegate conforms to the `UISearchDisplayDelegate` protocol. It is notified of events such as when the search starts or ends, and when the search interface is displayed or hidden. As a convenience, it may also be told about changes to the search string or search scope, so that the results table view can be reloaded.

4. The search bar's delegate.

   This object is responsible for responding to changes in the search criteria.

Typically you initialize a search display controller from a view controller (usually an instance of `UITableViewController`) that's displaying a list; you set `self` for the search display controller's view controller and search results data source and delegate:

```
searchController = [[UISearchDisplayController alloc]
                        initWithSearchBar:searchBar contentsController:self];
searchController.delegate = self;
searchController.searchResultsDataSource = self;
searchController.searchResultsDelegate = self;
```

If you follow this pattern, then in the table view data source and delegate methods you can check the methods' table view argument to determine which table view is sending the message:

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {

    if (tableView == self.tableView) {
        return ...;
    }
    // If necessary (if self is the data source for other table views),
    // check whether tableView is searchController.searchResultsTableView.
    return ...;
}
```

Important: Any given view controller or search bar can only be associated with a single search display controller at a time. If a search display controller is destroyed (for example, in response to a memory warning), then you can create a new one and associate it with the original view controller or search bar.

Tasks

Initialization

`- initWithSearchBar:contentsController:` (page 513)
Returns a display controller initialized with the given search bar and contents controller.

Displaying the Search Interface

`active` (page 511) *property*
The visibility state of the search interface.

`- setActive:animated:` (page 513)
Displays or hides the search interface, optionally with animation.

Configuration

`delegate` (page 511) *property*
The controller's delegate.

`searchBar` (page 512) *property*
The search bar. (read-only)

`searchContentsController` (page 512) *property*
The view controller that manages the contents being searched. (read-only)

`searchResultsTableView` (page 513) *property*
The table view in which the search results are displayed. (read-only)

`searchResultsDataSource` (page 512) *property*
The data source for the table view in which the search results are displayed.

`searchResultsDelegate` (page 512) *property*
The delegate for the table view in which the search results are displayed.

Properties

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

active

The visibility state of the search interface.

```
@property(nonatomic, getter=isActive) BOOL active
```

Discussion
The default value is `NO`.

If you set this value directly, any change is performed without animation. Use `setActive:animated:` (page 513) if a change in state should be animated.

When the user focus in the search field of a managed search bar, the search display controller automatically displays the search interface. You can use this property to force the search interface to appear.

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchDisplayController.h`

delegate

The controller's delegate.

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

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchDisplayController.h`

searchBar

The search bar. (read-only)

```
@property(nonatomic, readonly) UISearchBar *searchBar
```

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchDisplayController.h`

searchContentsController

The view controller that manages the contents being searched. (read-only)

```
@property(nonatomic, readonly) UIViewController *searchContentsController
```

Discussion
This is typically an instance of `UITableViewController`.

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchDisplayController.h`

searchResultsDataSource

The data source for the table view in which the search results are displayed.

```
@property(nonatomic, assign) id<UITableViewDataSource> searchResultsDataSource
```

Discussion
The default is `nil`.

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchDisplayController.h`

searchResultsDelegate

The delegate for the table view in which the search results are displayed.

```
@property(nonatomic, assign) id<UITableViewDelegate> searchResultsDelegate
```

Discussion
The default is `nil`.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchResultsTableView

The table view in which the search results are displayed. (read-only)

```
@property(nonatomic, readonly) UITableView *searchResultsTableView
```

Discussion
This method creates a new table view if one does not already exist.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

Instance Methods

initWithSearchBar:contentsController:

Returns a display controller initialized with the given search bar and contents controller.

```
- (id)initWithSearchBar:(UISearchBar *)searchBar contentsController:(UIViewController
    *)viewController
```

Parameters

searchBar

A search bar.

The search bar must not currently be associated with another search display controller.

viewController

The view controller that manages display of the original contents that are to be searched.

The view controller must not currently be associated with another search display controller.

Return Value
A display controller initialized with the given search bar and contents controller.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

setActive:animated:

Displays or hides the search interface, optionally with animation.

```
- (void)setActive:(BOOL)visible animated:(BOOL)animated
```

Parameters

visible

YES to display the search interface if it is not already displayed; NO to hide the search interface if it is currently displayed.

animated;

YES to use animation for a change in visible state, otherwise NO.

Discussion

When the user focus in the search field of a managed search bar, the search display controller automatically displays the search interface. You can use this method to force the search interface to appear.

Availability

Available in iOS 3.0 and later.

Declared In

UISearchDisplayController.h

UISegmentedControl Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UISegmentedControl.h
Related sample code	SimpleGestureRecognizers

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. When you add and remove segments, you can request that the action be animated with sliding and fading effects.

You register the target-action methods for a segmented control using the `UIControlEventValueChanged` constant as shown below.

```
[segmentedControl addTarget:self
                     action:@selector(action:)
           forControlEvents:UIControlEventValueChanged];
```

How you configure a segmented control can affect its display behavior:

- 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.
- Prior to iOS 3.0, if a segmented control has only two segments, then it behaves like a switch—tapping the currently-selected segment causes the other segment to be selected. (On iOS 3.0 and later, tapping the currently-selected segment does not cause the other segment to be selected.)

Tasks

Initializing a Segmented Control

- `initWithItems:` (page 520)
  Initializes and returns a segmented control with segments having the given titles or images.

Managing Segment Content

- `setImage:forSegmentAtIndex:` (page 523)
  Sets the content of a segment to a given image.
- `imageForSegmentAtIndex:` (page 519)
  Returns the image for a specific segment
- `setTitle:forSegmentAtIndex:` (page 524)
  Sets the title of a segment.
- `titleForSegmentAtIndex:` (page 525)
  Returns the title of the specified segment.

Managing Segments

- `insertSegmentWithImage:atIndex:animated:` (page 520)
  Inserts a segment at a specified position in the receiver and gives it an image as content.
- `insertSegmentWithTitle:atIndex:animated:` (page 520)
  Inserts a segment at a specific position in the receiver and gives it a title as content.

`numberOfSegments` (page 517) *property*
Returns the number of segments the receiver has. (read-only)

- `removeAllSegments` (page 521)
  Removes all segments of the receiver
- `removeSegmentAtIndex:animated:` (page 522)
  Removes the specified segment from the receiver, optionally animating the transition.

`selectedSegmentIndex` (page 518) *property*
The index number identifying the selected segment (that is, the last segment touched).

Managing Segment Behavior and Appearance

`momentary` (page 517) *property*
A Boolean value that determines whether segments in the receiver show selected state.

`segmentedControlStyle` (page 518) *property*
The style of the segmented control.

`tintColor` (page 518) *property*
The tint color of the segmented control.

- `setEnabled:forSegmentAtIndex:` (page 523)
  Enables the specified segment.
- `isEnabledForSegmentAtIndex:` (page 521)
  Returns whether the indicated segment is enabled.
- `setContentOffset:forSegmentAtIndex:` (page 522)
  Adjusts the offset for drawing the content (image or text) of the specified segment.
- `contentOffsetForSegmentAtIndex:` (page 519)
  Returns the offset for drawing the content (image or text) of the specified segment.
- `setWidth:forSegmentAtIndex:` (page 524)
  Sets the width of the specified segment of the receiver.
- `widthForSegmentAtIndex:` (page 525)
  Returns the width of the indicated segment of the receiver.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 `selectedSegmentIndex` (page 518) after tracking ends.

Availability
Available in iOS 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 iOS 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 526). See "Segmented Control Style" (page 526) for descriptions of valid constants.

Availability
Available in iOS 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 527) (no segment selected) until the user touches a segment. Set this property to -1 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 224) 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 iOS 2.0 and later.

See Also
`@property momentary` (page 517)

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` (page 526).

Availability
Available in iOS 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 517)) 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 iOS 2.0 and later.

See Also

`- setContentOffset:forSegmentAtIndex:` (page 522)

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 517)) 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 iOS 2.0 and later.

See Also

`- setImage:forSegmentAtIndex:` (page 523)

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 iOS 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 517)) 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 iOS 2.0 and later.

See Also

- `insertSegmentWithTitle:atIndex:animated:` (page 520)
- `removeSegmentAtIndex:animated:` (page 522)

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 517)) 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 iOS 2.0 and later.

See Also
- `insertSegmentWithImage:atIndex:animated:` (page 520)
- `removeSegmentAtIndex:animated:` (page 522)

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 517)) 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 iOS 2.0 and later.

See Also
- `setEnabled:forSegmentAtIndex:` (page 523)

Declared In
`UISegmentedControl.h`

removeAllSegments

Removes all segments of the receiver

```
- (void)removeAllSegments
```

Availability
Available in iOS 2.0 and later.

See Also
- `removeSegmentAtIndex:animated:` (page 522)

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 517)) 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 iOS 2.0 and later.

See Also
- `removeAllSegments` (page 521)
- `insertSegmentWithImage:atIndex:animated:` (page 520)
- `insertSegmentWithTitle:atIndex:animated:` (page 520)

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 517)) minus 1; values exceeding this upper range are pinned to it.

Availability
Available in iOS 2.0 and later.

See Also
- `contentOffsetForSegmentAtIndex:` (page 519)

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 517)) minus 1; values exceeding this upper range are pinned to it.

Availability
Available in iOS 2.0 and later.

See Also
- `isEnabledForSegmentAtIndex:` (page 521)

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 517)) 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 iOS 2.0 and later.

See Also
- `imageForSegmentAtIndex:` (page 519)

Declared In
UISegmentedControl.h

setTitle:forSegmentAtIndex:

Sets the title of a segment.

```
- (void)setTitle:(NSString *)title forSegmentAtIndex:(NSUInteger)segment
```

Parameters

title

A 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 517)) 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 iOS 2.0 and later.

See Also
- titleForSegmentAtIndex: (page 525)

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 517)) minus 1; values exceeding this upper range are pinned to it.

Availability
Available in iOS 2.0 and later.

See Also
- widthForSegmentAtIndex: (page 525)

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 517)) 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 iOS 2.0 and later.

See Also

`- setTitle:forSegmentAtIndex:` (page 524)

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 517)) 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 iOS 2.0 and later.

See Also

`- setWidth:forSegmentAtIndex:` (page 524)

Declared In

`UISegmentedControl.h`

Constants

UISegmentedControlStyle

The styles of the segmented control.

```
typedef enum {
    UISegmentedControlStylePlain,
    UISegmentedControlStyleBordered,
    UISegmentedControlStyleBar,
    UISegmentedControlStyleBezeled,
} UISegmentedControlStyle;
```

Constants

`UISegmentedControlStylePlain`
The large plain style for segmented controls. This style is the default.
Available in iOS 2.0 and later.
Declared in `UISegmentedControl.h`.

`UISegmentedControlStyleBordered`
The large bordered style for segmented controls.
Available in iOS 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 518)).
Available in iOS 2.0 and later.
Declared in `UISegmentedControl.h`.

`UISegmentedControlStyleBezeled`
The large bezeled style for segmented controls. Segmented controls in this style can have a tint color (see `tintColor` (page 518)).
Available in iOS 4.0 and later.
Declared in `UISegmentedControl.h`.

Discussion
You use these constants as values for the `segmentedControlStyle` (page 518) 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 `selectedSegmentIndex` (page 518) for further information.

Available in iOS 2.0 and later.

Declared in `UISegmentedControl.h`.

Declared In

`UISegmentedControl.h`

UISlider Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 534) *property*
Contains the receiver's current value.

- `setValue:animated:` (page 538)
Sets the receiver's current value, allowing you to animate the change visually.

Accessing the Slider's Value Limits

`minimumValue` (page 534) *property*
Contains the minimum value of the receiver.

`maximumValue` (page 533) *property*
Contains the maximum value of the receiver.

Modifying the Slider's Behavior

`continuous` (page 531) *property*
Contains a Boolean value indicating whether changes in the sliders value generate continuous update events.

Changing the Slider's Appearance

`minimumValueImage` (page 534) *property*
Contains the image that is drawn on the side of the slider representing the minimum value.

`maximumValueImage` (page 533) *property*
Contains the image that is drawn on the side of the slider representing the maximum value.

`currentMinimumTrackImage` (page 532) *property*
Contains the minimum track image currently being used to render the receiver. (read-only)

- `minimumTrackImageForState:` (page 536)
Returns the minimum track image associated with the specified control state.

- `setMinimumTrackImage:forState:` (page 537)
Assigns a minimum track image to the specified control states.

`currentMaximumTrackImage` (page 531) *property*
Contains the maximum track image currently being used to render the receiver. (read-only)

- `maximumTrackImageForState:` (page 535)
Returns the maximum track image associated with the specified control state.

- `setMaximumTrackImage:forState:` (page 537)
  Assigns a maximum track image to the specified control states.

`currentThumbImage` (page 532) *property*
Contains the thumb image currently being used to render the receiver. (read-only)

- `thumbImageForState:` (page 539)
  Returns the thumb image associated with the specified control state.
- `setThumbImage:forState:` (page 538)
  Assigns a thumb image to the specified control states.

Overrides for Subclasses

- `maximumValueImageRectForBounds:` (page 535)
  Returns the drawing rectangle for the maximum value image.
- `minimumValueImageRectForBounds:` (page 536)
  Returns the drawing rectangle for the minimum value image.
- `trackRectForBounds:` (page 540)
  Returns the drawing rectangle for the slider's track.
- `thumbRectForBounds:trackRect:value:` (page 539)
  Returns the drawing rectangle for the slider's thumb image.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 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 iOS 2.0 and later.

See Also
- `maximumTrackImageForState:` (page 535)
- `setMaximumTrackImage:forState:` (page 537)

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 iOS 2.0 and later.

See Also
- `minimumTrackImageForState:` (page 536)
- `setMinimumTrackImage:forState:` (page 537)

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 iOS 2.0 and later.

See Also
- `thumbImageForState:` (page 539)
- `setThumbImage:forState:` (page 538)

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
`- setValue:animated:` (page 538)

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 529).

Availability
Available in iOS 2.0 and later.

See Also
`- setMaximumTrackImage:forState:` (page 537)

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 iOS 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 529).

Availability
Available in iOS 2.0 and later.

See Also
- setMinimumTrackImage:forState: (page 537)

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 iOS 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 303) property. For more information about how this value defines the regions of the slider, see the `UIImage` class.

Availability

Available in iOS 2.0 and later.

See Also

`- maximumTrackImageForState:` (page 535)

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 303) property. For more information about how this value defines the regions of the slider, see the `UIImage` class.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- thumbImageForState: (page 539)

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 iOS 2.0 and later.

See Also
@property value (page 534)

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 529).

Availability

Available in iOS 2.0 and later.

See Also

- setThumbImage:forState: (page 538)

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 540) 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 iOS 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 iOS 2.0 and later.

Declared In

`UISlider.h`

UISplitViewController Class Reference

Inherits from	UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UISplitViewController.h
Companion guide	iPad Programming Guide
Related sample code	MultipleDetailViews

Overview

The `UISplitViewController` class is a container view controller that manages the presentation of two side-by-side view controllers. You use this class to implement a master-detail interface, in which the left-side view controller presents a list of items and the right-side presents details of the selected item. Split view controllers are for use exclusively on iPad devices. Attempting to create one on other devices results in an exception.

After creating and initializing an instance of this class, you must assign two view controllers to the `viewControllers` (page 543) property. The split view controller has no significant interface of its own. Its job is to coordinate the presentation of its two child view controllers and to manage the transitions among different orientations.

A split view controller supports the same interface orientations as its currently visible child view controllers. Both view controllers are displayed in landscape orientations but only the detail view controller is displayed in portrait orientations. When transitioning between orientations, the split view controller sends messages to its `delegate` (page 542) object to coordinate the display of a popover with the hidden view controller. For more information on the methods of this delegate object, see *UISplitViewControllerDelegate Protocol Reference*.

Note: A split view controller does not provide any inherent support for managing the communication between the custom view controllers you assign to it. It is your responsibility to determine the best way to do that. However, the delegation pattern often works well for master-detail interfaces. To implement such a pattern, your master view controller sends messages to a custom delegate object whenever the selected item changed or some other relevant event occurred. The detail view controller would then assign itself as the delegate of the master and would use the associated messages to refresh its contents.

Message Forwarding to Its Child View Controllers

A split view controller interposes itself between the application's window and its child view controllers. As a result, all messages to the visible view controllers must flow through the split view controller. This works generally as you might expect and the flow of messages should be relatively intuitive. For example, view appearance and disappearance messages are sent only when the corresponding child view controller actually appears on screen. Thus, when a split view controller is first displayed in a portrait orientation, it calls the `viewWillAppear:` (page 773) and `viewDidAppear:` (page 771) methods of only the view controller that is shown initially. The view controller that is presented using a popover does not receive those messages until the popover is shown or until the split view controller rotates to a landscape orientation.

Tasks

Managing the Child View Controllers

`viewControllers` (page 543) *property*
The array of view controllers managed by the receiver.

Accessing the Delegate Object

`delegate` (page 542) *property*
The delegate you want to receive split view controller messages.

Properties

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

delegate

The delegate you want to receive split view controller messages.

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

Discussion
The split view controller uses its delegate to manage the showing and hiding of related view controllers. For more information about the methods you can implement in your delegate, see *UISplitViewControllerDelegate Protocol Reference*.

Availability
Available in iOS 3.2 and later.

Declared In
`UISplitViewController.h`

viewControllers

The array of view controllers managed by the receiver.

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

Discussion
The array in this property must contain exactly two view controllers. The view controllers are presented left-to-right in the split view interface when it is in a landscape orientation. Thus, the view controller at index 0 is displayed on the left side and the view controller at index 1 is displayed on the right side of the interface.

The first view controller in this array is typically hidden when the device is in a portrait orientation. Assign a delegate object to the receiver if you want to coordinate the display of this view controller using a popover.

Availability
Available in iOS 3.2 and later.

Related Sample Code
MultipleDetailViews

Declared In
`UISplitViewController.h`

UISwipeGestureRecognizer Class Reference

Inherits from	UIGestureRecognizer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UISwipeGestureRecognizer.h
Companion guide	Event Handling Guide for iOS
Related sample code	SimpleGestureRecognizers

Overview

`UISwipeGestureRecognizer` is a concrete subclass of `UIGestureRecognizer` that looks for swiping gestures in one or more directions. A swipe is a discrete gesture, and thus the associated action message is sent only once per gesture.

`UISwipeGestureRecognizer` recognizes a swipe when the specified number of touches (`numberOfTouchesRequired` (page 546)) have moved mostly in an allowable direction (`direction` (page 546)) far enough to be considered a swipe. Swipes can be slow or fast. A slow swipe requires high directional precision but a small distance; a fast swipe requires low directional precision but a large distance.

You may determine the location where a swipe began by calling the `UIGestureRecognizer` methods `locationInView:` (page 289) and `locationOfTouch:inView:` (page 290). The former method gives you the centroid if more than one touch was involved in the gesture; the latter gives the location of a particular touch.

Tasks

Configuring the Gesture

`direction` (page 546) *property*
The permitted directions of the swipe.

`numberOfTouchesRequired` (page 546) *property*
The number of touches that must be present for the swipe gesture to be recognized.

Properties

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

direction

The permitted directions of the swipe.

```
@property(nonatomic) UISwipeGestureRecognizerDirection direction
```

Discussion
You may specify multiple directions by specifying multiple `UISwipeGestureRecognizerDirection` (page 546) constants using bitwise-OR operands. The default direction is `UISwipeGestureRecognizerDirectionRight` (page 547).

Availability
Available in iOS 3.2 and later.

Related Sample Code
SimpleGestureRecognizers

Declared In
`UISwipeGestureRecognizer.h`

numberOfTouchesRequired

The number of touches that must be present for the swipe gesture to be recognized.

```
@property(nonatomic) NSUInteger numberOfTouchesRequired
```

Discussion
The default value is 1.

Availability
Available in iOS 3.2 and later.

Declared In
`UISwipeGestureRecognizer.h`

Constants

UISwipeGestureRecognizerDirection

The direction of the swipe.

```
typedef enum {
    UISwipeGestureRecognizerDirectionRight = 1 << 0,
    UISwipeGestureRecognizerDirectionLeft  = 1 << 1,
    UISwipeGestureRecognizerDirectionUp    = 1 << 2,
    UISwipeGestureRecognizerDirectionDown  = 1 << 3
} UISwipeGestureRecognizerDirection;
```

Constants

`UISwipeGestureRecognizerDirectionRight`

The touch or touches swipe to the right. This direction is the default.

Available in iOS 3.2 and later.

Declared in `UISwipeGestureRecognizer.h`.

`UISwipeGestureRecognizerDirectionLeft`

The touch or touches swipe to the left.

Available in iOS 3.2 and later.

Declared in `UISwipeGestureRecognizer.h`.

`UISwipeGestureRecognizerDirectionUp`

The touch or touches swipe to the up.

Available in iOS 3.2 and later.

Declared in `UISwipeGestureRecognizer.h`.

`UISwipeGestureRecognizerDirectionDown`

The touch or touches swipe to the down.

Available in iOS 3.2 and later.

Declared in `UISwipeGestureRecognizer.h`.

Availability

Available in iOS 3.2 and later.

Declared In

`UISwipeGestureRecognizer.h`

UISwitch Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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. As with `UISlider`, when the user manipulates the switch control ("flips" it) a `UIControlEventValueChanged` (page 224) 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 550)
Returns an initialized switch object.

Setting the Off/On State

`on` (page 550) *property*
A Boolean value that determines the off/on state of the switch.

`- setOn:animated:` (page 550)
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 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 iOS 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 729) and enforces a size appropriate for the control.

Availability
Available in iOS 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 turned 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 iOS 2.0 and later.

Declared In
UISwitch.h

UITabBar Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 557) 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 555) property to access the current item.

Important: In iOS 3.0 and later, you should not attempt to use the methods and properties of this class to modify the tab bar when it is associated with a tab bar controller object. Modifying the tab bar in this way results in the throwing of an exception. Instead, any modifications to the tab bar or its items should occur through the tab bar controller interface. You may still directly modify a tab bar object that is not associated with a tab bar controller.

Tasks

Getting and Setting Properties

`delegate` (page 554) *property*
The tab bar's delegate object.

Configuring Items

`items` (page 554) *property*
The items displayed on the tab bar.

`selectedItem` (page 555) *property*
The currently selected item on the tab bar.

- `setItems:animated:` (page 557)
Sets the items on the tab bar, with or without animation.

Customizing Tab Bars

- `beginCustomizingItems:` (page 555)
Presents a modal view allowing the user to customize the tab bar by adding, removing, and rearranging items on the tab bar.
- `endCustomizingAnimated:` (page 556)
Dismisses the modal view used to modify items on the tab bar.
- `isCustomizing` (page 556)
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 Programming Language*.

delegate

The tab bar's delegate object.

```
@property(nonatomic, 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 iOS 2.0 and later.

Declared In
`UITabBar.h`

items

The items displayed on the tab bar.

```
@property(nonatomic, 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 557) method to animate changes.

The default value is `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`@property selectedItem` (page 555)
`- setItems:animated:` (page 557)

Declared In
`UITabBar.h`

selectedItem

The currently selected item on the tab bar.

```
@property(nonatomic, 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 iOS 2.0 and later.

See Also
`@property items` (page 554)
`- setItems:animated:` (page 557)

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 555) property is set to `nil`. Set the `delegate` (page 554) property to an object conforming to the `UITabBarDelegate` protocol to further modify this behavior.

Availability

Available in iOS 2.0 and later.

See Also

- `endCustomizingAnimated:` (page 556)
- `isCustomizing` (page 556)

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 iOS 2.0 and later.

See Also

- `beginCustomizingItems:` (page 555)
- `isCustomizing` (page 556)

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 iOS 2.0 and later.

See Also
- beginCustomizingItems: (page 555)
- endCustomizingAnimated: (page 556)

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 iOS 2.0 and later.

See Also
- @property items (page 554)
- @property selectedItem (page 555)

Declared In
UITabBar.h

UITabBarController Class Reference

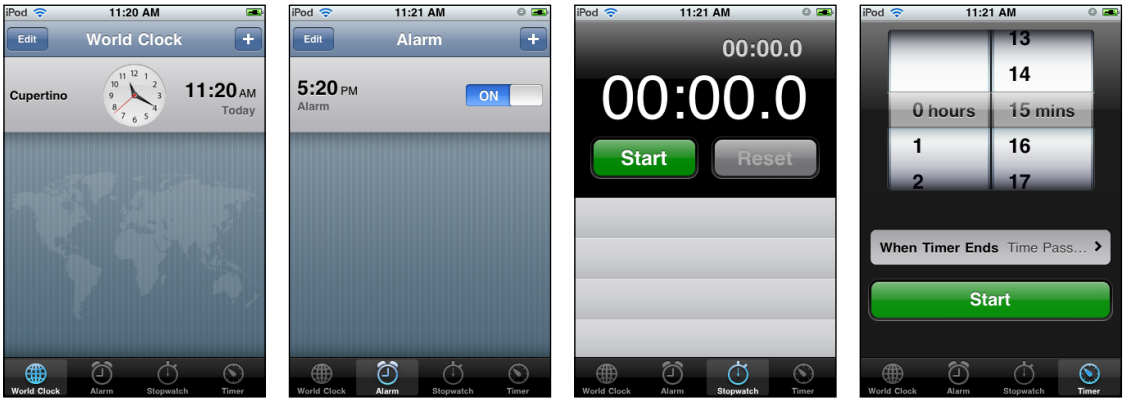
Inherits from	UIViewController : UIResponder : NSObject
Conforms to	NSCoding UITabBarDelegate NSCoding (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITabBarController.h
Companion guide	View Controller Programming Guide for iOS
Related sample code	MoviePlayer

Overview

The `UITabBarController` class implements a specialized view controller that manages a radio-style selection interface. This class is not intended for subclassing. Instead, you use instances of it as-is to present an interface that allows the user to choose between different modes of operation. This **tab bar interface** displays tabs at the bottom of the window for selecting between the different modes and for displaying the views for that mode.

Each tab of a tab bar controller interface is associated with a custom view controller. When the user selects a specific tab, the tab bar controller displays the root view of the corresponding view controller, replacing any previous views. (User taps always display the root view of the tab, regardless of which tab was previously selected. This is true even if the tab was already selected.) Because selecting a tab replaces the contents of the interface, the type of interface managed in each tab need not be similar in any way. In fact, tab bar interfaces are commonly used either to present different types of information or to present the same information using a completely different style of interface. Figure 57-1 shows the tab bar interface presented by the Clock application, each tab of which presents a type of time based information.

Figure 57-1 The tab bar interface in the Clock application



You should never access the tab bar view of a tab bar controller directly. To configure the tabs of a tab bar controller, you assign the view controllers that provide the root view for each tab to the `viewControllers` (page 565) property. The order in which you specify the view controllers determines the order in which they appear in the tab bar. When setting this property, you should also assign a value to the `selectedViewController` (page 564) property to indicate which view controller is selected initially. (You can also select view controllers by array index using the `selectedIndex` (page 564) property.) When you embed the tab bar controller's view (obtained using the inherited `view` (page 761) property) in your application window, the tab bar controller automatically selects that view controller and displays its contents, resizing them as needed to fit the tab bar interface.

Tab bar items are configured through their corresponding view controller. To associate a tab bar item with a view controller, create a new instance of the `UITabBarItem` class, configure it appropriately for the view controller, and assign it to the view controller's `tabBarItem` (page 760) property. If you do not provide a custom tab bar item for your view controller, the view controller creates a default item containing no image and the text from the view controller's `title` (page 760) property.

As the user interacts with a tab bar interface, the tab bar controller object sends notifications about the interactions to its delegate. The delegate can be any object you specify but must conform to the `UITabBarControllerDelegate` protocol. You can use the delegate to prevent specific tab bar items from being selected and to perform additional tasks when tabs are selected. You can also use the delegate to monitor changes to the tab bar that are made by the More navigation controller, which is described in more detail in "The More Navigation Controller" (page 561).

For more information about using tab bar controllers to build your user interface, see *View Controller Programming Guide for iOS*. For information about how to set up a tab bar controller using Interface Builder, see Nib Objects.

The Views of a Tab Bar Controller

Because the `UITabBarController` class inherits from the `UIViewController` class, tab bar controllers have their own view that is accessible through the `view` (page 761) property. When deploying a tab bar interface, you must install this view as the root of your window. Unlike other view controllers, a tab bar interface should never be installed as a child of another view controller.

The view for a tab bar controller is just a container for a tab bar view and the view containing your custom content. The tab bar view provides the selection controls for the user and consists of one or more tab bar items. Figure 57-2 shows how these views are assembled to present the overall tab bar interface. Although the items in the tab bar and toolbar views can change, the views that manage them do not. Only the custom content view changes to reflect the view controller for the currently selected tab.

Figure 57-2 The primary views of a tab bar controller

You can use navigation controllers or custom view controllers as the root view controller for a tab. If the root view controller is a navigation controller, the tab bar controller makes further adjustments to the size of the displayed navigation content so that it does not overlap the tab bar. Any views you display in a tab bar interface should therefore have their `autoresizingMask` (page 697) property set to resize the view appropriately under any conditions.

The More Navigation Controller

The tab bar has limited space for displaying your custom items. If you add six or more custom view controllers to a tab bar controller, the tab bar controller displays only the first four items plus the standard More item on the tab bar. Tapping the More item brings up a standard interface for selecting the remaining items.

The interface for the standard More item includes an Edit button that allows the user to reconfigure the tab bar. By default, the user is allowed to rearrange all items on the tab bar. If you do not want the user to modify some items, though, you can remove the appropriate view controllers from the array in the `customizableViewControllers` (page 562) property.

Tasks

Accessing the Tab Bar Controller Properties

`delegate` (page 563) *property*
The tab bar controller's delegate object.

`tabBar` (page 565) *property*
The tab bar view associated with this controller. (read-only)

Managing the View Controllers

`viewControllers` (page 565) *property*
An array of the root view controllers displayed by the tab bar interface.

`- setViewControllers:animated:` (page 566)
Sets the root view controllers of the tab bar controller.

`customizableViewControllers` (page 562) *property*
The subset of view controllers managed by this tab bar controller that can be customized.

`moreNavigationController` (page 563) *property*
The view controller that manages the More navigation interface. (read-only)

Managing the Selected Tab

`selectedViewController` (page 564) *property*
The view controller associated with the currently selected tab item.

`selectedIndex` (page 564) *property*
The index of the view controller associated with the currently selected tab item.

Properties

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

customizableViewControllers

The subset of view controllers managed by this tab bar controller that can be customized.

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

Discussion
This property controls which items in the tab bar can be rearranged by the user. When the user taps the More item on the tab bar view, a custom interface appears displaying any items that did not fit on the main tab bar. This interface also contains an Edit button that allows the user to rearrange the items. Only the items whose associated view controllers are in this array can be rearranged from this interface. If the array is empty or the value of this property is `nil`, the tab bar does not allow any items to be rearranged.

Changing the value of the `viewControllers` (page 565) property (either directly or using the `setViewControllers:animated:` (page 566) method) also changes the value of this property. When first assigned to the tab bar controller, all view controllers are customizable by default.

Availability
Available in iOS 2.0 and later.

See Also
`@property moreNavigationController` (page 563)

Declared In
`UITabBarController.h`

delegate

The tab bar controller's delegate object.

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

Discussion
You can use the delegate object to track changes to the items in the tab bar and to monitor the selection of tabs. The delegate object you provide should conform to the `UITabBarControllerDelegate` protocol. The default value for this property is `nil`.

Availability
Available in iOS 2.0 and later.

Declared In
`UITabBarController.h`

moreNavigationController

The view controller that manages the More navigation interface. (read-only)

```
@property(nonatomic, readonly) UINavigationController *moreNavigationController
```

Discussion
This property always contains a valid More navigation controller, even if a More button is not displayed on the screen. You can use the value of this property to select the More navigation controller in the tab bar interface or to compare it against the currently selected view controller.

Do not add the object stored in this property to your tab bar interface manually. The More controller is displayed automatically by the tab bar controller as it is needed. You must also not look for the More navigation controller in the array of view controllers stored in the `viewControllers` (page 565) property. The tab bar controller does not include the More navigation controller in that array of objects.

Availability
Available in iOS 2.0 and later.

See Also
`@property customizableViewControllers` (page 562)

Declared In
`UITabBarController.h`

selectedIndex

The index of the view controller associated with the currently selected tab item.

```
@property(nonatomic) NSUInteger selectedIndex
```

Discussion
This property nominally represents an index into the array of the `viewControllers` (page 565) property. However, if the selected view controller is currently the More navigation controller, this property contains the value `NSNotFound`. Setting this property changes the selected view controller to the one at the designated index in the `viewControllers` array. To select the More navigation controller itself, you must change the value of the `selectedViewController` (page 564) property instead.

In versions of iOS prior to version 3.0, this property reflects the index of the selected tab bar item only. Attempting to set this value to an index of a view controller that is not visible in the tab bar, but is instead managed by the More navigation controller, has no effect.

Availability
Available in iOS 2.0 and later.

See Also
`@property selectedViewController` (page 564)

Declared In
`UITabBarController.h`

selectedViewController

The view controller associated with the currently selected tab item.

```
@property(nonatomic, assign) UIViewController *selectedViewController
```

Discussion
This view controller is the one whose custom view is currently displayed by the tab bar interface. The specified view controller must be in the `viewControllers` (page 565) array. Assigning a new view controller to this property changes the currently displayed view and also selects an appropriate tab in the tab bar. Changing the view controller also updates the `selectedIndex` (page 564) property accordingly. The default value of this property is `nil`.

In iOS 3.0 and later, you can use this property to select any of the view controllers in the `viewControllers` property. This includes view controllers that are managed by the More navigation controller and whose tab bar items are not visible in the tab bar. You can also use it to select the More navigation controller itself, which is available from the `moreNavigationController` (page 563) property. Prior to iOS 3.0, you could select only the More navigation controller and the subset of view controllers whose tab bar item was visible. Attempting to set this property to a view controller whose tab bar item was not visible had no effect.

Availability
Available in iOS 2.0 and later.

See Also
`@property selectedIndex` (page 564)

Declared In
`UITabBarController.h`

tabBar

The tab bar view associated with this controller. (read-only)

```
@property(nonatomic,readonly) UITabBar *tabBar
```

Discussion
You should never attempt to manipulate the `UITabBar` object itself stored in this property. If you attempt to do so, the tab bar view throws an exception. To configure the items for your tab bar interface, you should instead assign one or more custom view controllers to the `viewControllers` (page 565) property. The tab bar collects the needed tab bar items from the view controllers you specify.

The tab bar view provided by this property is only for situations where you want to display an action sheet using the `showFromTabBar:` (page 87) method of the `UIActionSheet` class.

Availability
Available in iOS 3.0 and later.

Declared In
`UITabBarController.h`

viewControllers

An array of the root view controllers displayed by the tab bar interface.

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

Discussion
The default value of this property is `nil`. When configuring a tab bar controller, you can use this property to specify the content for each tab of the tab bar interface. The order of the view controllers in the array corresponds to the display order in the tab bar. Thus, the controller at index 0 corresponds to the left-most tab, the controller at index 1 the next tab to the right, and so on. If there are more view controllers than can fit in the tab bar, view controllers at the end of the array are managed by the More navigation controller, which is itself not included in this array.

If you change the value of this property at runtime, the tab bar controller removes all of the old view controllers before installing the new ones. The tab bar items for the new view controllers are displayed immediately and are not animated into position. When changing the view controllers, the tab bar controller remembers the view controller object that was previously selected and attempts to reselect it. If the selected view controller is no longer present, it attempts to select the view controller at the same index in the array as the previous selection. If that index is invalid, it selects the view controller at index 0.

Setting this property also sets the `customizableViewControllers` (page 562) property to the same set of view controllers.

Availability
Available in iOS 2.0 and later.

See Also
`- setViewControllers:animated:` (page 566)

Declared In
`UITabBarController.h`

Instance Methods

setViewControllers:animated:

Sets the root view controllers of the tab bar controller.

```
- (void)setViewControllers:(NSArray *)viewControllers  animated:(BOOL)animated
```

Parameters

viewControllers

The array of custom view controllers to display in the tab bar interface. The order of the view controllers in this array corresponds to the display order in the tab bar, with the controller at index 0 representing the left-most tab, the controller at index 1 the next tab to the right, and so on.

animated

If `YES`, the tab bar items for the view controllers are animated into position. If `NO`, changes to the tab bar items are reflected immediately.

Discussion
When you assign a new set of view controllers runtime, the tab bar controller removes all of the old view controllers before installing the new ones. When changing the view controllers, the tab bar controller remembers the view controller object that was previously selected and attempts to reselect it. If the selected view controller is no longer present, it attempts to select the view controller at the same index in the array as the previous selection. If that index is invalid, it selects the view controller at index 0.

This method also sets the value of the `customizableViewControllers` (page 562) property to the contents of the *viewControllers* parameter.

Availability
Available in iOS 2.0 and later.

See Also
`@property viewControllers` (page 565)

Declared In
`UITabBarController.h`

UITabBarItem Class Reference

Inherits from	UIBarItem : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 570) method to create one of the system items. Use the `initWithTitle:image:tag:` (page 571) method to create a custom item with the specified title and image.

Tasks

Initializing a Item

- `initWithTabBarSystemItem:tag:` (page 570)
  Creates and returns a new item containing the specified system item.
- `initWithTitle:image:tag:` (page 571)
  Creates and returns a new item using the specified properties.

Getting and Setting Properties

`badgeValue` (page 570) *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 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 iOS 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 571).

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`.

Discussion
This method returns a system-supplied tab bar item. The title and image properties of the returned item cannot be changed later.

Availability
Available in iOS 2.0 and later.

See Also
`- initWithTitle:image:tag:` (page 571)

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 iOS 2.0 and later.

See Also
`- initWithTabBarSystemItem:tag:` (page 570)

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 iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemFavorites`

The favorites system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemFeatured`

The featured system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemTopRated`

The top rated system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemRecents`

The recents system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemContacts`

The contacts system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemHistory`

The history system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemBookmarks`

The bookmarks system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemSearch`

The search system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemDownloads`

The downloads system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemMostRecent`

The most recent system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

`UITabBarSystemItemMostViewed`

The most viewed system item. 

Available in iOS 2.0 and later.

Declared in `UITabBarItem.h`.

Discussion
The title and image of system tab bar items cannot be changed.

Availability
Available in iOS 2.0 and later.

Declared In
`UITabBarItem.h`

UITableView Class Reference

Inherits from	UIScrollView : UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIScrollView) NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITableView.h
Companion guide	Table View Programming Guide for iOS
Related sample code	AddMusic BonjourWeb CryptoExercise GKRocket WiTap

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 zero 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 600) and `UITableViewStyleGrouped` (page 600). 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 42) property) and section index (`section` (page 42) property), and to construct an index path from a given row index and section index (`indexPathForRow:inSection:` (page 42) 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 an 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 599) 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 615) of each associated `UITableViewCell`. Clicking on the insertion or deletion control causes the data source to receive a `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) message. You commit a deletion or insertion by calling `deleteRowsAtIndexPaths:withRowAnimation:` (page 586) or `insertRowsAtIndexPaths:withRowAnimation:` (page 591), as appropriate. Also in editing mode, if a table-view cell has its `showsReorderControl` (page 621) property set to `YES`, the data source receives a `tableView:moveRowAtIndexPath:toIndexPath:` (page 930) message. The data source can selectively remove the reordering control for cells by implementing `tableView:canMoveRowAtIndexPath:` (page 928).

`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 iOS* explains how.

`UITableView` overrides the `layoutSubviews` (page 732) method of `UIView` so that it calls `reloadData` (page 595) only when you create a new instance of `UITableView` or when you assign a new data source. Reloading the table view clears current state, including the current selection. However, if you explicitly call `reloadData`, it clears this state and any subsequent direct or indirect call to `layoutSubviews` does not trigger a reload.

Tasks

Initializing a UITableView Object

`- initWithFrame:style:` (page 591)
Initializes and returns a table view object having the given frame and style.

Configuring a Table View

- `dequeueReusableCellWithIdentifier:` (page 587)
  Returns a reusable table-view cell object located by its identifier.

`style` (page 584) *property*
Returns the style of the receiver. (read-only)

- `numberOfRowsInSection:` (page 593)
  Returns the number of rows (table cells) in a specified section.
- `numberOfSections` (page 593)
  Returns the number of sections for the receiver.

`rowHeight` (page 581) *property*
The height of each row (table cell) in the receiver.

`separatorStyle` (page 583) *property*
The style for table cells used as separators.

`separatorColor` (page 583) *property*
The color of separator rows in the table view.

`backgroundView` (page 580) *property*
The background view of the table view.

`tableHeaderView` (page 584) *property*
Returns an accessory view that is displayed above the table.

`tableFooterView` (page 584) *property*
Returns an accessory view that is displayed below the table.

`sectionHeaderHeight` (page 582) *property*
The height of section headers in the receiving table view.

`sectionFooterHeight` (page 582) *property*
The height of section footers in the receiving table view.

`sectionIndexMinimumDisplayRowCount` (page 583) *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 585)
  Returns the table cell at the specified index path.
- `indexPathForCell:` (page 589)
  Returns an index path representing the row and section of a given table-view cell.
- `indexPathForRowAtPoint:` (page 589)
  Returns an index path identifying the row and section at the given point.
- `indexPathsForRowsInRect:` (page 590)
  An array of index paths each representing a row enclosed by a given rectangle.
- `visibleCells` (page 600)
  Returns the table cells that are visible in the receiver.
- `indexPathsForVisibleRows` (page 590)
  Returns an array of index paths each identifying a visible row in the receiver.

Scrolling the Table View

- `scrollToRowAtIndexPath:atScrollPosition:animated:` (page 598)
  Scrolls the receiver until a row identified by index path is at a particular location on the screen.
- `scrollToNearestSelectedRowAtScrollPosition:animated:` (page 597)
  Scrolls the table view so that the selected row nearest to a specified position in the table view is at that position.

Managing Selections

- `indexPathForSelectedRow` (page 589)
  Returns an index path identifying the row and section of the selected row.
- `selectRowAtIndexPath:animated:scrollPosition:` (page 598)
  Selects a row in the receiver identified by index path, optionally scrolling the row to a location in the receiver.
- `deselectRowAtIndexPath:animated:` (page 588)
  Deselects a given row identified by index path, with an option to animate the deselection.

`allowsSelection` (page 579) *property*
A Boolean value that determines whether users can select cells

`allowsSelectionDuringEditing` (page 580) *property*
A Boolean value that determines whether users can select cells while the receiver is in editing mode.

Inserting and Deleting Cells

- `beginUpdates` (page 585)
  Begin a series of method calls that insert, delete, select, or delete rows and sections of the receiver.
- `endUpdates` (page 588)
  Conclude a series of method calls that insert, delete, select, or reload rows and sections of the receiver.
- `insertRowsAtIndexPaths:withRowAnimation:` (page 591)
  Inserts rows in the receiver at the locations identified by an array of index paths, with an option to animate the insertion.
- `deleteRowsAtIndexPaths:withRowAnimation:` (page 586)
  Deletes the rows specified by an array of index paths, with an option to animate the deletion.
- `insertSections:withRowAnimation:` (page 592)
  Inserts one or more sections in the receiver, with an option to animate the insertion.
- `deleteSections:withRowAnimation:` (page 586)
  Deletes one or more sections in the receiver, with an option to animate the deletion.

Managing the Editing of Table Cells

`editing` (page 581) *property*
A Boolean value that determines whether the receiver is in editing mode.

- `setEditing:animated:` (page 599)
  Toggles the receiver into and out of editing mode.

Reloading the Table View

- `reloadData` (page 595)
  Reloads the rows and sections of the receiver.
- `reloadRowsAtIndexPaths:withRowAnimation:` (page 595)
  Reloads the specified rows using a certain animation effect.
- `reloadSections:withRowAnimation:` (page 597)
  Reloads the specified sections using a given animation effect.
- `reloadSectionIndexTitles` (page 596)
  Reloads the items in the index bar along the right side of the table view.

Accessing Drawing Areas of the Table View

- `rectForSection:` (page 595)
  Returns the drawing area for a specified section of the receiver.
- `rectForRowAtIndexPath:` (page 594)
  Returns the drawing area for a row identified by index path.
- `rectForFooterInSection:` (page 593)
  Returns the drawing area for the footer of the specified section.
- `rectForHeaderInSection:` (page 594)
  Returns the drawing area for the header of the specified section.

Managing the Delegate and the Data Source

`dataSource` (page 580) *property*
The object that acts as the data source of the receiving table view.

`delegate` (page 581) *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 Programming Language*.

allowsSelection

A Boolean value that determines whether users can select cells

```
@property(nonatomic) BOOL allowsSelection
```

Discussion
If the value of this property is `YES` (the default), users can select rows. If you set it to `NO`, they cannot select rows. Setting this property affects cell selection only when the table view is not in editing mode. If you want to restrict selection of cells in editing mode, use `allowsSelectionDuringEditing` (page 580).

Availability
Available in iOS 3.0 and later.

Declared In
`UITableView.h`

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`. If you want to restrict selection of cells regardless of mode, use `allowsSelection` (page 579).

Availability
Available in iOS 2.0 and later.

Declared In
`UITableView.h`

backgroundView

The background view of the table view.

```
@property(nonatomic, readwrite, retain) UIView *backgroundView
```

Discussion
A table view's background view is automatically resized to match the size of the table view. This view is placed as a subview of the table view behind all cells, header views, and footer views.

Availability
Available in iOS 3.2 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 iOS 2.0 and later.

See Also
`@property delegate` (page 581)

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 iOS 2.0 and later.

See Also
`@property dataSource` (page 580)

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 929). The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`- setEditing:animated:` (page 599)

Related Sample Code
BonjourWeb

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 941) method. If you do not explicitly set the row height, `UITableView` sets it to a standard value.

There are performance implications to using `tableView:heightForRowAtIndexPath:` instead of `rowHeight`. Every time a table view is displayed, it calls `tableView:heightForRowAtIndexPath:` on the delegate for each of its rows, which can result in a significant performance problem with table views having a large number of rows (approximately 1000 or more).

Availability
Available in iOS 2.0 and later.

Related Sample Code
MultipleDetailViews

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 940) method.

Availability
Available in iOS 2.0 and later.

See Also
`@property tableFooterView` (page 584)

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 if delegate the doesn't implement the `tableView:heightForHeaderInSection:` (page 941) method.

Availability
Available in iOS 2.0 and later.

See Also
`@property tableHeaderView` (page 584)

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 600) style. The default value is `NSIntegerMax`.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property separatorStyle` (page 583)

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 928).

Availability
Available in iOS 2.0 and later.

See Also
`@property separatorColor` (page 583)

Declared In
`UITableView.h`

style

Returns the style of the receiver. (read-only)

```
@property(nonatomic, readonly) UITableViewStyle style
```

Discussion
See "Table View Style" (page 600) for descriptions of the constants used to specify table-view style.

Availability
Available in iOS 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 footer.

Availability
Available in iOS 2.0 and later.

See Also
`@property sectionFooterHeight` (page 582)

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 iOS 2.0 and later.

See Also
`@property sectionHeaderHeight` (page 582)

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 585) and `indexPathsForVisibleRows` (page 590)) to be animated simultaneously. This group of methods must conclude with an invocation of `endUpdates` (page 588). These method pairs can 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. You should not call `reloadData` (page 595) within the group; if you call this method within the group, you will need to perform any animations yourself.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`- indexPathForCell:` (page 589)

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

A constant that indicates how the deletion is to be animated, for example, fade out or slide out from the bottom. See "Table Cell Insertion and Deletion Animation" (page 601) for descriptions of these constants.

Discussion
Note the behavior of this method when it is called in an animation block defined by the lbeginUpdates (page 585) and endUpdates (page 588) methods. UITableView defers any insertions of rows or sections until after it has handled the deletions of rows or sections. This happens regardless of ordering of the insertion and deletion method calls. This is unlike inserting or removing an item in a mutable array, where the operation can affect the array index used for the successive insertion or removal operation. For more on this subject, see "Batch Insertion and Deletion of Rows and Sections" in *Table View Programming Guide for iOS*.

Availability
Available in iOS 2.0 and later.

See Also
- insertRowsAtIndexPaths:withRowAnimation: (page 591)
- reloadRowsAtIndexPaths:withRowAnimation: (page 595)

Related Sample Code
BonjourWeb

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.

Discussion
Note the behavior of this method when it is called in an animation block defined by the `beginUpdates` (page 585) and `endUpdates` (page 588) methods. `UITableView` defers any insertions of rows or sections until after it has handled the deletions of rows or sections. This happens regardless of ordering of the insertion and deletion method calls. This is unlike inserting or removing an item in a mutable array, where the operation can affect the array index used for the successive insertion or removal operation. For more on this subject, see "Batch Insertion and Deletion of Rows and Sections" in *Table View Programming Guide for iOS*.

Availability
Available in iOS 2.0 and later.

See Also
- `insertSections:withRowAnimation:` (page 592)
- `reloadSections:withRowAnimation:` (page 597)

Related Sample Code
BonjourWeb

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 data source should generally reuse `UITableViewCell` objects when it assigns cells to rows in its `tableView:cellForRowAtIndexPath:` (page 928) 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 624) method of `UITableViewCell`). The data source 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 iOS 2.0 and later.

Related Sample Code
AddMusic
GKRocket
ToolbarSearch

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 947) or tableView:didSelectRowAtIndexPath: (page 939) message, nor will it send UITableViewSelectionDidChangeNotification (page 603) notifications to observers.

Calling this method does not cause any scrolling to the deselected row.

Availability
Available in iOS 2.0 and later.

See Also
- indexPathForSelectedRow (page 589)

Related Sample Code
AddMusic

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 585) 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 (page 585) and endUpdates can 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 iOS 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 iOS 2.0 and later.

See Also

`- cellForRowAtIndexPath:` (page 585)

Declared In

`UITableView.h`

indexPathForRowAtPoint:

Returns an index path identifying the row and section at the given point.

```
- (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 iOS 2.0 and later.

See Also

`- indexPathForCell:` (page 589)

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 iOS 2.0 and later.

See Also
- `selectRowAtIndexPath:animated:scrollPosition:` (page 598)

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 iOS 2.0 and later.

See Also
- `indexPathForRowAtPoint:` (page 589)

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 iOS 2.0 and later.

See Also
- `visibleCells` (page 600)

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 600) 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 729), the `UITableViewStylePlain` (page 600) style is used as a default.

Availability

Available in iOS 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 601) for descriptions of the constants.

Discussion

`UITableView` calls the relevant delegate and data source methods immediately afterwards to get the cells and other content for visible cells.

Note the behavior of this method when it is called in an animation block defined by the l`beginUpdates` (page 585) and `endUpdates` (page 588) methods. `UITableView` defers any insertions of rows or sections until after it has handled the deletions of rows or sections. This happens regardless of ordering of the insertion and

deletion method calls. This is unlike inserting or removing an item in a mutable array, where the operation can affect the array index used for the successive insertion or removal operation. For more on this subject, see "Batch Insertion and Deletion of Rows and Sections" in *Table View Programming Guide for iOS*.

Availability
Available in iOS 2.0 and later.

See Also
- `insertSections:withRowAnimation:` (page 592)
- `deleteRowsAtIndexPaths:withRowAnimation:` (page 586)
- `reloadRowsAtIndexPaths:withRowAnimation:` (page 595)

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

A constant that indicates how the insertion is to be animated, for example, fade in or slide in from the left. See "Table Cell Insertion and Deletion Animation" (page 601) for descriptions of these constants.

Discussion
`UITableView` calls the relevant delegate and data source methods immediately afterwards to get the cells and other content for visible cells.

Note the behavior of this method when it is called in an animation block defined by the l`beginUpdates` (page 585) and `endUpdates` (page 588) methods. `UITableView` defers any insertions of rows or sections until after it has handled the deletions of rows or sections. This happens regardless of ordering of the insertion and deletion method calls. This is unlike inserting or removing an item in a mutable array, where the operation can affect the array index used for the successive insertion or removal operation. For more on this subject, see "Batch Insertion and Deletion of Rows and Sections" in *Table View Programming Guide for iOS*.

Availability
Available in iOS 2.0 and later.

See Also
- `insertRowsAtIndexPaths:withRowAnimation:` (page 591)
- `deleteSections:withRowAnimation:` (page 586)
- `reloadSections:withRowAnimation:` (page 597)

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
The number of rows in the section.

Discussion
`UITableView` gets the value returned by this method from its data source and caches it.

Availability
Available in iOS 2.0 and later.

See Also
`- numberOfSections` (page 593)

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 iOS 2.0 and later.

See Also
`- numberOfRowsInSection:` (page 593)

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 iOS 2.0 and later.

Declared In

UITableView.h

rectForHeaderInSection:

Returns the drawing 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 iOS 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 `CGRectZero` if *indexPath* is invalid.

Availability

Available in iOS 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 iOS 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 585) and `endUpdates` (page 588)

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

Declared In
UITableView.h

reloadRowsAtIndexPaths:withRowAnimation:

Reloads the specified rows using a certain animation effect.

```
- (void)reloadRowsAtIndexPaths:(NSArray *)indexPaths
    withRowAnimation:(UITableViewRowAnimation)animation
```

Parameters

indexPaths

An array of `NSIndexPath` objects identifying the rows to reload.

animation

A constant that indicates how the reloading is to be animated, for example, fade out or slide out from the bottom. See "Table Cell Insertion and Deletion Animation" (page 601) for descriptions of these constants.

The animation constant affects the direction in which both the old and the new rows slide. For example, if the animation constant is `UITableViewRowAnimationRight` (page 602), the old rows slide out to the right and the new cells slide in from the right.

Discussion
Reloading a row causes the table view to ask its data source for a new cell for that row. The table animates that new cell in as it animates the old row out. Call this method if you want to alert the user that the value of a cell is changing. If, however, notifying the user is not important—that is, you just want to change the value that a cell is displaying—you can get the cell for a particular row and set its new value.

When this method is called in an animation block defined by the `beginUpdates` (page 585) and `endUpdates` (page 588) methods, it behaves similarly to `deleteRowsAtIndexPaths:withRowAnimation:` (page 586). The indexes that `UITableView` passes to the method are specified in the state of the table view prior to any updates. This happens regardless of ordering of the insertion, deletion, and reloading method calls within the animation block.

Availability
Available in iOS 3.0 and later.

See Also
`- insertRowsAtIndexPaths:withRowAnimation:` (page 591)

Related Sample Code
BonjourWeb
WiTap

Declared In
`UITableView.h`

reloadSectionIndexTitles

Reloads the items in the index bar along the right side of the table view.

```
- (void)reloadSectionIndexTitles
```

Discussion
This method gives you a way to update the section index after inserting or deleting sections without having to reload the whole table.

Availability
Available in iOS 3.0 and later.

See Also
`- sectionIndexTitlesForTableView:` (page 927) (`UITableViewDataSource`)

Declared In
`UITableView.h`

reloadSections:withRowAnimation:

Reloads the specified sections using a given animation effect.

```
- (void)reloadSections:(NSIndexSet
    *)sectionswithRowAnimation:(UITableViewRowAnimation)animation
```

Parameters

sections

An index set identifying the sections to reload.

animation

A constant that indicates how the reloading is to be animated, for example, fade out or slide out from the bottom. See "Table Cell Insertion and Deletion Animation" (page 601) for descriptions of these constants.

The animation constant affects the direction in which both the old and the new section rows slide. For example, if the animation constant is `UITableViewRowAnimationRight` (page 602), the old rows slide out to the right and the new cells slide in from the right.

Discussion

Calling this method causes the table view to ask its data source for new cells for the specified sections. The table view animates the insertion of new cells in as it animates the old cells out. Call this method if you want to alert the user that the values of the designated sections are changing. If, however, you just want to change values in cells of the specified sections without alerting the user, you can get those cells and directly set their new values.

When this method is called in an animation block defined by the `beginUpdates` (page 585) and `endUpdates` (page 588) methods, it behaves similarly to `deleteSections:withRowAnimation:` (page 586). The indexes that `UITableView` passes to the method are specified in the state of the table view prior to any updates. This happens regardless of ordering of the insertion, deletion, and reloading method calls within the animation block.

Availability

Available in iOS 3.0 and later.

See Also

`- insertSections:withRowAnimation:` (page 592)

Declared In

`UITableView.h`

scrollToNearestSelectedRowAtScrollPosition:animated:

Scrolls the table view so that the selected row nearest to a specified position in the table view is at 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 601) 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 iOS 2.0 and later.

See Also
- `scrollToRowAtIndexPath:atScrollPosition:animated:` (page 598)

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 601) 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 `scrollViewDidScroll:` (page 894) message, as is normal for programmatically-invoked user interface operations.

Availability
Available in iOS 2.0 and later.

See Also
- `scrollToNearestSelectedRowAtScrollPosition:animated:` (page 597)

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 601) a descriptions of valid constants.

Discussion

Calling this method does not cause the delegate to receive a `tableView:willSelectRowAtIndexPath:` (page 947) or `tableView:didSelectRowAtIndexPath:` (page 939) message, nor will it send `UITableViewSelectionDidChangeNotification` (page 603) notifications to observers.

Availability

Available in iOS 2.0 and later.

See Also

`- indexPathForSelectedRow` (page 589)

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 626) 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 927).

Availability

Available in iOS 2.0 and later.

See Also
`@property editing` (page 581)

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 iOS 2.0 and later.

See Also
`- indexPathsForVisibleRows` (page 590)

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 iOS 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 iOS 2.0 and later.

Declared in `UITableView.h`.

Discussion
You set the table style when you initialize the table view (see `initWithFrame:style:` (page 591)). 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 scrolls the row of interest to be fully visible with a minimum of movement. If the row is already fully visible, no scrolling occurs. For example, if the row is above the visible area, the behavior is identical to that specified by UITableViewScrollPositionTop (page 601). This is the default.

  Available in iOS 2.0 and later.

  Declared in UITableView.h.

UITableViewScrollPositionTop
: The table view scrolls the row of interest to the top of the visible table view.

  Available in iOS 2.0 and later.

  Declared in UITableView.h.

UITableViewScrollPositionMiddle
: The table view scrolls the row of interest to the middle of the visible table view.

  Available in iOS 2.0 and later.

  Declared in UITableView.h.

UITableViewScrollPositionBottom
: The table view scrolls the row of interest to the bottom of the visible table view.

  Available in iOS 2.0 and later.

  Declared in UITableView.h.

Discussion
You set the scroll position through a parameter of the selectRowAtIndexPath:animated:scrollPosition: (page 598), scrollToNearestSelectedRowAtScrollPosition:animated: (page 597), cellForRowAtIndexPath: (page 585), and indexPathForSelectedRow (page 589) methods.

Declared In
UITableView.h

Table Cell Insertion and Deletion Animation

The type of animation when rows are inserted or deleted.

```
typedef enum {
    UITableViewRowAnimationFade,
    UITableViewRowAnimationRight,
    UITableViewRowAnimationLeft,
    UITableViewRowAnimationTop,
    UITableViewRowAnimationBottom,
    UITableViewRowAnimationNone,
    UITableViewRowAnimationMiddle
} UITableViewRowAnimation;
```

Constants

`UITableViewRowAnimationFade`
The inserted or deleted row or rows fades into or out of the table view.

Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `UITableView.h`.

`UITableViewRowAnimationBottom`
The inserted row or rows slides in from the bottom; the deleted row or rows slides out toward the bottom.

Available in iOS 2.0 and later.

Declared in `UITableView.h`.

`UITableViewRowAnimationNone`
No animation is performed. The new cell value appears as if the cell had just been reloaded.

Available in iOS 3.0 and later.

Declared in `UITableView.h`.

`UITableViewRowAnimationMiddle`
The table view attempts to keep the old and new cells centered in the space they did or will occupy.
Available in iPhone 3.2.

Available in iOS 3.2 and later.

Declared in `UITableView.h`.

Discussion

You specify one of these constants as a parameter of the `insertRowsAtIndexPaths:withRowAnimation:` (page 591), `insertSections:withRowAnimation:` (page 592),

`deleteRowsAtIndexPaths:withRowAnimation:` (page 586),`deleteSections:withRowAnimation:` (page 586), `reloadRowsAtIndexPaths:withRowAnimation:` (page 595), and `reloadSections:withRowAnimation:` (page 597) methods.

Declared In
`UITableView.h`

Section Index Icons

Requests icon to be shown in the section index of a table view.

```
UIKIT_EXTERN NSString *const UITableViewIndexSearch;
```

Constants
`UITableViewIndexSearch`
If the data source includes this constant string in the array of strings it returns in `sectionIndexTitlesForTableView:` (page 927), the section index displays a magnifying glass icon at the corresponding index location. This location should generally be the first title in the index.

Available in iOS 3.0 and later.

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 iOS 2.0 and later.

Declared In
`UITableView.h`

UITableViewCell Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITableViewCell.h
Companion guide	Table View Programming Guide for iOS
Related sample code	AddMusic BonjourWeb CryptoExercise GKRocket WiTap

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 cell selection, highlighted state, editing state and controls, accessory views, reordering controls, cell background, and content indentation. The class additionally includes properties for setting and managing cell content, specifically text and images.

For iOS 3.0, `UITableViewCell` includes two major improvements:

- Predefined cell styles that position elements of the cell (labels and images) in certain locations and with certain attributes. See "Cell Styles" (page 629) for descriptions of the constants that apply to these styles.
- Properties for accessing the content of the cell. These properties include `textLabel` (page 623), `detailTextLabel` (page 612), and `imageView` (page 617). Once you get the associated `UILabel` and `UIImageView` objects, you can set their attributes, such as text color, font, image, highlighted image, and so on.

You have two ways of extending the standard `UITableViewCell` object beyond the given styles. 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 612) 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 iOS* for details.

Note: If you want to change the background color of a cell (by setting the background color of a cell via the `backgroundColor` property declared by `UIView`) you must do it in the `tableView:willDisplayCell:forRowAtIndexPath:` (page 946) method of the delegate and not in `tableView:cellForRowAtIndexPath:` (page 928) of the data source. Changes to the background colors of cells in a group-style table view has an effect in iOS 3.0 that is different than previous versions of the operating system. It now affects the area inside the rounded rectangle instead of the area outside of it.

Tasks

Initializing a UITableViewCell Object

- `initWithStyle:reuseIdentifier:` (page 625)
  Initializes a table cell with a style and a reuse identifier and returns it to the caller.
- `initWithFrame:reuseIdentifier:` (page 624)
  Initializes and returns a table cell object. (Deprecated. Use `initWithStyle:reuseIdentifier:` (page 625) instead.)

Reusing Cells

`reuseIdentifier` (page 618) *property*
A string used to identify a cell that is reusable. (read-only)

- `prepareForReuse` (page 626)
  Prepares a reusable cell for reuse by the table view's delegate.

Managing Text as Cell Content

`textLabel` (page 623) *property*
Returns the label used for the main textual content of the table cell. (read-only)

`detailTextLabel` (page 612) *property*
Returns the secondary label of the table cell if one exists. (read-only)

`text` (page 622) *property*
The text of the cell. (Deprecated. Use the `textLabel` (page 623) and `detailTextLabel` (page 612) properties instead.)

`textAlignment` (page 622) *property*
A constant that specifies the alignment of text in the cell. (Deprecated. Instead set the text alignment of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

`textColor` (page 623) *property*
The color of the title text. (Deprecated. Instead set the text color attribute of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

`selectedTextColor` (page 620) *property*
The color of the title text when the cell is selected. (Deprecated. Instead set the `highlightedTextColor` (page 345) property of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

`font` (page 615) *property* Deprecated in iOS 3.0
The font of the title. (Deprecated. Instead, set the fonts of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

`lineBreakMode` (page 618) *property* Deprecated in iOS 3.0
The mode for for wrapping and truncating text in the cell. (Deprecated. Instead set the line-break mode attribute of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

Managing Images as Cell Content

`imageView` (page 617) *property*
Returns the image view of the table cell. (read-only)

`selectedImage` (page 619) *property*
The image to use a cell content when the cell is selected. (Deprecated. Instead use the `imageView` (page 617) property to obtain the `UIImageView` object and then get or set its `highlightedImage` property.)

`image` (page 616) *property* Deprecated in iOS 3.0
The image to use as content for the cell. (Deprecated. Instead use the `imageView` (page 617) property to get `UIImageView` object and then get or set the encapsulated image.)

Accessing Views of the Cell Object

`contentView` (page 612) *property*
Returns the content view of the cell object. (read-only)

`backgroundView` (page 611) *property*
The view used as the background of the cell.

`selectedBackgroundView` (page 619) *property*
The view used as the background of the cell when it is selected.

Managing Accessory Views

`accessoryType` (page 610) *property*
The type of standard accessory view the cell should use (normal state).

`accessoryView` (page 611) *property*
A view that is used, typically as a control, on the right side of the cell (normal state).

`editingAccessoryType` (page 613) *property*
The type of standard accessory view the cell should use in the table view's editing state.

`editingAccessoryView` (page 614) *property*
A view that is used typically as a control on the right side of the cell when it is in editing mode.

`hidesAccessoryWhenEditing` (page 615) *property*
A Boolean value that determines whether the accessory view is hidden when the cell is being edited. (Deprecated. Use the `editingAccessoryType` (page 613) and `editingAccessoryView` (page 614) properties instead.)

Managing Cell Selection and Highlighting

`selected` (page 618) *property*
A Boolean value that indicates whether the cell is selected.

`selectionStyle` (page 620) *property*
The style of selection for a cell.

- `setSelected:animated:` (page 627)
Sets the selected state of the cell, optionally animating the transition between states.

`highlighted` (page 616) *property*
A Boolean value that indicates whether the cell is highlighted.

- `setHighlighted:animated:` (page 627)
Sets the highlighted state of the cell, optionally animating the transition between states.

Editing the Cell

`editing` (page 613) *property*
A Boolean value that indicates whether the cell is in an editable state.

- `setEditing:animated:` (page 626)
Toggles the receiver into and out of editing mode.

`editingStyle` (page 615) *property*
The editing style of the cell. (read-only)

`showingDeleteConfirmation` (page 621) *property*
Returns whether the cell is currently showing the delete-confirmation button. (read-only)

`showsReorderControl` (page 621) *property*
A Boolean value that determines whether the cell shows the reordering control.

Adjusting to State Transitions

- `willTransitionToState:` (page 628)
Called on the cell just before it transitions between cell states.

- `didTransitionToState:` (page 623)
Called on the cell just after it transitions between cell states.

Managing Content Indentation

`indentationLevel` (page 617) *property*
Adjusts the indentation level of a cell whose content is indented.

`indentationWidth` (page 617) *property*
The width for each level of indentation of a cell's content.

`shouldIndentWhileEditing` (page 620) *property*
A Boolean value that controls whether the cell background is indented when the table view is in editing mode.

Managing Targets and Actions

These properties are deprecated as of iOS 3.0. Instead, use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) method of the `UITableViewDataSource` protocol or the `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) method of the `UITableViewDelegate` protocol.

`target` (page 621) *property*
The target object to receive action messages. (Deprecated. Instead use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) or `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) for handling taps on cells.)

`editAction` (page 613) *property*
The selector defining the action message to invoke when users tap the insert or delete button. (Deprecated. Instead use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) or `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) for handling taps on cells.)

`accessoryAction` (page 609) *property*
The selector defining the action message to invoke when users tap the accessory view. (Deprecated. Instead use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) or `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) for handling taps on cells.)

Properties

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

accessoryAction

The selector defining the action message to invoke when users tap the accessory view. (Deprecated in iOS 3.0. Instead use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) or `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) for handling taps on cells.)

```
@property(nonatomic) SEL accessoryAction
```

Discussion
If you specify a selector for the accessory action, a message is sent only if the accessory view is a detail disclosure button—that is, the cell's `accessoryType` (page 610) property is assigned a value of`UITableViewCellAccessoryDetailDisclosureButton` (page 632). If the value of this property is `NULL`, no action message is sent.

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. If the accessory view inherits from `UIControl`, you may set a target and action through the `addTarget:action:forControlEvents:` (page 218) method. See `accessoryView` (page 611) for more information.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property target` (page 621)
`@property editAction` (page 613)

Declared In
`UITableViewCell.h`

accessoryType

The type of standard accessory view the cell should use (normal state).

```
@property(nonatomic) UITableViewCellAccessoryType accessoryType
```

Discussion
The accessory view appears in the the right side of the cell in the table view's normal (default) state. The standard accessory views include the disclosure chevron; for a description of valid `accessoryType` constants, see "Cell Accessory Type" (page 631). The default is `UITableViewCellAccessoryNone` (page 631). If a custom accessory view is set through the `accessoryView` (page 611) property, the value of this property is ignored. If the cell is enabled and the accessory type is `UITableViewCellAccessoryDetailDisclosureButton` (page 632), the accessory view tracks touches and, when tapped, sends the data-source object a `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) message.

The accessory-type image cross-fades between normal and editing states if it set for both states; use the `editingAccessoryType` (page 613) property to set the accessory type for the cell during editing mode. If this property is not set for both states, the cell is animated to slide in or out, as necessary.

Availability
Available in iOS 2.0 and later.

See Also
`@property editingAccessoryType` (page 613)
`@property editingAccessoryView` (page 614)
`- willTransitionToState:` (page 628)
`- didTransitionToState:` (page 623)

Related Sample Code
BonjourWeb
CryptoExercise
WiTap

Declared In
`UITableViewCell.h`

accessoryView

A view that is used, typically as a control, on the right side of the cell (normal state).

```
@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 in the table view's normal (default) state; it ignores the value of the `accessoryType` (page 610) 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.

The accessory view cross-fades between normal and editing states if it set for both states; use the `editingAccessoryView` (page 614) property to set the accessory view for the cell during editing mode. If this property is not set for both states, the cell is animated to slide in or out, as necessary.

Availability
Available in iOS 2.0 and later.

See Also
`@property editingAccessoryType` (page 613)
`@property editingAccessoryView` (page 614)
`- willTransitionToState:` (page 628)
`- didTransitionToState:` (page 623)

Related Sample Code
BonjourWeb
WiTap

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 600)) and non-`nil` for grouped-style tables `UITableViewStyleGrouped` (page 600)). `UITableViewCell` adds the background view as a subview behind all other views and uses its current frame location.

Availability
Available in iOS 2.0 and later.

See Also
`@property contentView` (page 612)
`@property selectedBackgroundView` (page 619)

Declared In
`UITableViewCell.h`

contentView

Returns the content view 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 iOS 2.0 and later.

See Also
`@property backgroundView` (page 611)

Declared In
`UITableViewCell.h`

detailTextLabel

Returns the secondary label of the table cell if one exists. (read-only)

```
@property(nonatomic, readonly, retain) UILabel *detailTextLabel
```

Discussion
Holds the secondary (or detail) label of the cell. `UITableViewCell` adds an appropriate label when you create the cell in a style that supports secondary labels. If the style doesn't support detail labels, `nil` is returned. See "Cell Styles" (page 629) for descriptions of the main label in currently defined cell styles.

Availability
Available in iOS 3.0 and later.

See Also
`@property textLabel` (page 623)
`- initWithStyle:reuseIdentifier:` (page 625)

Declared In
`UITableViewCell.h`

editAction

The selector defining the action message to invoke when users tap the insert or delete button. (Deprecated in iOS 3.0. Instead use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) or `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) for handling taps on cells.)

```
@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 619) constant applied to the cell via the `editingStyle` (page 615) 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 iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property target` (page 621)
`@property accessoryAction` (page 609)

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 615) and `showsReorderControl` (page 621) to specify these controls for the cell.

Availability
Available in iOS 2.0 and later.

Declared In
`UITableViewCell.h`

editingAccessoryType

The type of standard accessory view the cell should use in the table view's editing state.

```
@property(nonatomic) UITableViewCellAccessoryType editingAccessoryType
```

Discussion
The accessory view appears in the the right side of the cell when the table view is in editing mode. The standard accessory views include the disclosure chevron; for a description of valid constants, see "Cell Accessory Type" (page 631). The default is `UITableViewCellAccessoryNone` (page 631). If a custom accessory

view for editing mode is set through the `editingAccessoryView` (page 614) property, the value of this property is ignored. If the cell is enabled and the accessory type is `UITableViewCellAccessoryDetailDisclosureButton` (page 632), the accessory view tracks touches and, when tapped, sends the delegate object a `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) message.

The accessory type cross-fades between normal and editing states if it set for both states; use the `accessoryType` (page 610) property to set the accessory view for the cell during the table view's normal state. If this property is not set for both states, the cell is animated to slide or out, as necessary.

Availability
Available in iOS 3.0 and later.

See Also
`@property accessoryType` (page 610)
`@property accessoryView` (page 611)
`- willTransitionToState:` (page 628)
`- didTransitionToState:` (page 623)

Declared In
`UITableViewCell.h`

editingAccessoryView

A view that is used typically as a control on the right side of the cell when it is in editing mode.

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

Discussion
If the value of this property is not `nil`, the `UITableViewCell` class uses the given view for the accessory view in the table view's editing state; it ignores the value of the `editingAccessoryType` (page 613) 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.

The accessory type cross-fades between normal and editing states if it set for both states; use the `accessoryType` (page 610) property to set the accessory view for the cell during the table view's normal state. If this property is not set for both states, the cell is animated to slide or out, as necessary.

Availability
Available in iOS 3.0 and later.

See Also
`@property accessoryType` (page 610)
`@property accessoryView` (page 611)
`- willTransitionToState:` (page 628)
`- didTransitionToState:` (page 623)

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 630) 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 631) (not editable). The delegate returns the value this property for a particular cell in its implementation of the `tableView:editingStyleForRowAtIndexPath:` (page 939) method.

Availability
Available in iOS 2.0 and later.

See Also
`@property editing` (page 613)

Declared In
`UITableViewCell.h`

font

The font of the title. (Deprecated in iOS 3.0. Instead, set the fonts of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 3.0.

Declared In
`UITableViewCell.h`

hidesAccessoryWhenEditing

A Boolean value that determines whether the accessory view is hidden when the cell is being edited. (Deprecated in iOS 3.0. Use the `editingAccessoryType` (page 613) and `editingAccessoryView` (page 614) properties instead.)

```
@property(nonatomic) BOOL hidesAccessoryWhenEditing
```

Discussion
The default value is `YES`.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 3.0.

Declared In
UITableViewCell.h

highlighted

A Boolean value that indicates whether the cell is highlighted.

```
@property(nonatomic, getter=isHighlighted) BOOL highlighted
```

Discussion
The highlighting affects the appearance of labels, image, and background. When the the highlighted state of a cell is set to YES, labels are drawn in their highlighted text color (default is white). The default value is is NO. If you set the highlighted state to YES through this property, the transition to the new state appearance is not animated. For animated highlighted-state transitions, see the setHighlighted:animated: (page 627) method.

Note that for highlighting to work properly, you must fetch the cell's labels using the textLabel (page 623) and detailTextLabel (page 612) properties and set each label's highlightedTextColor property; for images, get the cell's image using the imageView (page 617) property and set the UIImageView object's highlightedImage property.

Availability
Available in iOS 3.0 and later.

Declared In
UITableViewCell.h

image

The image to use as content for the cell. (Deprecated in iOS 3.0. Instead use the imageView (page 617) property to get UIImageView object and then get or set the encapsulated image.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
@property selectedImage (page 619)

Declared In
UITableViewCell.h

imageView

Returns the image view of the table cell. (read-only)

```
@property(nonatomic, readonly, retain) UIImageView *imageView
```

Discussion
Returns the image view (`UIImageView` object) of the table view, which initially has no image set. If an image is set, it appears on the left side of the cell, before any label. `UITableViewCell` creates the image-view object when you create the cell.

Availability
Available in iOS 3.0 and later.

See Also
`- initWithStyle:reuseIdentifier:` (page 625)

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 617) property.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property indentationLevel` (page 617)

Declared In
`UITableViewCell.h`

lineBreakMode

The mode for for wrapping and truncating text in the cell. (Deprecated in iOS 3.0. Instead set the line-break mode attribute of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

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

Discussion
For further information, see the `UILineBreakMode` (page 56) constants described in *NSString UIKit Additions Reference*. The default value is `UILineBreakModeTailTruncation` (page 57).

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

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's 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 624) 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 587) method.

Availability
Available in iOS 2.0 and later.

See Also
`- prepareForReuse` (page 626)

Declared In
`UITableViewCell.h`

selected

A Boolean value that indicates whether the cell is selected.

```
@property(nonatomic, getter=isSelected) BOOL selected
```

Discussion
The selection affects the appearance of labels, image, and background. When the the selected state of a cell is set 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 627) method.

Availability
Available in iOS 2.0 and later.

See Also
`@property selectionStyle` (page 620)

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 600)) and non-`nil` for section-group tables `UITableViewStyleGrouped` (page 600)). `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 611)) if it is not `nil`, or behind all other views. Calling `setSelected:animated:` (page 627) causes the selected background view to animate in and out with an alpha fade.

Availability
Available in iOS 2.0 and later.

See Also
`@property backgroundView` (page 611)

Declared In
`UITableViewCell.h`

selectedImage

The image to use a cell content when the cell is selected. (Deprecated in iOS 3.0. Instead use the `imageView` (page 617) property to obtain the `UIImageView` object and then get or set its `highlightedImage` property.)

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

Discussion
The default value of this property is `nil` (no image).

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property image` (page 616)

Declared In
`UITableViewCell.h`

selectedTextColor

The color of the title text when the cell is selected. (Deprecated in iOS 3.0. Instead set the `highlightedTextColor` (page 345) property of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 3.0.

Declared In
`UITableViewCell.h`

selectionStyle

The style of selection for a cell.

```
@property(nonatomic) UITableViewCellSelectionStyle selectionStyle
```

Discussion
The selection style is a `backgroundView` (page 611) constant that determines the color of a cell when it is selected. The default value is `UITableViewCellSelectionStyleBlue` (page 630). See "Cell Selection Style" (page 630) for a description of valid constants.

Availability
Available in iOS 2.0 and later.

See Also
`@property selected` (page 618)
`- setSelected:animated:` (page 627)

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 617). The delegate can override this value in `tableView:shouldIndentWhileEditingRowAtIndexPath:` (page 942). This property has an effect only on table views created in the grouped style (`UITableViewStyleGrouped` (page 600)); it has no effect on `UITableViewStylePlain` (page 600) table views.

Availability
Available in iOS 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 iOS 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.

For the reordering control to appear, you must not only set this property but implement the `UITableViewDataSource` method `tableView:moveRowAtIndexPath:toIndexPath:` (page 930). In addition, if the data source implements `tableView:canMoveRowAtIndexPath:` (page 928) to return `NO`, the reordering control does not appear in that designated row.

Availability
Available in iOS 2.0 and later.

Declared In
`UITableViewCell.h`

target

The target object to receive action messages. (Deprecated in iOS 3.0. Instead use the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) or `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937) for handling taps on cells.)

```
@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 iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property editAction` (page 613)
`@property accessoryAction` (page 609)

Declared In
`UITableViewCell.h`

text

The text of the cell. (Deprecated in iOS 3.0. Use the `textLabel` (page 623) and `detailTextLabel` (page 612) properties instead.)

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

Discussion
The default is `nil` (no text).

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

Declared In
`UITableViewCell.h`

textAlignment

A constant that specifies the alignment of text in the cell. (Deprecated in iOS 3.0. Instead set the text alignment of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

```
@property(nonatomic) UITextAlignment textAlignment
```

Discussion
If the value of the property is `nil` (the default), the title is left-aligned (`UITextAlignmentLeft` (page 57)). See the descriptions of the `UITextAlignment` (page 57) constants for alternative text alignments.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

Declared In
`UITableViewCell.h`

textColor

The color of the title text. (Deprecated in iOS 3.0. Instead set the text color attribute of the `UILabel` objects assigned to the `textLabel` (page 623) and `detailTextLabel` (page 612) properties.)

```
@property(nonatomic, retain) UIColor *textColor
```

Discussion
If the value of property is `nil` (the default), the color of text is black.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

Declared In
`UITableViewCell.h`

textLabel

Returns the label used for the main textual content of the table cell. (read-only)

```
@property(nonatomic, readonly, retain) UILabel *textLabel
```

Discussion
Holds the main label of the cell. `UITableViewCell` adds an appropriate label when you create the cell in a given cell style. See "Cell Styles" (page 629) for descriptions of the main label in currently defined cell styles.

Availability
Available in iOS 3.0 and later.

See Also
`@property detailTextLabel` (page 612)
`- initWithStyle:reuseIdentifier:` (page 625)

Related Sample Code
AddMusic
BonjourWeb
GKRocket
MultipleDetailViews
WiTap

Declared In
`UITableViewCell.h`

Instance Methods

didTransitionToState:

Called on the cell just after it transitions between cell states.

```
- (void)didTransitionToState:(UITableViewCellStateMask)state
```

Parameters

state

A bit mask indicating the state or combination of states the cell is transitioning to.

Discussion

Subclasses of `UITableViewCell` can implement this method to animate additional changes to a cell when it is changing state. `UITableViewCell` calls this method whenever a cell transitions between states, such as from a normal state (the default) to editing mode. This method is called at the end of the animation block, which gives the custom cell a chance to clean up after the state change—for example, removing the edit and reorder controls after transitioning out of editing. Subclasses must always call `super` when overriding this method.

Note that when the user swipes a cell to delete it, the cell transitions to the state identified by the `UITableViewCellStateShowingDeleteConfirmationMask` (page 632) constant but the `UITableViewCellStateShowingEditControlMask` (page 632) is not set.

Availability

Available in iOS 3.0 and later.

See Also

- `willTransitionToState:` (page 628)
- `@property accessoryType` (page 610)
- `@property accessoryView` (page 611)
- `@property editingAccessoryType` (page 613)
- `@property editingAccessoryView` (page 614)

Declared In

`UITableViewCell.h`

initWithFrame:reuseIdentifier:

Initializes and returns a table cell object. (Deprecated in iOS 3.0. Use `initWithStyle:reuseIdentifier:` (page 625) instead.)

```
- (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 928), the table view's delegate calls the `UITableView` method `dequeueReusableCellWithIdentifier:` (page 587), passing in a reuse identifier, to obtain the cell object to use as the basis for the current row.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
`@property reuseIdentifier` (page 618)

Related Sample Code
AddMusic
GKRocket

Declared In
`UITableViewCell.h`

initWithStyle:reuseIdentifier:

Initializes a table cell with a style and a reuse identifier and returns it to the caller.

```
- (id)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString
    *)reuseIdentifier
```

Parameters

style
A constant indicating a cell style. See "Cell Styles" (page 629) for descriptions of these constants.

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. You should use the same reuse identifier for all cells of the same form.

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 928), the table view's delegate calls the `UITableView` method `dequeueReusableCellWithIdentifier:` (page 587), passing in a reuse identifier, to obtain the cell object to use as the basis for the current row.

If you want a table cell that has a configuration different that those defined by `UITableViewCell` for *style*, you must create your own custom cell. If you want to set the row height of cells on an individual basis, implement the delegate method `tableView:heightForRowAtIndexPath:` (page 941).

Availability
Available in iOS 3.0 and later.

Related Sample Code
BonjourWeb
CryptoExercise
MultipleDetailViews
ToolbarSearch
WiTap

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 587). 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 928) 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 iOS 2.0 and later.

See Also
`- initWithFrame:reuseIdentifier:` (page 624)
`@property reuseIdentifier` (page 618)

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 599) method of UITableView is invoked. Calling this method with *editing* set to NO removes the controls from the cell.

Availability
Available in iOS 2.0 and later.

See Also
@property editing (page 613)

Declared In
UITableViewCell.h

setHighlighted:animated:

Sets the highlighted state of the cell, optionally animating the transition between states.

```
- (void)setHighlighted:(BOOL)highlighted animated:(BOOL)animated
```

Parameters
highlighted
YES to set the cell as highlighted, NO to set it as unhighlighted. The default is NO.

animated
YES to animate the transition between highlighted states, NO to make the transition immediate.

Discussion
Highlights or unhighlights the cell, animating the transition between regular and highlighted state if animated is YES. Highlighting affects the appearance of the cell's labels, image, and background.

Note that for highlighting to work properly, you must fetch the cell's label (or labels) using the textLabel (page 623) (and detailTextLabel (page 612) properties and set the label's highlightedTextColor property; for images, get the cell's image using the imageView (page 617) property and set the UIImageView object's highlightedImage property.

A custom table cell may override this method to make any transitory appearance changes.

Availability
Available in iOS 3.0 and later.

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

The selection affects the appearance of labels, image, and background. When the the selected state of a cell to YES, it draws the background for selected cells ("Reusing Cells" (page 606)) with its title in white.

Availability

Available in iOS 2.0 and later.

See Also

@property selected (page 618)

@property selectionStyle (page 620)

Declared In

UITableViewCell.h

willTransitionToState:

Called on the cell just before it transitions between cell states.

```
- (void)willTransitionToState:(UITableViewCellStateMask)state
```

Parameters

state

A bit mask indicating the state or combination of states the cell is transitioning to.

Discussion

Subclasses of UITableViewCell can implement this method to animate additional changes to a cell when it is changing state. UITableViewCell calls this method whenever a cell transitions between states, such as from a normal state (the default) to editing mode. The custom cell can set up and position any new views that appear with the new state. The cell then receives a layoutSubviews (page 732) message (UIView) in which it can position these new views in their final locations for the new state. Subclasses must always call super when overriding this method.

Note that when the user swipes a cell to delete it, the cell transitions to the state identified by the UITableViewCellStateShowingDeleteConfirmationMask (page 632) constant but the UITableViewCellStateShowingEditControlMask (page 632) is not set.

Availability

Available in iOS 3.0 and later.

See Also

- didTransitionToState: (page 623)

@property accessoryType (page 610)

@property accessoryView (page 611)

@property editingAccessoryType (page 613)

@property editingAccessoryView (page 614)

Declared In
`UITableViewCell.h`

Constants

Cell Styles

An enumeration for the various styles of cells.

```
typedef enum {
    UITableViewCellStyleDefault,
    UITableViewCellStyleValue1,
    UITableViewCellStyleValue2,
    UITableViewCellStyleSubtitle
} UITableViewCellStyle;
```

Constants

`UITableViewCellStyleDefault`
A simple style for a cell with a text label (black and left-aligned) and an optional image view. Note that this is the default style for cells prior to iOS 3.0.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellStyleValue1`
A style for a cell with a label on the left side of the cell with left-aligned and black text; on the right side is a label that has smaller blue text and is right-aligned. The Settings application uses cells in this style.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellStyleValue2`
A style for a cell with a label on the left side of the cell with text that is right-aligned and blue; on the right side of the cell is another label with smaller text that is left-aligned and black. The Phone/Contacts application uses cells in this style.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellStyleSubtitle`
A style for a cell with a left-aligned label across the top and a left-aligned label below it in smaller gray text. The iPod application uses cells in this style.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

Discussion
In all these cell styles, the larger of the text labels is accessed via the `textLabel` (page 623) property and the smaller via the `detailTextLabel` (page 612) property.

UITableViewCellStateMask

The type of the constants used as cell-state masks.

```
typedef NSUInteger UITableViewCellStateMask;
```

Discussion
See "Cell State Mask Constants" (page 632) for descriptions of the constants.

Availability
Available in iOS 3.0 and later.

Declared In
`UITableViewCell.h`

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 iOS 2.0 and later.
Declared in `UITableViewCell.h`.

`UITableViewCellSelectionStyleBlue`
The cell when selected has a blue background. This is the default value.
Available in iOS 2.0 and later.
Declared in `UITableViewCell.h`.

`UITableViewCellSelectionStyleGray`
Then cell when selected has a gray background.
Available in iOS 2.0 and later.
Declared in `UITableViewCell.h`.

Discussion
You use these constants to to set the value of the `selectionStyle` (page 620) 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `UITableViewCell.h`.

Discussion

You use them to to set the value of the `editingStyle` (page 615) 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellAccessoryCheckmark`
The cell has a check mark on its right side. This control does not 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 939) method.

Available in iOS 2.0 and later.

Declared in `UITableViewCell.h`.

Discussion
You use these constants when setting the value of the `accessoryType` (page 610) property.

Declared In
`UITableViewCell.h`

Cell State Mask Constants

Constants used to determine the new state of a cell as it transitions between states.

```
enum {
    UITableViewCellStateDefaultMask                 = 0,
    UITableViewCellStateShowingEditControlMask      = 1 << 0,
    UITableViewCellStateShowingDeleteConfirmationMask   = 1 << 1
};
```

Constants
`UITableViewCellStateDefaultMask`
The normal state of a table cell.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellStateShowingEditControlMask`
The state of a table view cell when the table view is in editing mode.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellStateShowingDeleteConfirmationMask`
The state of a table view cell that shows a button requesting confirmation of a delete gesture.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

Discussion
The methods that use these constants are `didTransitionToState:` (page 623) and `willTransitionToState:` (page 628).

Cell Separator Style

The style for cells used as separators.

```
typedef enum {
    UITableViewCellSeparatorStyleNone,
    UITableViewCellSeparatorStyleSingleLine,
    UITableViewCellSeparatorStyleSingleLineEtched
} UITableViewCellSeparatorStyle;
```

Constants

`UITableViewCellSeparatorStyleNone`

The separator cell has no distinct style.

Available in iOS 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 iOS 2.0 and later.

Declared in `UITableViewCell.h`.

`UITableViewCellSeparatorStyleSingleLineEtched`

The separator cell has double lines running across its width, giving it an etched look. This style is currently only supported for grouped-style table views.

Available in iOS 3.2 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`

Convenience Definitions for Table View Cells

Synonyms for certain table-view cell constants.

```
#define UITableViewCellSeparatorStyleDoubleLineEtched
UITableViewCellSeparatorStyleSingleLineEtched
#define UITableViewCellStateEditingMask UITableViewCellStateShowingEditControlMask
```

Constants

`UITableViewCellSeparatorStyleDoubleLineEtched`

The separator cell has double lines running across its width, giving it an etched look. This style is currently only supported for grouped-style table views.

`UITableViewCellStateEditingMask`

The state of a table view cell when the table view is in editing mode.

Available in iOS 3.0 and later.

Declared in `UITableViewCell.h`.

Declared In

`UITableViewCell.h`

UITableViewController Class Reference

Inherits from	UIViewController : UIResponder : NSObject
Conforms to	UITableViewDelegate UITableViewDataSource NSCoding (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITableViewController.h
Companion guide	Table View Programming Guide for iOS
Related sample code	BonjourWeb GKRocket MultipleDetailViews ToolbarSearch WiTap

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 637) 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 the first time it's loaded, the table-view controller reloads the table view's data. It also clears its selection (with or without animation, depending on the request) every time the table view is displayed. The `UITableViewController` class implements this in the superclass method `viewWillAppear:` (page 773). You can disable this behavior by changing the value in the `clearsSelectionOnViewWillAppear` (page 636) property.
- 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 771).

- It implements the superclass method `setEditing:animated:` (page 769) 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 637), 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 767) 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 637)
Initializes a table-view controller to manage a table view of a given style.

Getting the Table View

`tableView` (page 637) *property*
Returns the table view managed by the controller object.

Configuring the Table Behavior

`clearsSelectionOnViewWillAppear` (page 636) *property*
A Boolean value indicating if the controller clears the selection when the table appears.

Properties

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

clearsSelectionOnViewWillAppear

A Boolean value indicating if the controller clears the selection when the table appears.

```
@property(nonatomic) BOOL clearsSelectionOnViewWillAppear
```

Discussion
The default value of this property is `YES`. When `YES`, the table view controller clears the table's current selection when it receives a `viewWillAppear:` (page 773) message. Setting this property to `NO` preserves the selection.

Availability
Available in iOS 3.2 and later.

Declared In
`UITableViewController.h`

tableView

Returns the table view managed by the controller object.

```
@property(nonatomic, retain) UITableView *tableView
```

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKRocket
MultipleDetailViews
ToolbarSearch

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 600) or `UITableViewStyleGrouped` (page 600)).

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 iOS 2.0 and later.

Related Sample Code
ToolbarSearch

Declared In
`UITableViewController.h`

UITapGestureRecognizer Class Reference

Inherits from	UIGestureRecognizer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITapGestureRecognizer.h
Companion guide	Event Handling Guide for iOS
Related sample code	ScrollViewSuite SimpleGestureRecognizers

Overview

`UITapGestureRecognizer` is a concrete subclass of `UIGestureRecognizer` that looks for single or multiple taps. For the gesture to be recognized, the specified number of fingers must tap the view a specified number of times.

Taps are discrete gestures, and thus the associated action message is sent only once per gesture. Action methods handling this gesture may get the location of the gesture as a whole by calling the `UIGestureRecognizer` method `locationInView:` (page 289); if there are multiple taps, this location is the first tap; if there are multiple touches, this location is the centroid of all fingers tapping the view. Clients may get the location of particular touches in the tap by calling `locationOfTouch:inView:` (page 290); if multiple taps are allowed, this location is that of the first tap.

Tasks

Configuring the Gesture

`numberOfTapsRequired` (page 640) *property*
The number of taps for the gesture to be recognized.

`numberOfTouchesRequired` (page 640) *property*
The number of fingers required to tap for the gesture to be recognized.

Properties

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

numberOfTapsRequired

The number of taps for the gesture to be recognized.

```
@property(nonatomic) NSUInteger numberOfTapsRequired
```

Discussion
The default value is 1.

Availability
Available in iOS 3.2 and later.

Declared In
`UITapGestureRecognizer.h`

numberOfTouchesRequired

The number of fingers required to tap for the gesture to be recognized.

```
@property(nonatomic) NSUInteger numberOfTouchesRequired
```

Discussion
The default value is 1.

Availability
Available in iOS 3.2 and later.

Declared In
`UITapGestureRecognizer.h`

UITextField Class Reference

Inherits from	UIControl : UIView : UIResponder : NSObject
Conforms to	NSCoding UITextInputTraits NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITextField.h
Related sample code	BonjourWeb MoviePlayer

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 466) 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 803)
- `UIKeyboardDidShowNotification` (page 803)
- `UIKeyboardWillHideNotification` (page 804)
- `UIKeyboardDidHideNotification` (page 804)

For more information about these notifications, see their descriptions in *UIWindow Class Reference*. For information about how to show and hide the keyboard, see "Text and Web".

Tasks

Accessing the Text Attributes

`text` (page 651) *property*
The text displayed by the text field.

`placeholder` (page 649) *property*
The string that is displayed when there is no other text in the text field.

`font` (page 647) *property*
The font of the text.

`textColor` (page 651) *property*
The color of the text.

`textAlignment` (page 651) *property*
The technique to use for aligning the text.

Sizing the Text Field's Text

`adjustsFontSizeToFitWidth` (page 644) *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 649) *property*
The size of the smallest permissible font with which to draw the text field's text.

Managing the Editing Behavior

`editing` (page 647) *property*
A Boolean value indicating whether the text field is currently in edit mode. (read-only)

`clearsOnBeginEditing` (page 646) *property*
A Boolean value indicating whether the text field removes old text when editing begins.

Setting the View's Background Appearance

`borderStyle` (page 645) *property*
The border style used by the text field.

`background` (page 645) *property*
The image that represents the background appearance of the text field when it is enabled.

`disabledBackground` (page 647) *property*
The image that represents the background appearance of the text field when it is disabled.

Managing Overlay Views

`clearButtonMode` (page 646) *property*
Controls when the standard clear button appears in the text field.

`leftView` (page 648) *property*
The overlay view displayed on the left side of the text field.

`leftViewMode` (page 649) *property*
Controls when the left overlay view appears in the text field.

`rightView` (page 650) *property*
The overlay view displayed on the right side of the text field.

`rightViewMode` (page 650) *property*
Controls when the right overlay view appears in the text field.

Accessing the Delegate

`delegate` (page 646) *property*
The receiver's delegate.

Drawing and Positioning Overrides

- `textRectForBounds:` (page 655)
  Returns the drawing rectangle for the text field's text.
- `drawTextInRect:` (page 653)
  Draws the receiver's text in the specified rectangle.
- `placeholderRectForBounds:` (page 654)
  Returns the drawing rectangle for the text field's placeholder text
- `drawPlaceholderInRect:` (page 653)
  Draws the receiver's placeholder text in the specified rectangle.
- `borderRectForBounds:` (page 652)
  Returns the receiver's border rectangle.
- `editingRectForBounds:` (page 653)
  Returns the rectangle in which editable text can be displayed.
- `clearButtonRectForBounds:` (page 652)
  Returns the drawing rectangle for the built-in clear button.
- `leftViewRectForBounds:` (page 654)
  Returns the drawing rectangle of the receiver's left overlay view.
- `rightViewRectForBounds:` (page 655)
  Returns the drawing location of the receiver's right overlay view.

Replacing the System Input Views

`inputView` (page 648) *property*
The custom input view to display when the text field becomes the first responder.

`inputAccessoryView` (page 648) *property*
The custom accessory view to display when the text field becomes the first responder

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 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 iOS 2.0 and later.

See Also
`@property minimumFontSize` (page 649)

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 iOS 2.0 and later.

See Also
`@property borderStyle` (page 645)
`@property disabledBackground` (page 647)

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 the value is set to the `UITextBorderStyleRoundedRect` style, the custom background image associated with the text field is ignored.

Availability
Available in iOS 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 iOS 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 952) method.

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

See Also
`@property background` (page 645)

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 iOS 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 iOS 2.0 and later.

Declared In
`UITextField.h`

inputAccessoryView

The custom accessory view to display when the text field becomes the first responder

```
@property (readwrite, retain) UIView *inputAccessoryView
```

Discussion
The default value of this property is `nil`. Assigning a view to this property causes that view to be displayed above the standard system keyboard (or above the custom input view if one is provided) when the text field becomes the first responder. For example, you could use this property to attach a custom toolbar to the keyboard.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextField.h`

inputView

The custom input view to display when the text field becomes the first responder.

```
@property (readwrite, retain) UIView *inputView
```

Discussion
If the value in this property is `nil`, the text field displays the standard system keyboard when it becomes first responder. Assigning a custom view to this property causes that view to be presented instead.

The default value of this property is `nil`.

Availability
Available in iOS 3.2 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 iOS 2.0 and later.

See Also
`- leftViewRectForBounds:` (page 654)

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 iOS 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 iOS 2.0 and later.

See Also
`@property adjustsFontSizeToFitWidth` (page 644)

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 iOS 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 right 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 iOS 2.0 and later.

See Also
`- rightViewRectForBounds:` (page 655)

Declared In
`UITextField.h`

rightViewMode

Controls when the right overlay view appears in the text field.

```
@property(nonatomic) UITextFieldViewMode rightViewMode
```

Discussion
The default value for this property is `UITextFieldViewModeNever`.

Availability
Available in iOS 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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 57).

Availability
Available in iOS 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 a black color. The value for the property can only be set to a non-`nil` value; setting this property to `nil` raises an exception.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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.

If the placeholder string is empty or `nil`, this method is not called.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in `UITextField.h`.

`UITextBorderStyleLine`
Displays a thin rectangle around the text field.
Available in iOS 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 iOS 2.0 and later.
Declared in `UITextField.h`.

`UITextBorderStyleRoundedRect`
Displays a rounded-style border for the text field.
Available in iOS 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 iOS 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 iOS 2.0 and later.
Declared in `UITextField.h`.

`UITextFieldViewModeUnlessEditing`
The overlay view is displayed only when text is not being edited.
Available in iOS 2.0 and later.
Declared in `UITextField.h`.

`UITextFieldViewModeAlways`
The overlay view is always displayed.
Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
`UITextField.h`

UITextInputStringTokenizer Class Reference

Inherits from	NSObject
Conforms to	UITextInputTokenizer NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h

Overview

The `UITextInputStringTokenizer` class is a base implementation of the UITextInputTokenizer protocol provided by the UIKit framework.

If you want to take advantage of this base implementation, you should subclass this class and handle application-specific directions and granularities affected by layout. When you instantiate a class you must supply the document class that is adopting the UITextInput protocol for your application.

Subclassing Notes

When you subclass `UITextInputStringTokenizer`, override all `UITextInputTokenizer` methods, calling the superclass implementation (`super`) when method parameters are not affected by layout. For example, the subclass needs a custom implementation of all methods for line granularity. For the left direction, it needs to decide whether left corresponds at a given position to forward or backward, and then call `super` passing in the storage direction (UITextStorageDirection).

Tasks

Initializing a Tokenizer

`- initWithTextInput:` (page 660)
Returns an object initialized with the document object that directly communicates with the text input system.

Instance Methods

initWithTextInput:

Returns an object initialized with the document object that directly communicates with the text input system.

```
- (id)initWithTextInput:(UIResponder < UITextInput > *)textInput
```

Parameters

textInput

The document object in the application that adopts the UITextInput protocol for the purposes of communicating with the text input system.

Return Value

An instance of a subclass of `UITextInputStringTokenizer`, or `nil` if the object couldn't be created.

Discussion

The subclass of UITextInputStringTokenizer should not retain *textInput*; the tokenizer should always have a lifetime bounded by that of the UITextInput-conforming object and a retaining reference would create a retain cycle.

Availability

Available in iOS 3.2 and later.

Declared In

`UITextInput.h`

UITextPosition Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h

Overview

A `UITextPosition` object represents a position in a text container; in other words, it is an index into the backing string in a text-displaying view.

Classes that adopt the UITextInput protocol must create custom `UITextPosition` objects for representing specific locations within the text managed by the class. The text input system uses both these objects and UITextRange objects for communicating text-layout information. There are two reasons for using objects for text positions rather than primitive types such as `NSInteger`:

- Some documents contain nested elements (for example, HTML tags and embedded objects) and you need to track both absolute position and position in the visible text.
- The WebKit framework, which the iPhone text system is based on, requires that text indexes and offsets be represented by objects.

The simplest of `UITextPosition` objects—for example, one used in plain text—might have a single integer property that represents an offset into a string. If you adopt the UITextInput protocol, you must create a custom UITextRange subclass as well as a custom `UITextPosition` subclass.

This class declares no methods of its own.

UITextRange Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h

Overview

A `UITextRange` object represents a range of characters in a text container; in other words, it identifies a starting index and an ending index in string backing a text-entry object.

Classes that adopt the UITextInput protocol must create custom `UITextRange` objects for representing ranges within the text managed by the class. The starting and ending indexes of the range are represented by UITextPosition objects. The text system uses both `UITextRange` and `UITextPosition` objects for communicating text-layout information. There are two reasons for using objects for text ranges rather than primitive types such as `NSRange`:

- Some documents contain nested elements (for example, HTML tags and embedded objects) and you need to track both absolute position and position in the visible text.
- The WebKit framework, which the iPhone text system is based on, requires that text indexes and offsets be represented by objects.

If you adopt the UITextInput protocol, you must create a custom `UITextRange` subclass as well as a custom `UITextPosition` subclass.

Tasks

Defining Ranges of Text

`start` (page 664) *property*
The start of a range of text. (read-only)

`end` (page 664) *property*
The end of the range of text. (read-only)

`empty` (page 664) *property*
A Boolean value that indicates whether the range of text represented by the receiver is zero-length. (read-only)

Properties

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

empty

A Boolean value that indicates whether the range of text represented by the receiver is zero-length. (read-only)

```
@property(nonatomic, readonly, getter=isEmpty) BOOL empty
```

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

end

The end of the range of text. (read-only)

```
@property(nonatomic, readonly) UITextPosition *end
```

Discussion
Compute and store in this property the UITextPosition object representing the end of a range of text.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

start

The start of a range of text. (read-only)

```
@property(nonatomic, readonly) UITextPosition *start
```

Discussion
Compute and store in this property the UITextPosition object representing the start of a range of text.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

UITextView Class Reference

Inherits from	UIScrollView : UIView : UIResponder : NSObject
Conforms to	UITextInputTraits NSCoding (UIScrollView) NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITextView.h
Companion guide	iOS Application Programming Guide
Related sample code	CryptoExercise KeyboardAccessory

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 466) 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 803)
- `UIKeyboardDidShowNotification` (page 803)
- `UIKeyboardWillHideNotification` (page 804)
- `UIKeyboardDidHideNotification` (page 804)

For more information about these notifications, see their descriptions in *UIWindow Class Reference*.

Tasks

Configuring the Text Attributes

`text` (page 670) *property*
The text displayed by the text view.

`font` (page 668) *property*
The font of the text.

`textColor` (page 670) *property*
The color of the text.

`editable` (page 668) *property*
A Boolean value indicating whether the receiver is editable.

`dataDetectorTypes` (page 667) *property*
The types of data converted to clickable URLs in the text view.

`textAlignment` (page 670) *property*
The technique to use for aligning the text.

- `hasText` (page 671)
Returns a Boolean value indicating whether the text view currently contains any text.

Working with the Selection

`selectedRange` (page 669) *property*
The current selection range of the receiver.

`- scrollRangeToVisible:` (page 671)
Scrolls the receiver until the text in the specified range is visible.

Accessing the Delegate

`delegate` (page 667) *property*
The receiver's delegate.

Replacing the System Input Views

`inputView` (page 669) *property*
The custom input view to display when the text view becomes the first responder.

`inputAccessoryView` (page 668) *property*
The custom accessory view to display when the text view becomes the first responder

Properties

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

dataDetectorTypes

The types of data converted to clickable URLs in the text view.

```
@property(nonatomic) UIDataDetectorTypes dataDetectorTypes
```

Discussion
You can use this property to specify the types of data (phone numbers, `http` links, and so on) that should be automatically converted to clickable URLs in the text view. When clicked, the text view opens the application responsible for handling the URL type and passes it the URL.

Availability
Available in iOS 3.0 and later.

Declared In
`UITextView.h`

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.

For information about the methods implemented by the delegate, see *UITextViewDelegate Protocol Reference*.

Availability
Available in iOS 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 iOS 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.

Note: You can get information about the fonts available on the system using the methods of the `UIFont` class.

Availability
Available in iOS 2.0 and later.

Declared In
`UITextView.h`

inputAccessoryView

The custom accessory view to display when the text view becomes the first responder

```
@property (readwrite, retain) UIView *inputAccessoryView
```

Discussion
The default value of this property is `nil`. Assigning a view to this property causes that view to be displayed above the standard system keyboard (or above the custom input view if one is provided) when the text view becomes the first responder. For example, you could use this property to attach a custom toolbar to the keyboard.

Availability
Available in iOS 3.2 and later.

See Also
`@property inputView` (page 669)

Related Sample Code
KeyboardAccessory

Declared In
`UITextView.h`

inputView

The custom input view to display when the text view becomes the first responder.

```
@property (readwrite, retain) UIView *inputView
```

Discussion
If the value in this property is `nil`, the text view displays the standard system keyboard when it becomes first responder. Assigning a custom view to this property causes that view to be presented instead.

The default value of this property is `nil`.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextView.h`

selectedRange

The current selection range of the receiver.

```
@property(nonatomic) NSRange selectedRange
```

Discussion
In iOS 2.2 and earlier, the length of the selection range is always 0, indicating that the selection is actually an insertion point. In iOS 3.0 and later, the length of the selection range may be non-zero.

Availability
Available in iOS 2.0 and later.

Related Sample Code
KeyboardAccessory

Declared In
UITextView.h

text

The text displayed by the text view.

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

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise
KeyboardAccessory

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 57).

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property backgroundColor` (page 697) (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 iOS 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 of text to scroll into view.

Availability
Available in iOS 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 iOS 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. The `userInfo` dictionary is not used.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
`UITextView.h`

UIToolbar Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIToolbar.h
Related sample code	MultipleDetailViews ToolbarSearch

Overview

An instance of the `UIToolbar` class is 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 675) method to add them to a toolbar. All methods with an `animated:` argument allow you to optionally animate changes to the display.

Toolbar images that represent normal and highlighted states of an item derive from the image you set using the inherited `image` (page 150) property from the `UIBarItem` class. For example, the image is converted to white and then bevelled by adding a shadow for the normal state.

Tasks

Configuring the Toolbar

`barStyle` (page 674) *property*
The toolbar style that specifies its appearance.

`tintColor` (page 675) *property*
The color used to tint the bar.

`translucent` (page 675) *property*
A Boolean value that indicates whether the toolbar is translulcent (`YES`) or not (`NO`).

Configuring Toolbar Items

`items` (page 674) *property*
The items displayed on the toolbar.

`- setItems:animated:` (page 675)
Sets the items on the toolbar by animating the changes.

Properties

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

barStyle

The toolbar style that specifies its appearance.

```
@property(nonatomic) UIBarStyle barStyle
```

Discussion
See `UIBarStyle` (page 1011) for possible values. The default value is `UIBarStyleDefault` (page 1011).

Availability
Available in iOS 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 675) method to animate changes.

The default value is `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`- setItems:animated:` (page 675)

Related Sample Code
ToolbarSearch

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 iOS 2.0 and later.

Declared In
UIToolbar.h

translucent

A Boolean value that indicates whether the toolbar is translulcent (`YES`) or not (`NO`).

```
@property(nonatomic,assign,getter=isTranslucent) BOOL translucent
```

Discussion
Applying translucence to a toolbar is intended primarily for landscape orientation, when you want the user to be able to view the area beneath the toolbar. The default value for this property is `NO`. However, if you set the toolbar style to `UIBarStyleBlackTranslucent` (page 1012), the value for this property is always `YES`.

Availability
Available in iOS 3.0 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 iOS 2.0 and later.

See Also
@property items (page 674)

Declared In
UIToolbar.h

UITouch Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITouch.h
Related sample code	aurioTouch GKTank MoviePlayer ScrollViewSuite SimpleGestureRecognizers

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.

The `gestureRecognizers` (page 678) property, which was introduced in iOS 3.2, returns the gesture recognizers—that is, instances of a concrete subclass of UIGestureRecognizer—that are currently handling the given touch.

See Event Handling in *iOS Application Programming Guide* for further information on event handling.

Tasks

Getting the Location of Touches

`- locationInView:` (page 681)
Returns the current location of the receiver in the coordinate system of the given view.

`- previousLocationInView:` (page 681)
Returns the previous location of the receiver in the coordinate system of the given view.

`view` (page 680) *property*
The view in which the touch initially occurred. (read-only)

`window` (page 680) *property*
The window in which the touch initially occurred. (read-only)

Getting Touch Attributes

`tapCount` (page 679) *property*
The number of times the finger was tapped for this given touch. (read-only)

`timestamp` (page 679) *property*
The time when the touch occurred or when it was last mutated. (read-only)

`phase` (page 679) *property*
The type of touch. (read-only)

Getting a Touch Object's Gesture Recognizers

`gestureRecognizers` (page 678) *property*
The gesture recognizers that are receiving the touch object.

Properties

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

gestureRecognizers

The gesture recognizers that are receiving the touch object.

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

Discussion
The objects in the array are instances of a subclass of the abstract base class UIGestureRecognizer. If there are no gesture recognizers currently receiving the touch objects, this property holds an empty array.

Availability
Available in iOS 3.2 and later.

Declared In
`UITouch.h`

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 682).

Availability
Available in iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 you want to determine whether the user single-tapped, double-tapped, or even triple-tapped a particular view or window, you should evaluate the value returned by this method.

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKTank
ScrollViewSuite

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. For a definition of the time-since-boot value, see the description of the `systemUptime` method of the `NSProcessInfo` class.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`@property window` (page 680)

Related Sample Code
SimpleGestureRecognizers

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 iOS 2.0 and later.

See Also
`@property view` (page 680)

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 iOS 2.0 and later.

See Also
`- previousLocationInView:` (page 681)

Related Sample Code
GKTank
ScrollViewSuite

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 iOS 2.0 and later.

See Also
`- locationInView:` (page 681)

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 iOS 2.0 and later.
Declared in `UITouch.h`.

`UITouchPhaseMoved`
A finger for a given event moved on the screen.
Available in iOS 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 iOS 2.0 and later.
Declared in `UITouch.h`.

`UITouchPhaseEnded`
A finger for a given event was lifted from the screen.
Available in iOS 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 iOS 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 679) property.

Declared In

`UITouch.h`

UIVideoEditorController Class Reference

Inherits from	UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.1 and later.
Declared in	

Overview

A `UIVideoEditorController` object, or video editor, manages the system-supplied user interface for trimming video frames from the start and end of a previously recorded movie as well as reencoding to lower quality. The object manages user interactions and provides the filesystem path of the edited movie to your delegate object (see *UIVideoEditorControllerDelegate Protocol Reference*). The features of the `UIVideoEditorController` class are available only on devices that support video recording.

Use a video editor when your intent is to provide an interface for movie editing. While the `UIImagePickerController` class also lets a user trim movies, its primary roles are choosing saved pictures and movies, and capturing new pictures and movies.

Important: The `UIVideoEditorController` class supports portrait mode only. This class is intended to be used as-is and does not support subclassing. The view hierarchy for this class is private; do not modify the view hierarchy. This class does not support modifications to its appearance by use of overlay views.

Tasks

Determining Editing Availability

`+ canEditVideoAtPath:` (page 687)
Returns a Boolean value indicating whether a video file can be edited.

Configuring the Editor

`delegate` (page 686) *property*
The video editor's delegate object.

`videoMaximumDuration` (page 686) *property*
The maximum duration, in seconds, permitted for trimmed movies saved by the video editor.

`videoPath` (page 687) *property*
The filesystem path to the movie to be loaded by the video editor.

`videoQuality` (page 687) *property*
The video quality to use when saving a trimmed movie.

Properties

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

delegate

The video editor's delegate object.

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

Discussion
The delegate receives a notification when the system has finished saving an edited movie or when the user cancels the video editor. The delegate also decides when to dismiss the editor interface, so you must provide a delegate to use a video editor. If this property is `nil`, the editor is dismissed immediately if you try to show it. The delegate protocol is described in *UIVideoEditorControllerDelegate Protocol Reference*.

Availability
Available in iOS 3.1 and later.

Declared In
`UIVideoEditorController.h`

videoMaximumDuration

The maximum duration, in seconds, permitted for trimmed movies saved by the video editor.

```
@property(nonatomic) NSTimeInterval videoMaximumDuration
```

Discussion
The system-enforced maximum duration for a video recording is 10 minutes; you can set this value to 10 minutes or less. The default value for this property is also 10 minutes.

The video editor user interface forces the user to trim a loaded movie to fit within this property's value prior to saving.

Availability
Available in iOS 3.1 and later.

Declared In
`UIVideoEditorController.h`

videoPath

The filesystem path to the movie to be loaded by the video editor.

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

Availability
Available in iOS 3.1 and later.

Declared In
`UIVideoEditorController.h`

videoQuality

The video quality to use when saving a trimmed movie.

```
@property(nonatomic) UIImagePickerControllerQualityType videoQuality
```

Discussion
The available video qualities are described in the "UIImagePickerControllerQualityType" (page 329) enumeration. The default value for this property is `UIImagePickerControllerQualityTypeLow` (page 330).

If a user attempts to reencode a movie to a higher quality, the movie is saved at its existing quality. Reencoding never increases movie dimensions, frame rate, or bit rate.

Availability
Available in iOS 3.1 and later.

Declared In
`UIVideoEditorController.h`

Class Methods

canEditVideoAtPath:

Returns a Boolean value indicating whether a video file can be edited.

```
+ (BOOL)canEditVideoAtPath:(NSString *)videoPath
```

Parameters

videoPath
The filesystem path to the video file you want to edit.

Return Value
`YES` if the specified video file can be edited on the current device or `NO` if it cannot.

Discussion
Video editing requires the presence of specific hardware and is available only for specific file formats. Use this method to check whether video editing is available for a given video file, before you create a video editor.

Availability
Available in iOS 3.1 and later.

Declared In
`UIVideoEditorController.h`

UIView Class Reference

Inherits from	UIResponder : NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIView.h
Related sample code	KeyboardAccessory ScrollViewSuite SimpleGestureRecognizers SpeakHere WiTap

Overview

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

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

The `UIView` class provides common methods you use to create all types of views and access their properties. For example, unless a subclass has its own designated initializer, you use the `initWithFrame:` (page 729) method to create a view. The `frame` (page 701) 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 698) and `bounds` (page 698) 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 696) property, especially if you subclass `UIView`, to specify whether the view should automatically resize its subviews. Use the `autoresizingMask` (page 697) property with the constants described in `UIViewAutoresizing` (page 741) 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 705) property to get the parent view and the `subviews` (page 705) property to get the child views in the hierarchy. There are also a number of methods, listed in "Managing the View Hierarchy" (page 691), 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 726) 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 734) or `setNeedsDisplayInRect:` (page 735) to the view. You can also set the `contentMode` (page 700) to `UIViewContentModeRedraw` (page 740) to invoke the `drawRect:` (page 726) 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 729), to create a view hierarchy. If you want to programmatically force the layout of subviews before drawing, send `setNeedsLayout` (page 735) to the view. Then when `layoutIfNeeded` (page 731) is invoked, the `layoutSubviews` (page 732) method is invoked just before displaying. Subclasses should override `layoutSubviews` (page 732) to perform any custom arrangement of subviews.

Some of the property changes to view objects can be animated—for example, setting the `frame` (page 701), `bounds` (page 698), `center` (page 698), and `transform` (page 706) 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 709) class method to begin an animation block, set the properties you want animated, and then invoke the `commitAnimations` (page 710) 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 728) and `pointInside:withEvent:` (page 732) methods if you are processing events and want to know where they occur. The `UIView` class inherits other event processing methods from `UIResponder`. For more information on how views handle events, read *UIResponder Class Reference*.

To associate a gesture recognizer with a view so that object can interpret gestures made on the view, you must call the `addGestureRecognizer:` (page 721) method. (Gesture recognizers are instances of a concrete subclass of `UIGestureRecognizer`.) You remove a gesture recognizer with the `removeGestureRecognizer:` (page 733) method and find out which gesture recognizers are associated with a view using the `gestureRecognizers` (page 702) property. Gesture recognition is a feature that was introduced in iOS 3.2.

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

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

Tasks

Creating Instances

- `initWithFrame:` (page 729)
  Initializes and returns a newly allocated view object with the specified frame rectangle.

Setting and Getting Attributes

`userInteractionEnabled` (page 706) *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 701) *property*
The receiver's frame rectangle.

`bounds` (page 698) *property*
The receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

`center` (page 698) *property*
The center of the frame.

`transform` (page 706) *property*
Specifies the transform applied to the receiver, relative to the center of its bounds.

Managing the View Hierarchy

`superview` (page 705) *property*
The receiver's superview, or `nil` if it has none. (read-only)

`subviews` (page 705) *property*
The receiver's immediate subviews. (read-only)

`window` (page 707) *property*
The receiver's window object, or `nil` if it has none. (read-only)

- `addSubview:` (page 722)
  Adds a view to the receiver's subviews so it's displayed above its siblings.
- `bringSubviewToFront:` (page 723)
  Moves the specified subview to the front of its siblings.

- `sendSubviewToBack:` (page 734)
  Moves the specified subview to the back of its siblings.
- `removeFromSuperview` (page 733)
  Unlinks the receiver from its superview and its window, and removes it from the responder chain.
- `insertSubview:atIndex:` (page 730)
  Inserts a subview at the specified index.
- `insertSubview:aboveSubview:` (page 729)
  Inserts a view above another view in the view hierarchy.
- `insertSubview:belowSubview:` (page 730)
  Inserts a view below another view in the view hierarchy.
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 727)
  Exchanges the subviews in the receiver at the given indices.
- `isDescendantOfView:` (page 731)
  Returns a Boolean value indicating whether the receiver is a subview of a given view or whether it is identical to that view.

Converting Coordinates

- `convertPoint:toView:` (page 723)
  Converts a point from the receiver's coordinate system to that of a given view.
- `convertPoint:fromView:` (page 723)
  Converts a point from the coordinate system of a given view to that of the receiver.
- `convertRect:toView:` (page 725)
  Converts a rectangle from the receiver's coordinate system to that of another view.
- `convertRect:fromView:` (page 724)
  Converts a rectangle from the coordinate system of another view to that of the receiver.

`contentScaleFactor` (page 700) *property*
The scale factor applied to the view.

Resizing Subviews

`autoresizesSubviews` (page 696) *property*
A Boolean value that determines whether the receiver automatically resizes its subviews when its frame size changes.

`autoresizingMask` (page 697) *property*
An integer bit mask that determines how the receiver resizes itself when its bounds change.

- `sizeThatFits:` (page 736)
  Asks the view to calculate and return the size that best fits its subviews.
- `sizeToFit` (page 736)
  Resizes and moves the receiver view so it just encloses its subviews.

`contentMode` (page 700) *property*
A flag used to determine how a view lays out its content when its bounds rectangle changes.

`contentStretch` (page 700) *property*
The rectangle that defines the stretchable and nonstretchable regions of a view.

Searching for Views

`tag` (page 705) *property*
The receiver's tag, an integer that you can use to identify view objects in your application.

`- viewWithTag:` (page 737)
Returns the view with the specified tag.

Laying out Views

`- setNeedsLayout` (page 735)
Sets whether subviews need to be rearranged before displaying.

`- layoutIfNeeded` (page 731)
Lays out the subviews if needed.

`- layoutSubviews` (page 732)
Lays out subviews.

Displaying

`clipsToBounds` (page 699) *property*
A Boolean value that determines whether subviews can be drawn outside the bounds of the receiver.

`backgroundColor` (page 697) *property*
The receiver's background color.

`alpha` (page 696) *property*
The receiver's alpha value.

`opaque` (page 704) *property*
A Boolean value that determines whether the receiver is opaque.

`clearsContextBeforeDrawing` (page 699) *property*
A Boolean value that determines whether the receiver's bounds should be automatically cleared before drawing.

`- drawRect:` (page 726)
Draws the receiver's image within the passed-in rectangle.

`- setNeedsDisplay` (page 734)
Controls whether the receiver's entire bounds rectangle is marked as needing display.

`- setNeedsDisplayInRect:` (page 735)
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 711)
Returns the class used to create the layer for instances of this class.

`layer` (page 703) *property*
The view's Core Animation layer used for rendering. (read-only)

`hidden` (page 703) *property*
A Boolean value that determines whether the receiver is hidden.

Animating Views with Blocks

+ `animateWithDuration:delay:options:animations:completion:` (page 708)
  Animate changes to one or more views using the specified duration, delay, options, and completion handler.
+ `animateWithDuration:animations:completion:` (page 708)
  Animate changes to one or more views using the specified duration and completion handler.
+ `animateWithDuration:animations:` (page 707)
  Animate changes to one or more views using the specified duration.
+ `transitionWithView:duration:options:animations:completion:` (page 720)
  Creates a transition animation for the specified container view.
+ `transitionFromView:toView:duration:options:completion:` (page 720)
  Creates a transition animation between the specified views using the given parameters.

Animating Views

+ `beginAnimations:context:` (page 709)
  Begins an animation block.
+ `commitAnimations` (page 710)
  Ends an animation block and starts animations when this is the outer animation block.
+ `setAnimationStartDate:` (page 717)
  Sets the start time of animating property changes within an animation block.
+ `setAnimationsEnabled:` (page 717)
  Sets whether animations are enabled.
+ `setAnimationDelegate:` (page 713)
  Sets the delegate for animation messages.
+ `setAnimationWillStartSelector:` (page 719)
  Sets the message to send to the animation delegate when animation starts.
+ `setAnimationDidStopSelector:` (page 714)
  Sets the message to send to the animation delegate when animation stops.
+ `setAnimationDuration:` (page 715)
  Sets the duration (in seconds) of animating property changes within an animation block.
+ `setAnimationDelay:` (page 713)
  Sets the delay (in seconds) of animating property changes within an animation block.
+ `setAnimationCurve:` (page 712)
  Sets the curve of animating property changes within an animation block.
+ `setAnimationRepeatCount:` (page 716)
  Sets the number of times animations within an animation block repeat.
+ `setAnimationRepeatAutoreverses:` (page 716)
  Sets whether the animation of property changes within an animation block automatically reverses repeatedly.
+ `setAnimationBeginsFromCurrentState:` (page 711)
  Sets whether the animation should begin playing from the current state.

+ `setAnimationTransition:forView:cache:` (page 718)
  Sets a transition to apply to a view during an animation block.

+ `areAnimationsEnabled` (page 709)
  Returns a Boolean value indicating whether animations are enabled.

Handling Events

- `hitTest:withEvent:` (page 728)
  Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains a specified point.

- `pointInside:withEvent:` (page 732)
  Returns a Boolean value indicating whether the receiver contains the specified point.

`multipleTouchEnabled` (page 704) *property*
  A Boolean value indicating whether the receiver handles multi-touch events.

`exclusiveTouch` (page 701) *property*
  A Boolean value indicating whether the receiver handles touch events exclusively.

- `endEditing:` (page 727)
  Causes the view (or one of its embedded text fields) to resign the first responder status.

Managing Gesture Recognizers

- `addGestureRecognizer:` (page 721)
  Attaches a gesture recognizer to the receiving view.

- `removeGestureRecognizer:` (page 733)
  Detaches a gesture recognizer from the receiving view.

`gestureRecognizers` (page 702) *property*
  The gesture-recognizer objects currently attached to the view.

Observing Changes

- `didAddSubview:` (page 725)
  Tells the view when subviews are added.

- `didMoveToSuperview` (page 726)
  Informs the receiver that its superview has changed (possibly to `nil`).

- `didMoveToWindow` (page 726)
  Informs the receiver that it has been added to a window.

- `willMoveToSuperview:` (page 737)
  Informs the receiver that its superview is about to change to the specified superview (which may be `nil`).

- `willMoveToWindow:` (page 738)
  Informs the receiver that it's being added to the view hierarchy of the specified window object (which may be `nil`).

`- willRemoveSubview:` (page 738)
Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

Properties

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

alpha

The receiver's alpha value.

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

Discussion
Changes to this property can be animated. Use the `beginAnimations:context:` (page 709) class method to begin and the `commitAnimations` (page 710) class method to end an animation block.

Availability
Available in iOS 2.0 and later.

See Also
`@property backgroundColor` (page 697)
`@property opaque` (page 704)

Related Sample Code
SimpleGestureRecognizers

Declared In
`UIView.h`

autoresizesSubviews

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
`@property autoresizingMask` (page 697)

Declared In
`UIView.h`

autoresizingMask

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

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

Discussion
This mask can be specified by combining, using the C bitwise OR operator, any of the options described in `UIViewAutoresizing` (page 741).

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 701) and `autoresizingMask` (page 697) 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 732) method to explicitly adjust the position of subviews.

If the autoresizing mask is equal to `UIViewAutoresizingNone` (page 742), then the receiver doesn't resize at all when its bounds changes. The default value is `UIViewAutoresizingNone`.

Availability
Available in iOS 2.0 and later.

See Also
`@property autoresizesSubviews` (page 696)

Related Sample Code
BonjourWeb
WiTap

Declared In
`UIView.h`

backgroundColor

The receiver's background color.

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

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

Availability
Available in iOS 2.0 and later.

See Also
`@property alpha` (page 696)
`@property opaque` (page 704)

Related Sample Code
aurioTouch

Declared In
`UIView.h`

bounds

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

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

Discussion
The bounds rectangle determines the origin and scale in the view's coordinate system within its frame rectangle and is measured in points. Setting this property changes the value of the `frame` (page 701) property accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 726) method. If you want the `drawRect:` (page 726) method invoked when the frame rectangle changes, set the `contentMode` (page 700) property to `UIViewContentModeRedraw` (page 740).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 709) class method to begin and the `commitAnimations` (page 710) class method to end an animation block.

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

Availability
Available in iOS 2.0 and later.

See Also
`@property frame` (page 701)
`@property center` (page 698)
`@property transform` (page 706)

Related Sample Code
SpeakHere
WiTap

Declared In
`UIView.h`

center

The center of the frame.

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

Discussion
The center is specified within the coordinate system of its superview and is measured in points. Setting this property changes the values of the `frame` (page 701) properties accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 726) method. If you want the `drawRect:` (page 726) method invoked when the frame rectangle changes, set the `contentMode` (page 700) property to `UIViewContentModeRedraw` (page 740).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 709) class method to begin and the `commitAnimations` (page 710) class method to end an animation block.

Availability
Available in iOS 2.0 and later.

See Also
`@property frame` (page 701)
`@property bounds` (page 698)
`@property transform` (page 706)

Related Sample Code
GKRocket
GKTank
SimpleGestureRecognizers

Declared In
`UIView.h`

clearsContextBeforeDrawing

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

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

Discussion
The default value of this property is `YES`. When set to `YES`, the current graphics context buffer in the `drawRect:` (page 726) method is automatically cleared to transparent black before `drawRect:` (page 726) is invoked. If the view's `opaque` (page 704) property is also set to `YES`, the `backgroundColor` (page 697) property of the view must not be `nil` or drawing errors may occur.

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

clipsToBounds

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

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

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

Availability
Available in iOS 2.0 and later.

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 739). The default value is `UIViewContentModeScaleToFill` (page 740).

Availability
Available in iOS 2.0 and later.

Related Sample Code
SimpleGestureRecognizers

Declared In
`UIView.h`

contentScaleFactor

The scale factor applied to the view.

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

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

For views that implement a custom `drawRect:` (page 726) method and are associated with a window, the default value for this property is the scale factor associated with the screen currently displaying the view. For system views and views that are backed by a `CAEAGLLayer` object for, the value of this property may be `1.0` even on high resolution screens.

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

Availability
Available in iOS 4.0 and later.

Declared In
`UIView.h`

contentStretch

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

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

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

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

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

Availability
Available in iOS 3.0 and later.

Declared In
`UIView.h`

exclusiveTouch

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
`@property multipleTouchEnabled` (page 704)

Declared In
`UIView.h`

frame

The receiver's frame rectangle.

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

Discussion
This rectangle is measured in points. Setting the frame rectangle repositions and resizes the receiver within the coordinate system of its superview. The origin of the frame is in superview coordinates. Setting this property changes the values of the `center` (page 698) and `bounds` (page 698) properties accordingly.

Changing the frame rectangle automatically redisplays the receiver without invoking the `drawRect:` (page 726) method. If you want the `drawRect:` (page 726) method invoked when the frame rectangle changes, set the `contentMode` (page 700) property to `UIViewContentModeRedraw` (page 740).

Changes to this property can be animated. Use the `beginAnimations:context:` (page 709) class method to begin and the `commitAnimations` (page 710) class method to end an animation block. If the `transform` (page 706) property is also set, use the `bounds` (page 698) and `center` (page 698) properties instead; otherwise, animating changes to the `frame` property does not correctly reflect the actual location of the view.

Warning: If the `transform` (page 706) property is not the identity transform, the value of this property is undefined and therefore should be ignored.

Availability
Available in iOS 2.0 and later.

See Also
`@property bounds` (page 698)
`@property center` (page 698)
`@property transform` (page 706)

Related Sample Code
GKTank
KeyboardAccessory
ScrollViewSuite
SpeakHere
WiTap

Declared In
`UIView.h`

gestureRecognizers

The gesture-recognizer objects currently attached to the view.

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

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

Availability
Available in iOS 3.2 and later.

Declared In
`UIView.h`

hidden

A Boolean value that determines whether the receiver is hidden.

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

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

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

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

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

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise
GKTank

Declared In
`UIView.h`

layer

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

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
`+ layerClass` (page 711)

Related Sample Code
aurioTouch
GLSprite
ScrollViewSuite
SpeakHere

Declared In
`UIView.h`

multipleTouchEnabled

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

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

Discussion
If `YES`, the receiver handles multi-touch events; otherwise, it does not. If `NO`, the receiver is sent only the first touch event in a multi-touch sequence. Other views in the same window can still receive touch events when this property is `NO`. Set this property and the `exclusiveTouch` (page 701) property to `YES` if this view should handle multi-touch events exclusively—for example, when tracking a sequence of multi-touch events. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`@property exclusiveTouch` (page 701)

Related Sample Code
aurioTouch

Declared In
`UIView.h`

opaque

A Boolean value that determines whether the receiver is opaque.

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

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

Availability
Available in iOS 2.0 and later.

See Also
`@property backgroundColor` (page 697)
`@property alpha` (page 696)

Declared In
UIView.h

subviews

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

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

Availability
Available in iOS 2.0 and later.

See Also
@property superview (page 705)
- removeFromSuperview (page 733)

Related Sample Code
ScrollViewSuite

Declared In
UIView.h

superview

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

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

Availability
Available in iOS 2.0 and later.

See Also
@property subviews (page 705)
- removeFromSuperview (page 733)

Related Sample Code
WiTap

Declared In
UIView.h

tag

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
`- viewWithTag:` (page 737)

Related Sample Code
WiTap

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 698) property, or the layer's `anchorPoint` property if it was changed. (Use the `layer` (page 703) 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 709) class method to begin and the `commitAnimations` (page 710) 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 701) property is undefined and therefore should be ignored.

Availability
Available in iOS 2.0 and later.

See Also
`@property frame` (page 701)
`@property bounds` (page 698)
`@property center` (page 698)

Related Sample Code
aurioTouch
GKTank
MoviePlayer
SimpleGestureRecognizers

Declared In
`UIView.h`

userInteractionEnabled

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

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

window

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

Class Methods

animateWithDuration:animations:

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

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

Parameters

duration

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

animations

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

Discussion
This method performs the specified animations immediately using the default animation options. The default options are `UIViewAnimationOptionCurveEaseInOut` (page 745) and `UIViewAnimationOptionTransitionNone` (page 745).

Availability
Available in iOS 4.0 and later.

Declared In
UIView.h

animateWithDuration:animations:completion:

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

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

Parameters

duration

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

animations

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

completion

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

Discussion
This method performs the specified animations immediately using the default animation options. The default options are UIViewAnimationOptionCurveEaseInOut (page 745) and UIViewAnimationOptionTransitionNone (page 745).

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

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

Availability
Available in iOS 4.0 and later.

Declared In
UIView.h

animateWithDuration:delay:options:animations:completion:

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

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

Parameters

duration

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

delay

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

options

A mask of options indicating how you want to perform the animations. For a list of valid constants, see `UIViewAnimationOptions` (page 743).

animations

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

completion

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

Discussion

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

Availability

Available in iOS 4.0 and later.

Declared In

`UIView.h`

areAnimationsEnabled

Returns a Boolean value indicating whether animations are enabled.

```
+ (BOOL)areAnimationsEnabled
```

Return Value

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

Availability

Available in iOS 2.0 and later.

See Also

`+ setAnimationsEnabled:` (page 717)

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 719) and `setAnimationDidStopSelector:` (page 714) methods.

context

Additional application-supplied information that is passed to the animation delegate messages—the selectors set using the `setAnimationWillStartSelector:` (page 719) and `setAnimationDidStopSelector:` (page 714) 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 709) to begin and the `commitAnimations` (page 710) class method to end an animation block.

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

Availability

Available in iOS 2.0 and later.

See Also

`+ commitAnimations` (page 710)

`+ setAnimationWillStartSelector:` (page 719)

`+ setAnimationDidStopSelector:` (page 714)

`+ setAnimationDelegate:` (page 713)

Related Sample Code

AddMusic

KeyboardAccessory

ScrollViewSuite

SimpleGestureRecognizers

WiTap

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 711) for how to start animations while others are in progress.

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

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 709)

Related Sample Code
AddMusic
KeyboardAccessory
ScrollViewSuite
SimpleGestureRecognizers
WiTap

Declared In
`UIView.h`

layerClass

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

```
+ (Class)layerClass
```

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

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

Availability
Available in iOS 2.0 and later.

See Also
`@property layer` (page 703)

Related Sample Code
aurioTouch
GLSprite
SpeakHere

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 709) class method to start and the commitAnimations (page 710) class method to end an animation block. The default value is NO.

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

Availability

Available in iOS 2.0 and later.

See Also

+ beginAnimations:context: (page 709)
+ commitAnimations (page 710)
+ setAnimationStartDate: (page 717)
+ setAnimationDuration: (page 715)
+ setAnimationDelay: (page 713)
+ setAnimationCurve: (page 712)
+ setAnimationRepeatCount: (page 716)
+ setAnimationRepeatAutoreverses: (page 716)

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 709) class method to start and the commitAnimations (page 710) class method to end an animation block. The default value of the animation curve is UIViewAnimationCurveEaseInOut (page 739).

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

Availability

Available in iOS 2.0 and later.

See Also

+ beginAnimations:context: (page 709)
+ commitAnimations (page 710)

+ setAnimationStartDate: (page 717)
+ setAnimationDuration: (page 715)
+ setAnimationDelay: (page 713)
+ setAnimationRepeatCount: (page 716)
+ setAnimationRepeatAutoreverses: (page 716)
+ setAnimationBeginsFromCurrentState: (page 711)

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 709) class method to start and the commitAnimations (page 710) class method to end an animation block. The default value of the animation delay is 0.0.

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

Availability
Available in iOS 2.0 and later.

See Also
+ beginAnimations:context: (page 709)
+ commitAnimations (page 710)
+ setAnimationStartDate: (page 717)
+ setAnimationDuration: (page 715)
+ setAnimationCurve: (page 712)
+ setAnimationRepeatCount: (page 716)
+ setAnimationRepeatAutoreverses: (page 716)
+ setAnimationBeginsFromCurrentState: (page 711)

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 719) and `setAnimationDidStopSelector:` (page 714) methods.

Discussion

This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 709) class method to start and the `commitAnimations` (page 710) class method to end an animation block. The default value is `nil`.

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

Availability

Available in iOS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 709)

`+ commitAnimations` (page 710)

`+ setAnimationWillStartSelector:` (page 719)

`+ setAnimationDidStopSelector:` (page 714)

Declared In

`UIView.h`

setAnimationDidStopSelector:

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

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

Parameters

selector

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

`animationID`
An `NSString` containing an optional application-supplied identifier. This is the identifier that is passed to the `beginAnimations:context:` (page 709) method. This argument can be `nil`.

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

`context`
An optional application-supplied context. This is the context data passed to the `beginAnimations:context:` (page 709) method. This argument can be `nil`.

Discussion

This method does nothing if invoked outside of an animation block. Use the `beginAnimations:context:` (page 709) class method to start and the `commitAnimations` (page 710) class method to end an animation block. The default value is `NULL`.

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

Availability
Available in iOS 2.0 and later.

See Also
+ beginAnimations:context: (page 709)
+ commitAnimations (page 710)
+ setAnimationDelegate: (page 713)
+ setAnimationWillStartSelector: (page 719)

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 709) class method to start and the commitAnimations (page 710) class method to end an animation block. The default value is 0.2.

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

Availability
Available in iOS 2.0 and later.

See Also
+ beginAnimations:context: (page 709)
+ commitAnimations (page 710)
+ setAnimationStartDate: (page 717)
+ setAnimationDelay: (page 713)
+ setAnimationCurve: (page 712)
+ setAnimationRepeatCount: (page 716)
+ setAnimationRepeatAutoreverses: (page 716)
+ setAnimationBeginsFromCurrentState: (page 711)

Related Sample Code
AddMusic
KeyboardAccessory
ScrollViewSuite
SimpleGestureRecognizers

WiTap

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 716) 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 709) class method to start and the commitAnimations (page 710) class method to end an animation block. The default value is NO.

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

Availability
Available in iOS 2.0 and later.

See Also
+ beginAnimations:context: (page 709)
+ commitAnimations (page 710)
+ setAnimationStartDate: (page 717)
+ setAnimationDuration: (page 715)
+ setAnimationDelay: (page 713)
+ setAnimationCurve: (page 712)
+ setAnimationRepeatCount: (page 716)
+ setAnimationBeginsFromCurrentState: (page 711)

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 709) class method to start and the `commitAnimations` (page 710) class method to end an animation block. By default, animations don't repeat.

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

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 709)
`+ commitAnimations` (page 710)
`+ setAnimationStartDate:` (page 717)
`+ setAnimationDuration:` (page 715)
`+ setAnimationDelay:` (page 713)
`+ setAnimationCurve:` (page 712)
`+ setAnimationRepeatAutoreverses:` (page 716)
`+ setAnimationBeginsFromCurrentState:` (page 711)

Declared In
`UIView.h`

setAnimationsEnabled:

Sets whether animations are enabled.

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

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
`+ areAnimationsEnabled` (page 709)

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 709) class method to start and the `commitAnimations` (page 710) class method to end an animation block.

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

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

Availability

Available in iOS 2.0 and later.

See Also

`+ beginAnimations:context:` (page 709)
`+ commitAnimations` (page 710)
`+ setAnimationDuration:` (page 715)
`+ setAnimationDelay:` (page 713)
`+ setAnimationCurve:` (page 712)
`+ setAnimationRepeatCount:` (page 716)
`+ setAnimationRepeatAutoreverses:` (page 716)
`+ setAnimationBeginsFromCurrentState:` (page 711)

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 743).

view

The view to apply the transition to.

cache

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

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

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

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

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

Availability
Available in iOS 2.0 and later.

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 709) 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 709) class method to start and the `commitAnimations` (page 710) class method to end an animation block.

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

Availability
Available in iOS 2.0 and later.

See Also
`+ beginAnimations:context:` (page 709)
`+ commitAnimations` (page 710)
`+ setAnimationDelegate:` (page 713)
`+ setAnimationDidStopSelector:` (page 714)

Declared In
`UIView.h`

transitionFromView:toView:duration:options:completion:

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

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

Parameters

fromView

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

toView

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

duration

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

options

A mask of options indicating how you want to perform the animations. For a list of valid constants, see UIViewAnimationOptions (page 743).

completion

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

Discussion

This method provides a simple way to transition from the view in the *fromView* parameter to the view in the *toView* parameter. By default, the view in *fromView* is replaced in the view hierarchy by the view in *toView*. If both views are already part of your view hierarchy, you can include the UIViewAnimationOptionShowHideTransitionViews (page 745) option in the *options* parameter to simply hide or show them.

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

Availability

Available in iOS 4.0 and later.

Declared In

UIView.h

transitionWithView:duration:options:animations:completion:

Creates a transition animation for the specified container view.

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

Parameters

view

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

duration

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

options

A mask of options indicating how you want to perform the animations. For a list of valid constants, see `UIViewAnimationOptions` (page 743).

animations

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

completion

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

Discussion

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

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

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

Availability

Available in iOS 4.0 and later.

Declared In

`UIView.h`

Instance Methods

addGestureRecognizer:

Attaches a gesture recognizer to the receiving view.

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

Parameters

gestureRecognizer

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

Discussion

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

Availability

Available in iOS 3.2 and later.

See Also

`- removeGestureRecognizer:` (page 733)

`@property gestureRecognizers` (page 702)

Related Sample Code

ScrollViewSuite

Declared In

`UIView.h`

addSubview:

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

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

Discussion

This method also sets the receiver as the next responder of *view*. The receiver retains *view*. If you use `removeFromSuperview` (page 733) to remove *view* from the view hierarchy, *view* is released. If you want to keep using *view* after removing it from the view hierarchy (if, for example, you are swapping through a number of views), you must retain it before invoking `removeFromSuperview`.

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

Availability

Available in iOS 2.0 and later.

See Also

`- insertSubview:atIndex:` (page 730)

`- insertSubview:aboveSubview:` (page 729)

`- insertSubview:belowSubview:` (page 730)

`- exchangeSubviewAtIndex:withSubviewAtIndex:` (page 727)

Related Sample Code

MoviePlayer

ScrollViewSuite

WiTap

Declared In

`UIView.h`

bringSubviewToFront:

Moves the specified subview to the front of its siblings.

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

Parameters

view

The subview to move to the front.

Availability
Available in iOS 2.0 and later.

See Also
- sendSubviewToBack: (page 734)

Related Sample Code
ScrollViewSuite

Declared In
UIView.h

convertPoint:fromView:

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

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

Parameters

point

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

view

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

Return Value
The point converted to the coordinate system of the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- convertPoint:toView: (page 723)
- convertRect:toView: (page 725)
- convertRect:fromView: (page 724)

Declared In
UIView.h

convertPoint:toView:

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

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

Parameters

point

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

view

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

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

Availability
Available in iOS 2.0 and later.

See Also
- `convertPoint:fromView:` (page 723)
- `convertRect:toView:` (page 725)
- `convertRect:fromView:` (page 724)

Related Sample Code
ScrollViewSuite

Declared In
`UIView.h`

convertRect:fromView:

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

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

Parameters

rect

The rectangle in `view`'s coordinate system.

view

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

Return Value
The converted rectangle.

Availability
Available in iOS 2.0 and later.

See Also
- `convertPoint:toView:` (page 723)
- `convertPoint:fromView:` (page 723)
- `convertRect:toView:` (page 725)

Declared In
`UIView.h`

convertRect:toView:

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

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

Parameters

rect

A rectangle in the receiver's coordinate system.

view

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

Return Value
The converted rectangle.

Availability
Available in iOS 2.0 and later.

See Also
- `convertPoint:toView:` (page 723)
- `convertPoint:fromView:` (page 723)
- `convertRect:fromView:` (page 724)

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 722).

Availability
Available in iOS 2.0 and later.

See Also
- `willRemoveSubview:` (page 738)
- `addSubview:` (page 722)

Declared In
`UIView.h`

didMoveToSuperview

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

```
- (void)didMoveToSuperview
```

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

Availability
Available in iOS 2.0 and later.

See Also
`- willMoveToSuperview:` (page 737)

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

Availability
Available in iOS 2.0 and later.

See Also
`- willMoveToWindow:` (page 738)

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 704) 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 1040) function to get the current graphics context for drawing that also has the coordinate origin in the upper-left corner. Do not retain the graphics context since it can change between calls to the `drawRect:` method.

Availability
Available in iOS 2.0 and later.

See Also
- `setNeedsDisplay` (page 734)
- `setNeedsDisplayInRect:` (page 735)
- `@property contentMode` (page 700)

Declared In
`UIView.h`

endEditing:

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

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

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

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UITextField.h`

exchangeSubviewAtIndex:withSubviewAtIndex:

Exchanges the subviews in the receiver at the given indices.

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

Parameters

index1

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

index2

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

Availability
Available in iOS 2.0 and later.

See Also
- `addSubview:` (page 722)
- `insertSubview:atIndex:` (page 730)
- `insertSubview:aboveSubview:` (page 729)
- `insertSubview:belowSubview:` (page 730)

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 732) message to each subview to determine which subview should receive a touch event. If `pointInside:withEvent:` (page 732) returns `YES`, then the subview's hierarchy is traversed; otherwise, its branch of the view hierarchy is ignored. You rarely need to invoke this method, but you might override it to hide touch events from subviews.

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

Availability
Available in iOS 2.0 and later.

See Also
- `pointInside:withEvent:` (page 732)

Declared In
`UIView.h`

initWithFrame:

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

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

Parameters

aRect

The frame rectangle for the view, measured in points. The origin of the frame is relative to the superview in which you plan to add it. This method uses the frame rectangle to set the `center` (page 698) and `bounds` (page 698) properties accordingly.

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

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

insertSubview:aboveSubview:

Inserts a view above another view in the view hierarchy.

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

Parameters

view

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

siblingSubview

The sibling view that will be behind the inserted view.

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

Availability
Available in iOS 2.0 and later.

See Also
- `addSubview:` (page 722)
- `insertSubview:atIndex:` (page 730)
- `insertSubview:belowSubview:` (page 730)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 727)

Declared In
`UIView.h`

insertSubview:atIndex:

Inserts a subview at the specified index.

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

Parameters

view

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

index

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

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

Availability
Available in iOS 2.0 and later.

See Also
- `addSubview:` (page 722)
- `insertSubview:aboveSubview:` (page 729)
- `insertSubview:belowSubview:` (page 730)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 727)

Declared In
`UIView.h`

insertSubview:belowSubview:

Inserts a view below another view in the view hierarchy.

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

Parameters

view

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
- `addSubview:` (page 722)
- `insertSubview:atIndex:` (page 730)
- `insertSubview:aboveSubview:` (page 729)
- `exchangeSubviewAtIndex:withSubviewAtIndex:` (page 727)

Declared In
`UIView.h`

isDescendantOfView:

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

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

Parameters
view
The view to test for subview relationship within the view hierarchy.

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

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

layoutIfNeeded

Lays out the subviews if needed.

```
- (void)layoutIfNeeded
```

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

Availability
Available in iOS 2.0 and later.

See Also
- `setNeedsLayout` (page 735)
- `layoutSubviews` (page 732)

Declared In
`UIView.h`

layoutSubviews

Lays out subviews.

```
- (void)layoutSubviews
```

Discussion
Overridden by subclasses to layout subviews when `layoutIfNeeded` (page 731) is invoked. The default implementation of this method does nothing.

Availability
Available in iOS 2.0 and later.

See Also
- `setNeedsLayout` (page 735)
- `layoutIfNeeded` (page 731)

Related Sample Code
aurioTouch
GLSprite
ScrollViewSuite
SpeakHere

Declared In
`UIView.h`

pointInside:withEvent:

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

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

Parameters

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
`- hitTest:withEvent:` (page 728)

Declared In
`UIView.h`

removeFromSuperview

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

```
- (void)removeFromSuperview
```

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

Never invoke this method while displaying.

Availability
Available in iOS 2.0 and later.

See Also
`@property superview` (page 705)
`@property subviews` (page 705)

Related Sample Code
ScrollViewSuite
SpeakHere

Declared In
`UIView.h`

removeGestureRecognizer:

Detaches a gesture recognizer from the receiving view.

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

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

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

Availability
Available in iOS 3.2 and later.

See Also
- `addGestureRecognizer:` (page 721)
- `@property gestureRecognizers` (page 702)

Declared In
`UIView.h`

sendSubviewToBack:

Moves the specified subview to the back of its siblings.

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

Parameters

view

The subview to move to the back.

Availability
Available in iOS 2.0 and later.

See Also
- `bringSubviewToFront:` (page 723)

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 726) 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 734) message. Any `UIView` objects marked as needing display are automatically redisplayed when the application returns to the run loop.

Availability
Available in iOS 2.0 and later.

See Also
- `drawRect:` (page 726)
- `setNeedsDisplayInRect:` (page 735)
- `@property contentMode` (page 700)

Related Sample Code
GKRocket
SpeakHere

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 726) 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 734) method to mark a view as needing display.

Availability
Available in iOS 2.0 and later.

See Also
- `drawRect:` (page 726)
- `setNeedsDisplay` (page 734)
- `@property contentMode` (page 700)

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 731) lays out the subviews; otherwise, it does not.

Availability
Available in iOS 2.0 and later.

See Also
- `layoutIfNeeded` (page 731)
- `layoutSubviews` (page 732)

Related Sample Code
ScrollViewSuite

Declared In
`UIView.h`

sizeThatFits:

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

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

Parameters

size

The current size of the receiver.

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

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

This method does not resize the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `sizeToFit` (page 736)
`@property frame` (page 701)
`@property bounds` (page 698)

Declared In
`UIView.h`

sizeToFit

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

```
- (void)sizeToFit
```

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

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

Availability
Available in iOS 2.0 and later.

See Also
- sizeThatFits: (page 736)

Related Sample Code
BonjourWeb
WiTap

Declared In
UIView.h

viewWithTag:

Returns the view with the specified tag.

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

Parameters
tag
The tag used to search for the view.

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

Availability
Available in iOS 2.0 and later.

See Also
@property tag (page 705)

Declared In
UIView.h

willMoveToSuperview:

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

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
- didMoveToSuperview (page 726)

Declared In
UIView.h

willMoveToWindow:

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

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

Parameters

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

Discussion

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

Availability

Available in iOS 2.0 and later.

See Also

- `didMoveToWindow` (page 726)

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 733) message or *subview* is removed from the receiver because it is being added to another view.

Availability

Available in iOS 2.0 and later.

See Also

- `didAddSubview:` (page 725)
- `addSubview:` (page 722)

Declared In

`UIView.h`

Constants

UIViewAnimationCurve

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

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

Constants

`UIViewAnimationCurveEaseInOut`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationCurveEaseIn`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationCurveEaseOut`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationCurveLinear`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

Availability

Available in iOS 2.0 and later.

Declared In

`UIView.h`

UIViewContentMode

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

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

Constants

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewContentModeRedraw`
Redisplays the view when the bounds change by invoking the `setNeedsDisplay` (page 734) method.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

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

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 2.0 and later.

  Declared in `UIView.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIView.h`

UIViewAutoresizing

Specifies how a view is automatically resized.

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

Constants

`UIViewAutoresizingNone`

The view does not resize.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleLeftMargin`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleWidth`

The view resizes by expanding or shrinking its width.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleRightMargin`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleTopMargin`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleHeight`

The view resizes by expanding or shrinking its height.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAutoresizingFlexibleBottomMargin`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

Availability

Available in iOS 2.0 and later.

Declared In

`UIView.h`

UIViewAnimationTransition

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

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

Constants

`UIViewAnimationTransitionNone`

No transition specified.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionFlipFromLeft`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionFlipFromRight`

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

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionCurlUp`

A transition that curls a view up from the bottom.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

`UIViewAnimationTransitionCurlDown`

A transition that curls a view down from the top.

Available in iOS 2.0 and later.

Declared in `UIView.h`.

Availability

Available in iOS 2.0 and later.

Declared In

`UIView.h`

UIViewAnimationOptions

Specifies options for animating views with blocks.

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

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

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

Constants

`UIViewAnimationOptionLayoutSubviews`

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

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionAllowUserInteraction`

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

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionBeginFromCurrentState`

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

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionRepeat`

Repeat the animation indefinitely.

Available in iOS 4.0 and later.

Declared in `UIView.h`.

`UIViewAnimationOptionAutoreverse`

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

Available in iOS 4.0 and later.

Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

`UIViewAnimationOptionTransitionNone`
: No transition is specified.

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

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

  Available in iOS 4.0 and later.

  Declared in `UIView.h`.

Availability
Available in iOS 4.0 and later.

Declared In
`UIView.h`

UIViewController Class Reference

Inherits from	UIResponder : NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIViewController.h UINavigationController.h UITabBarController.h
Companion guide	View Controller Programming Guide for iOS
Related sample code	AddMusic CryptoExercise MoviePlayer MultipleDetailViews ScrollViewSuite

Overview

The `UIViewController` class provides the fundamental view-management model for iPhone applications. The basic view controller class supports the presentation of an associated view in addition to basic support for managing modal views and rotating views in response to device orientation changes. Subclasses such as `UINavigationController` and `UITabBarController` provide additional behavior for managing complex hierarchies of view controllers and views.

You use each instance of `UIViewController` to manage a full-screen view. For a simple view controller, this entails managing the view hierarchy responsible for presenting your application content. A typical view hierarchy consists of a root view—a reference to which is available in the `view` (page 761) property of this class—and one or more subviews presenting the actual content. In the case of navigation and tab bar controllers, the view controller manages not only the high-level view hierarchy (which provides the navigation controls) but also one or more additional view controllers that handle the presentation of the application content.

Note: You should not use view controllers to manage views that fill only a part of their window—that is, only part of the area defined by the application content rectangle. If you want to have an interface composed of several smaller views, embed them all in a single root view and manage that view with your view controller.

Because view controllers are tightly bound to the views they manage, they are also part of the responder chain used to handle events. View controllers are themselves descendants of the `UIResponder` class and are inserted into the responder chain between the managed view and its superview. Thus, if the view managed by a view controller does not handle an event, it passes the event to its view controller, which then has the option of handling the event or forwarding it to the view's superview.

The `UIViewController` class provides automatic support for rotating the views of the view controller in response to changes to the orientation of the device. As part of the autorotation behavior, the view controller slides any tab bars or navigation bars out of view, reorients the underlying views, and slides the bars back into position. If the autoresizing properties of your view are already configured, this behavior is essentially free. However, you can also customize the rotation behavior by specifying additional animations you want to be performed.

View controllers are fundamental to the design of most iPhone applications. The sections that follow provide basic information about using the methods and properties of the `UIViewController` class. For additional information about using view controllers to build and manage your application's user interface, see *View Controller Programming Guide for iOS*.

Using View Controllers With Other View Controllers

View controllers rarely operate in isolation. If your application uses a navigation or tab bar controller, or if your application presents views modally, then it typically has several view controllers interacting with each other to implement those navigation features.

A navigation controller interface consists of a `UINavigationController` object and one or more custom view controllers to present the different navigation screens. As the user selects new items in the interface, your code pushes new view controllers onto the navigation stack. Each new view controller then displays a new screen's worth of content. To manage the removal of view controllers from the stack, each view controller has an associated navigation item, which allows navigation back to that item. You can configure the navigation item for a given view controller by modifying its `navigationItem` (page 757) property.

A tab bar controller interface consists of a `UITabBarController` object and one or more view controllers for each tab. The root view controller for each tab can configure the information displayed on its tab by modifying the object in its `tabBarItem` (page 760) property.

In iOS, you can display views modally by presenting the controller for the modal view from your current view controller. When you present a view modally using the `presentModalViewController:animated:` (page 767) method, the view controller animates the appearance of the view using the technique you specify. (You can specify the desired technique by setting the `modalTransitionStyle` (page 756) property.) At the same time, the method creates a parent-child relationship between the current view controller and the modal view controller.

Because the relationships between view controllers can grow quite complex, each view controller object has properties that indicate whether it is managed by other view controllers. You can check the `tabBarController` (page 760) or `navigationController` (page 757) properties of a view controller to see if it is embedded inside of a tab bar or navigation bar interface. You can also find the controller's immediate parent controller using the `parentViewController` (page 759) property.

For more information about the relationships between view controllers and how you build complex navigation interfaces, see *View Controller Programming Guide for iOS*.

Subclassing Notes

In a typical iPhone application, there is usually at least one custom subclass of `UIViewController` and more often there are several. Because the amount of available screen space on iOS-based devices is limited, interfaces must typically be divided into one or more screen's worth of information. The views used to present each distinct screen are then managed by one of your `UIViewController` subclasses.

The job of each view controller object is two-fold. Because it is part of your application's controller layer, a view controller is responsible for coordinating interactions between your application's visual presentation (your custom views) and your application's data model (your custom objects). A view controller is also responsible for handling changes to the views that comprise its view layer. For example, when the user rotates a device from a portrait to a landscape orientation, the view controller is responsible for reorienting the views accordingly. Fortunately, the default behavior of the `UIViewController` class handles much of the work needed to manage your view layer. All you have to do is specify the initial set of views and their default behaviors. After that, you can focus on the interactions between those views and your data model.

When you define a new subclass of `UIViewController`, you must specify the views to be managed by the controller. There are two mutually exclusive ways to specify these views: manually or using a nib file. If you specify the views manually, you must implement the `loadView` (page 767) method and use it to assign a root view object to the `view` (page 761) property. If you specify views using a nib file, you must not override `loadView` (page 767) but should instead create a nib file in Interface Builder and then initialize your view controller object using the `initWithNibName:bundle:` method. Creating views using a nib file is often simpler because you can use the Interface Builder application to create and configure your views graphically (as opposed to programmatically). Both techniques have the same end result, however, which is to create the appropriate set of views and expose them through the `view` (page 761) property.

Important: A view controller is the sole owner of its view and any associated subviews. It is responsible for creating those views and for releasing them at the appropriate times, including during low-memory conditions and when the view controller itself is released. If you store your views in a nib file, each view controller object creates its own copy of the view in that nib file. However, if you create your views manually, you should never use the same view objects with multiple view controllers.

When creating the views for your view hierarchy, you should always set the autoresizing properties of your views. When a view controller is displayed on screen, its root view is typically resized to fit the available space, which can vary depending on the window's current orientation and the presence of other interface elements such as the status bar. You can configure the autoresizing properties in Interface Builder using the inspector window or programmatically by modifying the `autoresizesSubviews` (page 696) and `autoresizingMask` (page 697) properties of each view. Setting these properties is also important if your view controller supports both portrait and landscape orientations. During an orientation change, the system uses these properties to reposition and resize the views automatically to match the new orientation.

Memory Management

Memory is a critical resource in iOS, and view controllers provide built-in support for reducing their memory footprint at critical times. The `UIViewController` class provides some automatic handling of low-memory conditions through its `didReceiveMemoryWarning` (page 763) method, which releases unneeded memory.

Prior to iOS 3.0, this method was the only way to release additional memory associated with your custom view controller class but in iOS 3.0 and later, the `viewDidUnload` (page 772) method may be a more appropriate place for most needs.

When a low-memory warning occurs, the `UIViewController` class purges its views if it knows it can reload or recreate them again later. If this happens, it also calls the `viewDidUnload` method to give your code a chance to relinquish ownership of any objects that are associated with your view hierarchy, including objects loaded with the nib file, objects created in your `viewDidLoad` (page 772) method, and objects created lazily at runtime and added to the view hierarchy. Typically, if your view controller contains outlets (properties or raw variables that contain the `IBOutlet` (page 1018) keyword), you should use the `viewDidUnload` method to relinquish ownership of those outlets or any other view-related data that you no longer need.

For general information and guidance about memory management practices in iOS, see *Memory Management Programming Guide*.

Handling View Rotations

By default, the `UIViewController` class displays views in portrait mode only. To support additional orientations, you must override the `shouldAutorotateToInterfaceOrientation:` (page 770) method and return `YES` for any orientations your subclass supports. If the autoresizing properties of your views are configured correctly, that may be all you have to do. However, the `UIViewController` class provides additional hooks for you to implement additional behaviors as needed.

To temporarily turn off features that are not needed or might otherwise cause problems during the orientation change, you can override the `willRotateToInterfaceOrientation:duration:` (page 777) method and perform the needed actions there. You can then override the `didRotateFromInterfaceOrientation:` (page 764) method and use it to reenable those features once the orientation change is complete.

If you want to perform custom animations during an orientation change, you can do so in one of two ways. Orientation changes used to occur in two steps, with notifications occurring at the beginning, middle, and end points of the rotation. However, in iOS 3.0, support was added for performing orientation changes in one step. Using a one-step orientation change tends to be faster than the older two-step process and is generally recommended for any new code.

To add animations for a one-step orientation change, override the `willAnimateRotationToInterfaceOrientation:duration:` (page 775) method and perform your animations there. To use the older two-step method, override one or both of the `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 774) and `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 776) methods to configure your animations before each step. You must choose only one technique and override just the methods associated with that technique. If you override both sets of methods, the system uses the one-step rotation methods by default.

Tasks

Creating a View Controller Using Nib Files

`- initWithNibName:bundle:` (page 766)
Returns a newly initialized view controller with the nib file in the specified bundle.

`nibName` (page 758) *property*
Return the name of the receiver's nib file, if one was specified. (read-only)

`nibBundle` (page 758) *property*
Return the name of the receiver's nib bundle if it exists. (read-only)

Managing the View

`view` (page 761) *property*
The view that the controller manages.

- `loadView` (page 767)
Creates the view that the controller manages.

- `viewDidLoad` (page 772)
Called after the controller's view is loaded into memory.

- `viewDidUnload` (page 772)
Called when the controller's view is released from memory.

- `isViewLoaded` (page 766)
Returns a Boolean value indicating whether the view is currently loaded into memory.

`title` (page 760) *property*
A localized string that represents the view that this controller manages.

`modalInPopover` (page 755) *property*
A Boolean value indicating whether the view controller should be presented modally by a popover.

`contentSizeForViewInPopover` (page 754) *property* Deprecated in iOS 3.0
The size of the view controller's view while displayed in a popover.

Responding to View Events

- `viewWillAppear:` (page 773)
Notifies the view controller that its view is about to be become visible.

- `viewDidAppear:` (page 771)
Notifies the view controller that its view was added to a window.

- `viewWillDisappear:` (page 774)
Notifies the view controller that its view is about to be dismissed, covered, or otherwise hidden from view.

- `viewDidDisappear:` (page 771)
Notifies the view controller that its view was dismissed, covered, or otherwise hidden from view.

Configuring the View's Layout Behavior

`wantsFullScreenLayout` (page 762) *property*
A Boolean value indicating whether the view should overlap the status bar.

Configuring the View Rotation Settings

`interfaceOrientation` (page 755) *property*
The current orientation of the interface. (read-only)

- `shouldAutorotateToInterfaceOrientation:` (page 770)
Returns a Boolean value indicating whether the view controller supports the specified orientation.

- `rotatingHeaderView` (page 769)
Returns the header view that slides in and out before and after the user interface rotates.

- `rotatingFooterView` (page 768)
Returns the footer view that slides in and out before and after the user interface rotates.

Responding to View Rotation Events

- `willRotateToInterfaceOrientation:duration:` (page 777)
Sent to the view controller just before the user interface begins rotating.

- `willAnimateRotationToInterfaceOrientation:duration:` (page 775)
Sent to the view controller before performing a one-step user interface rotation.

- `didRotateFromInterfaceOrientation:` (page 764)
Sent to the view controller after the user interface rotates.

- `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 774)
Sent to the view controller before performing the first half of a user interface rotation.

- `didAnimateFirstHalfOfRotationToInterfaceOrientation:` (page 763)
Sent to the view controller after the completion of the first half of the user interface rotation.

- `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 776)
Sent to the view controller before the second half of the user interface rotates.

Handling Memory Warnings

- `didReceiveMemoryWarning` (page 763)
Sent to the view controller when the application receives a memory warning.

Getting Other Related View Controllers

`parentViewController` (page 759) *property*
The parent of the current view controller. (read-only)

`searchDisplayController` (page 759) *property*
The search display controller associated with the view controller. (read-only)

`splitViewController` (page 759) *property*
The parent or ancestor that is a split view controller. (read-only)

`modalViewController` (page 757) *property*
The controller for the active modal view—that is, the view that is temporarily displayed on top of the view managed by the receiver. (read-only)

`navigationController` (page 757) *property*
A parent or ancestor that is a navigation controller. (read-only)

`tabBarController` (page 760) *property*
A parent or ancestor that is a tab bar controller. (read-only)

Presenting Modal Views

- `presentModalViewController:animated:` (page 767)
Presents a modal view managed by the given view controller to the user.

- `dismissModalViewControllerAnimated:` (page 764)
Dismisses the modal view controller that was presented by the receiver.

`modalPresentationStyle` (page 756) *property*
The presentation style for modally presented view controllers.

`modalTransitionStyle` (page 756) *property* Deprecated in iOS 3.0
The transition style to use when presenting the current view controller modally.

Configuring a Navigation Interface

`navigationItem` (page 757) *property*
The navigation item used to represent the view controller. (read-only)

`editing` (page 754) *property*
A Boolean value indicating whether the view controller currently allows the user to edit the view contents.

- `setEditing:animated:` (page 769)
Sets whether the view controller shows an editable view.

- `editButtonItem` (page 765)
Returns a bar button item that toggles its title and associated state between Edit and Done.

`hidesBottomBarWhenPushed` (page 755) *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 the Navigation Controller's Toolbar

- `setToolbarItems:animated:` (page 770)
Sets the toolbar items to be displayed along with the view controller.

`toolbarItems` (page 761) *property*
The toolbar items associated with the view controller.

Configuring Tab Bar Items

`tabBarItem` (page 760) *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 Programming Language*.

contentSizeForViewInPopover

The size of the view controller's view while displayed in a popover.

```
@property(nonatomic, readwrite) CGSize contentSizeForViewInPopover
```

Discussion
This property contains the desired size for the view controller when it is displayed in a popover. By default, the width is set to 320 points and the height is set to 1100 points. You can change these values as needed.

The recommended width for popovers is 320 points. If needed, you can return a width value as large as 600 points but doing so is not recommended.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

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 765) method. Set the `editing` property to the initial state of your view. Use the `setEditing:animated:` (page 769) method as an action method to animate the transition of this state if the view is already displayed.

Availability
Available in iOS 2.0 and later.

See Also
- `setEditing:animated:` (page 769)
- `editButtonItem` (page 765)

Related Sample Code
BonjourWeb

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 iOS 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 `UIInterfaceOrientation` (page 128).

Availability
Available in iOS 2.0 and later.

See Also
- `- willRotateToInterfaceOrientation:duration:` (page 777)
- `- willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 774)
- `- willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 776)
- `- didRotateFromInterfaceOrientation:` (page 764)

Related Sample Code
CryptoExercise

Declared In
`UIViewController.h`

modalInPopover

A Boolean value indicating whether the view controller should be presented modally by a popover.

```
@property(nonatomic,readwrite,getter=isModalInPopover) BOOL modalInPopover
```

Discussion
The default value of this property is `NO`. Setting it to `YES` causes an owning popover controller to disallow interactions outside this view controller while it is displayed. You can use this behavior to ensure that the popover is not dismissed by taps outside the popover controller.

Availability
Available in iOS 3.2 and later.

Declared In
`UIPopoverController.h`

modalPresentationStyle

The presentation style for modally presented view controllers.

```
@property(nonatomic, assign) UIModalPresentationStyle modalPresentationStyle
```

Discussion
The presentation style determines how a modally presented view controller is displayed on the screen. On iPhone and iPod touch devices, modal view controllers are always presented full-screen, but on iPad devices there are several different presentation options. For a list of possible presentation styles, and their compatibility with the available transition styles, see the "`UIModalPresentationStyle`" (page 778) constant descriptions.

Availability
Available in iOS 3.2 and later.

Declared In
`UIViewController.h`

modalTransitionStyle

The transition style to use when presenting the current view controller modally.

```
@property(nonatomic, assign) UIModalTransitionStyle modalTransitionStyle
```

Discussion
This property affects the way the current view controller is presented when it is presented using the `presentModalViewController:animated:` method. To change the transition type, you must set this property before presenting the view controller. The default value for this property is `UIModalTransitionStyleCoverVertical`.

For a list of possible transition styles, and their compatibility with the available presentation styles, see the "`UIModalTransitionStyle`" (page 777) constant descriptions.

Availability
Available in iOS 3.0 and later.

See Also
`- presentModalViewController:animated:` (page 767)

Related Sample Code
AddMusic

Declared In
`UIViewController.h`

modalViewController

The controller for the active modal view—that is, the 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 iOS 2.0 and later.

See Also
`@property parentViewController` (page 759)

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 iOS 2.0 and later.

See Also
`@property tabBarController` (page 760)

Related Sample Code
ToolbarSearch

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.

You should avoid tying the creation of bar button items in your navigation item to the creation of your view controller's view. The navigation item of a view controller may be retrieved independently of the view controller's view. For example, when pushing two view controllers onto a navigation stack, the topmost view controller becomes visible, but the other view controller's navigation item may be retrieved in order to present its back button. To ensure the navigation item is configured, you can override this property and add code to load the bar button items there or load the items in your view controller's initialization code.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb

Declared In
`UINavigationController.h`

nibBundle

Return the name of the receiver's nib bundle if it exists. (read-only)

```
@property(nonatomic, readonly, retain) NSBundle *nibBundle
```

Availability
Available in iOS 2.0 and later.

See Also
`- initWithNibName:bundle:` (page 766)
`@property nibName` (page 758)

Declared In
`UIViewController.h`

nibName

Return the name of the receiver's nib file, if one was specified. (read-only)

```
@property(nonatomic, readonly, copy) NSString *nibName
```

Discussion
This property contains the value specified at initialization time to the `initWithNibName:bundle:` method. The value of this property may be `nil`.

If the value of this property is `nil` and you did not override the `loadView` method in your custom subclass, the view controller looks for a nib file whose name (without the `.nib` extension) matches the name of your view controller class.

Availability
Available in iOS 2.0 and later.

See Also
`- initWithNibName:bundle:` (page 766)
`@property nibBundle` (page 758)

Declared In
`UIViewController.h`

parentViewController

The parent of the current view controller. (read-only)

```
@property(nonatomic, readonly) UIViewController *parentViewController
```

Discussion
Parent view controllers are relevant in navigation, tab bar, and modal view controller hierarchies. In each of these hierarchies, the parent is the object responsible for displaying the current view controller. If you are using a view controller as a standalone object—that is, not as part of a view controller hierarchy—the value in this property is `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`@property modalViewController` (page 757)

Declared In
`UIViewController.h`

searchDisplayController

The search display controller associated with the view controller. (read-only)

```
@property(nonatomic, readonly, retain) UISearchDisplayController
    *searchDisplayController
```

Discussion
This property reflects the value of the `searchDisplayController` outlet that you set in Interface Builder. If you create your search display controller programmatically, this property is set automatically by the search display controller when it is initialized.

Availability
Available in iOS 3.0 and later.

Declared In
`UIViewController.h`

splitViewController

The parent or ancestor that is a split view controller. (read-only)

```
@property(nonatomic, readonly, retain) UISplitViewController *splitViewController
```

Discussion
If the receiver or one of its ancestors is currently embedded inside of a split view controller, this property contains the owning split view controller. This property is `nil` if the view controller is presented modally or is not embedded inside a split view controller.

Availability
Available in iOS 3.2 and later.

Declared In
`UISplitViewController.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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
BonjourWeb
WiTap

Declared In
`UIViewController.h`

toolbarItems

The toolbar items associated with the view controller.

```
@property(nonatomic, retain) NSArray *toolbarItems
```

Discussion
This property contains an array of `UIBarButtonItem` objects and works in conjunction with a `UINavigationController` object. If this view controller is embedded inside a navigation controller interface, and the navigation controller displays a toolbar, this property identifies the items to display in that toolbar.

You can set the value of this property explicitly or use the `setToolbarItems:animated:` method to animate changes to the visible set of toolbar items.

Availability
Available in iOS 3.0 and later.

See Also
`- setToolbarItems:animated:` (page 770)

Declared In
`UINavigationController.h`

view

The view that the controller manages.

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

Discussion
The view stored in this property represents the root view for the view controller's view hierarchy. Whenever you present the view controller on screen (either modally or as part of view controller–based interface), this view is retrieved and displayed in the application window. The default value of this property is `nil`.

If you access this property and its value is currently `nil`, the view controller automatically calls the `loadView` (page 767) method and returns the resulting view. The default `loadView` method attempts to load the view from the nib file associated with the view controller (if any). If your view controller does not have an associated view controller, you should override the `loadView` method and use it to create the root view and all of its subviews.

Each view controller object is the sole owner of its view. You must not associate the same view object with multiple view controller objects.

Because accessing this property can cause the view to be loaded automatically, you can use the `isViewLoaded` property to determine if the view is currently in memory. Unlike this property, the `isViewLoaded` property does not force the loading of the view if it is not currently in memory.

The `UIViewController` class automatically sets this property to `nil` during low-memory conditions and when the view controller itself is finally released.

Availability
Available in iOS 2.0 and later.

See Also
- `loadView` (page 767)
- `isViewLoaded` (page 766)
- `viewDidLoad` (page 772)

Related Sample Code
GKTank
KeyboardAccessory
ScrollViewSuite

Declared In
`UIViewController.h`

wantsFullScreenLayout

A Boolean value indicating whether the view should overlap the status bar.

```
@property(nonatomic, assign) BOOL wantsFullScreenLayout
```

Discussion
When a view controller presents its view, it normally shrinks that view so that its frame does not overlap the device's status bar. Setting this property to `YES` causes the view controller to size its view so that it fills the entire screen, including the area under the status bar. (Of course, for this to happen, the window hosting the view controller must itself be sized to fill the entire screen, including the area underneath the status bar.) You would typically set this property to `YES` in cases where you have a translucent status bar and want your view's content to be visible behind that view.

If this property is `YES`, the view is not resized in a way that would cause it to overlap a tab bar but is resized to overlap translucent toolbars. Regardless of the value of this property, navigation controllers always allow views to overlap translucent navigation bars.

The default value of this property is `NO`, which causes the view to be laid out so it does not overlap the status bar.

Availability
Available in iOS 3.0 and later.

Declared In
`UIViewController.h`

Instance Methods

didAnimateFirstHalfOfRotationToInterfaceOrientation:

Sent to the view controller after the completion of the first half of the user interface rotation.

```
- (void)didAnimateFirstHalfOfRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
```

Parameters

toInterfaceOrientation

The state of the application's user interface orientation after the rotation. The possible values are described in `UIInterfaceOrientation` (page 128).

Discussion

This method is called during two-step rotation animations only. Subclasses can override this method and use it to adjust their views between the first and second half of the animations. This method is called outside of any animation transactions and while any header or footer views are offscreen.

Availability

Available in iOS 2.1 and later.

See Also

- `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 774)
- `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 776)

Declared In

`UIViewController.h`

didReceiveMemoryWarning

Sent to the view controller when the application receives a memory warning.

```
- (void)didReceiveMemoryWarning
```

Discussion

The default implementation of this method checks to see if the view controller can safely release its view. This is possible if the view itself does not have a superview and can be reloaded either from a nib file or using a custom `loadView` method. If the view can be released, this method releases it and calls the `viewDidUnload` method.

You can override this method (as needed) to release any additional memory used by your view controller. If you do, be sure to call the `super` implementation at some point to allow the view controller to release its view. In iOS 3.0 and later, if your view controller holds references to objects in the view hierarchy, you should release those references in the `viewDidUnload` method instead. In earlier versions of iOS, you should continue to release them from this method. See the discussion in the `viewDidUnload` method for information about how to safely release outlets and other objects.

Availability

Available in iOS 2.0 and later.

See Also
- `loadView` (page 767)
- `viewDidUnload` (page 772)

Related Sample Code
AddMusic
GKRocket
GKTank
MoviePlayer
SpeakHere

Declared In
`UIViewController.h`

didRotateFromInterfaceOrientation:

Sent to the view controller after the user interface rotates.

```
- (void)didRotateFromInterfaceOrientation:(UIInterfaceOrientation)fromInterfaceOrientation
```

Parameters
fromInterfaceOrientation
The old orientation of the user interface. The possible values are described in `UIInterfaceOrientation` (page 128).

Discussion
Subclasses may override this method to perform additional actions immediately after the rotation. For example, you might use this method to reenable view interactions, start media playback again, or turn on expensive drawing or live updates. By the time this method is called, the `interfaceOrientation` (page 755) property is already set to the new orientation.

This method is called regardless of whether your code performs one-step or two-step rotations.

Availability
Available in iOS 2.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 777)
- `willAnimateRotationToInterfaceOrientation:duration:` (page 775)

Declared In
`UIViewController.h`

dismissModalViewControllerAnimated:

Dismisses the modal view controller that was presented by the receiver.

```
- (void)dismissModalViewControllerAnimated:(BOOL)animated
```

Parameters

animated

If YES, this method animates the view as it's dismissed; otherwise, it does not. The style of animation is determined by the value in the modalTransitionStyle property of the view controller being dismissed.

Discussion

The parent view controller is responsible for dismissing the modal view controller it presented using the presentModalViewController:animated: method. If you call this method on the modal view controller itself, however, the modal view controller automatically forwards the message to its parent view controller.

If you present several modal view controllers in succession, and thus build a stack of modal view controllers, calling this method on a view controller lower in the stack dismisses its immediate child view controller and all view controllers above that child on the stack. When this happens, only the top-most view is dismissed in an animated fashion; any intermediate view controllers are simply removed from the stack. The top-most view is dismissed using its modal transition style, which may differ from the styles used by other view controllers lower in the stack.

If you want to retain a reference to the receiver's modal view controller, get the value in the modalViewController (page 757) property before calling this method.

Availability

Available in iOS 2.0 and later.

See Also

- presentModalViewController:animated: (page 767)

Related Sample Code

CryptoExercise

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 757) property is set to the returned object, the associated navigation bar displays an Edit button if editing (page 754) is NO and a Done button if editing (page 754) is YES. The default button action invokes the setEditing:animated: (page 769) method.

Availability

Available in iOS 2.0 and later.

See Also

@property editing (page 754)

- setEditing:animated: (page 769)

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 772) method to do so. If this argument is `nil`, the `nibName` (page 758) property is set to `nil`. In this case, you should override the `loadView` (page 767) method to set the `view` (page 761) 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.

If you specify `nil` for the *nibName* parameter and do not override the `loadView` method in your custom subclass, the default view controller behavior is to look for a nib file whose name (without the `.nib` extension) matches the name of your view controller class. If it finds one, the class name becomes the value of the `nibName` property, which results in the corresponding nib file being associated with this view controller.

Availability
Available in iOS 2.0 and later.

See Also
`@property nibName` (page 758)
`@property nibBundle` (page 758)

Related Sample Code
AddMusic
CryptoExercise
MultipleDetailViews

Declared In
`UIViewController.h`

isViewLoaded

Returns a Boolean value indicating whether the view is currently loaded into memory.

```
- (BOOL)isViewLoaded
```

Discussion
Calling this method simply reports whether the view is loaded. Unlike the `view` property, it does not attempt to load the view if it is not already in memory.

Availability
Available in iOS 3.0 and later.

Declared In
UIViewController.h

loadView

Creates the view that the controller manages.

```
- (void)loadView
```

Discussion
You should never call this method directly. The view controller calls this method when the `view` (page 761) property is requested but is currently `nil`. If you create your views manually, you must override this method and use it to create your views. If you use Interface Builder to create your views and initialize the view controller—that is, you initialize the view using the `initWithNibName:bundle:` (page 766) method, set the `nibName` (page 758) and `nibBundle` (page 758) properties directly, or create both your views and view controller in Interface Builder—then you must not override this method.

The default implementation of this method looks for valid nib information and uses that information to load the associated nib file. If no nib information is specified, the default implementation creates a plain `UIView` object and makes it the main view.

If you override this method in order to create your views manually, you should do so and assign the root view of your hierarchy to the `view` (page 761) property. (The views you create should be unique instances and should not be shared with any other view controller object.) Your custom implementation of this method should not call `super`.

If you want to perform any additional initialization of your views, do so in the `viewDidLoad` (page 772) method. In iOS 3.0 and later, you should also override the `viewDidUnload` (page 772) method to release any references to the view or its contents.

Availability
Available in iOS 2.0 and later.

See Also
`@property view` (page 761)
- `viewDidLoad` (page 772)
- `viewDidUnload` (page 772)

Related Sample Code
ScrollViewSuite

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
On iPhone and iPod touch devices, this method presents the view controller using the full screen.

Sets the modalViewController (page 757) property to the specified view controller. Resizes its view and attaches it to the view hierarchy. The view is animated according to the transition style specified in the modalTransitionStyle property of the controller in the *modalViewController* parameter.

Availability
Available in iOS 2.0 and later.

See Also
- dismissModalViewControllerAnimated: (page 764)

Related Sample Code
AddMusic
CryptoExercise

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 navigation bar and the footer view is the tab bar. If the default behavior is not desired, subclasses should override this method to return the alternate view (that is already in the receiver's view hierarchy) that is used as the footer view.

Availability
Available in iOS 2.0 and later.

See Also
- shouldAutorotateToInterfaceOrientation: (page 770)
- rotatingHeaderView (page 769)

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 or nil if there is no header view. If the current view controller is a tab bar controller, this method returns the header view of the view controller in the selected tab. If the current view controller is a navigation controller, this method returns the associated navigation bar.

Discussion
In most cases, the header view is the navigation bar and the footer view is the tab bar. If the default behavior is not desired, subclasses should override this method to return the alternate view (that is already in the receiver's view hierarchy) that is used as the header view.

Availability
Available in iOS 2.0 and later.

See Also
- shouldAutorotateToInterfaceOrientation: (page 770)
- rotatingFooterView (page 768)

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 757) property is set to the value returned by the editButtonItem (page 765) 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 754) is YES and a noneditable state if it is NO. This method should invoke super's implementation before updating its view.

Availability
Available in iOS 2.0 and later.

See Also
`@property editing` (page 754)
`- editButtonItem` (page 765)

Declared In
`UIViewController.h`

setToolbarItems:animated:

Sets the toolbar items to be displayed along with the view controller.

```
- (void)setToolbarItems:(NSArray *)toolbarItems animated:(BOOL)animated
```

Parameters
toolbarItems
The toolbar items to display in a built-in toolbar.
animated
If `YES`, animate the change of items in the toolbar.

Discussion
View controllers that are managed by a navigation controller can use this method to specify toolbar items for the navigation controller's built-in toolbar. You can set the toolbar items for your view controller before your view controller is displayed or after it is already visible.

Availability
Available in iOS 3.0 and later.

Declared In
`UINavigationController.h`

shouldAutorotateToInterfaceOrientation:

Returns a Boolean value indicating whether the view controller supports the specified orientation.

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
```

Parameters
interfaceOrientation
The orientation of the application's user interface after the rotation. The possible values are described in `UIInterfaceOrientation` (page 128).

Return Value
`YES` if the view controller autorotates its view to the specified orientation; otherwise, `NO`.

Discussion
By default, this method returns `YES` for the `UIInterfaceOrientationPortrait` (page 128) orientation only. If your view controller supports additional orientations, override this method and return `YES` for all orientations it supports.

Your implementation of this method should simply return `YES` or `NO` based on the value in the *`interfaceOrientation`* parameter. Do not attempt to get the value of the `interfaceOrientation` (page 755) property or check the orientation value reported by the `UIDevice` class. Your view controller is either capable of supporting a given orientation or it is not.

Availability
Available in iOS 2.0 and later.

See Also
- `rotatingFooterView` (page 768)
- `rotatingHeaderView` (page 769)

Declared In
`UIViewController.h`

viewDidAppear:

Notifies the view controller that its view was added to a window.

```
- (void)viewDidAppear:(BOOL)animated
```

Parameters

`animated`
If `YES`, the view was added to the window using an animation.

Discussion
You can override this method to perform additional tasks associated with presenting the view. If you override this method, you must call `super` at some point in your implementation.

Availability
Available in iOS 2.0 and later.

See Also
- `viewWillAppear:` (page 773)

Declared In
`UIViewController.h`

viewDidDisappear:

Notifies the view controller that its view was dismissed, covered, or otherwise hidden from view.

```
- (void)viewDidDisappear:(BOOL)animated
```

Parameters

`animated`
If `YES`, the disappearance of the view was animated.

Discussion
You can override this method to perform additional tasks associated with dismissing or hiding the view. If you override this method, you must call `super` at some point in your implementation.

Availability
Available in iOS 2.0 and later.

See Also
- `viewWillDisappear:` (page 774)

Declared In
`UIViewController.h`

viewDidLoad

Called after the controller's view is loaded into memory.

```
- (void)viewDidLoad
```

Discussion
This method is called after the view controller has loaded its associated views into memory. This method is called regardless of whether the views were stored in a nib file or created programmatically in the `loadView` (page 767) method. This method is most commonly used to perform additional initialization steps on views that are loaded from nib files.

Availability
Available in iOS 2.0 and later.

See Also
- `@property view` (page 761)
- `loadView` (page 767)
- `viewDidUnload` (page 772)

Related Sample Code
AddMusic
CryptoExercise
GKRocket
GKTank
ToolbarSearch

Declared In
`UIViewController.h`

viewDidUnload

Called when the controller's view is released from memory.

```
- (void)viewDidUnload
```

Discussion
This method is called as a counterpart to the `viewDidLoad` method. It is called during low-memory conditions when the view controller needs to release its view and any objects associated with that view to free up memory. Because view controllers often store references to views and other view-related objects, you should use this method to relinquish ownership in those objects so that the memory for them can be reclaimed. You should do this only for objects that you can easily recreate later, either in your `viewDidLoad` method or from other parts of your application. You should not use this method to release user data or any other information that cannot be easily recreated.

Typically, a view controller stores references to objects using an outlet, which is a variable or property that includes the `IBOutlet` (page 1018) keyword and is configured using Interface Builder. A view controller may also store pointers to objects that it creates programmatically, such as in the `viewDidLoad` method. The preferred way to relinquish ownership of any object (including those in outlets) is to use the corresponding accessor method to set the value of the object to `nil`. However, if you do not have an accessor method for a given object, you may have to release the object explicitly. For more information about memory management practices, see *Memory Management Programming Guide*.

By the time this method is called, the `view` property is `nil`.

Special Considerations

If your view controller stores references to views and other custom objects, it is also responsible for relinquishing ownership of those objects safely in its `dealloc` method. If you implement this method but are building your application for iOS 2.x, your `dealloc` method should release each object but should also set the reference to that object to `nil` before calling `super`.

Availability
Available in iOS 3.0 and later.

Related Sample Code
AddMusic
KeyboardAccessory
MultipleDetailViews
SimpleGestureRecognizers
ToolbarSearch

Declared In
`UIViewController.h`

viewWillAppear:

Notifies the view controller that its view is about to be become visible.

```
- (void)viewWillAppear:(BOOL)animated
```

Parameters

animated
If `YES`, the view is being added to the window using an animation.

Discussion
This method is called before the receiver's view is about to be displayed onscreen and before any animations are configured for showing the view. You can override this method to perform custom tasks associated with presenting the view. For example, you might use this method to change the orientation or style of the status bar to coordinate with the orientation or style of the view being presented. If you override this method, you must call `super` at some point in your implementation.

For more information about the how views are added to windows, and the sequence of messages that occur, see the information on presenting a view controller's view in "Custom View Controllers" in *View Controller Programming Guide for iOS*.

Availability
Available in iOS 2.0 and later.

See Also
- `viewDidAppear:` (page 771)

Declared In
`UIViewController.h`

viewWillDisappear:

Notifies the view controller that its view is about to be dismissed, covered, or otherwise hidden from view.

```
- (void)viewWillDisappear:(BOOL)animated
```

Parameters

animated
If `YES`, the disappearance of the view is being animated.

Discussion
This method is called in response to a view being removed from its window or covered by another view. This method is called before the view is actually removed or covered and before any animations are configured.

Subclasses can override this method and use it to commit editing changes, resign the first responder status of the view, or perform other relevant tasks. For example, you might use this method to revert changes to the orientation or style of the status bar that were made in the `viewDidDisappear:` (page 771) method when the view was first presented. If you override this method, you must call `super` at some point in your implementation.

Availability
Available in iOS 2.0 and later.

See Also
- `viewDidDisappear:` (page 771)

Declared In
`UIViewController.h`

willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:

Sent to the view controller before performing the first half of a user interface rotation.

```
- (void)willAnimateFirstHalfOfRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
    duration:(NSTimeInterval)duration
```

Parameters

toInterfaceOrientation
The state of the application's user interface orientation before the rotation. The possible values are described in `UIInterfaceOrientation` (page 128).

duration
The duration of the first half of the pending rotation, measured in seconds.

Discussion
The default implementation of this method does nothing. To configure animations using the one-step technique, override the `willAnimateRotationToInterfaceOrientation:duration:` (page 775) method instead.

This method is called from within the animation block that is used to rotate the view and slide the header and footer views out. You can override this method and use it to configure additional animations that should occur during the first half of the view rotation. For example, you could use it to adjust the zoom level of your content, change the scroller position, or modify other animatable properties of your view.

At the time this method is called, the `interfaceOrientation` (page 755) property is still set to the old orientation.

Availability
Available in iOS 2.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 777)
- `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 776)
- `didRotateFromInterfaceOrientation:` (page 764)

Declared In
`UIViewController.h`

willAnimateRotationToInterfaceOrientation:duration:

Sent to the view controller before performing a one-step user interface rotation.

```
- (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
    duration:(NSTimeInterval)duration
```

Parameters

interfaceOrientation
The new orientation for the user interface. The possible values are described in `UIInterfaceOrientation` (page 128).

duration
The duration of the pending rotation, measured in seconds.

Discussion
The default implementation of this method does nothing. If you override this method, you should not override either the `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` or `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` method.

This method is called from within the animation block that is used to rotate the view. You can override this method and use it to configure additional animations that should occur during the view rotation. For example, you could use it to adjust the zoom level of your content, change the scroller position, or modify other animatable properties of your view.

Note: The animations used to slide the header and footer views in and out of position are performed in separate animation blocks.

By the time this method is called, the `interfaceOrientation` (page 755) property is already set to the new orientation. Thus, you can perform any additional layout required by your views in this method.

Availability
Available in iOS 3.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 777)
- `didRotateFromInterfaceOrientation:` (page 764)

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 `UIInterfaceOrientation` (page 128).

duration
The duration of the second half of the pending rotation, measured in seconds.

Discussion
The default implementation of this method does nothing. To configure animations using the one-step technique, override the `willAnimateRotationToInterfaceOrientation:duration:` (page 775) method instead.

This method is called from inside the animation block that is used to finish the view rotation and slide the header and footer views back into position. You can override this method and use it to configure additional animations that should occur during the second half of the view rotation. For example, you could use it to adjust the zoom level of your content, change the scroller position, or modify other animatable properties of your view.

At the time this method is invoked, the `interfaceOrientation` (page 755) property is set to the new orientation.

Availability
Available in iOS 2.0 and later.

See Also
- `willRotateToInterfaceOrientation:duration:` (page 777)
- `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 774)
- `didRotateFromInterfaceOrientation:` (page 764)

Declared In
UIViewController.h

willRotateToInterfaceOrientation:duration:

Sent to the view controller just before the user interface begins rotating.

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
    duration:(NSTimeInterval)duration
```

Parameters

toInterfaceOrientation

The new orientation for the user interface. The possible values are described in UIInterfaceOrientation (page 128).

duration

The duration of the pending rotation, measured in seconds.

Discussion
Subclasses may override this method to perform additional actions immediately prior to the rotation. For example, you might use this method to disable view interactions, stop media playback, or temporarily turn off expensive drawing or live updates. You might also use it to swap the current view for one that reflects the new interface orientation. When this method is called, the interfaceOrientation (page 755) property still contains the view's original orientation.

This method is called regardless of whether your code performs one-step or two-step rotations.

Availability
Available in iOS 2.0 and later.

See Also
- willAnimateRotationToInterfaceOrientation:duration: (page 775)
- didRotateFromInterfaceOrientation: (page 764)

Declared In
UIViewController.h

Constants

UIModalTransitionStyle

Transition styles available when presenting view controllers modally.

```
typedef enum {
        UIModalTransitionStyleCoverVertical = 0,
        UIModalTransitionStyleFlipHorizontal,
        UIModalTransitionStyleCrossDissolve,
        UIModalTransitionStylePartialCurl,
} UIModalTransitionStyle;
```

Constants

`UIModalTransitionStyleCoverVertical`

When the view controller is presented, its view slides up from the bottom of the screen. On dismissal, the view slides back down. This is the default transition style.

Available in iOS 3.0 and later.

Declared in `UIViewController.h`.

`UIModalTransitionStyleFlipHorizontal`

When the view controller is presented, the current view initiates a horizontal 3D flip from right-to-left, resulting in the revealing of the new view as if it were on the back of the previous view. On dismissal, the flip occurs from left-to-right, returning to the original view.

Available in iOS 3.0 and later.

Declared in `UIViewController.h`.

`UIModalTransitionStyleCrossDissolve`

When the view controller is presented, the current view fades out while the new view fades in at the same time. On dismissal, a similar type of cross-fade is used to return to the original view.

Available in iOS 3.0 and later.

Declared in `UIViewController.h`.

`UIModalTransitionStylePartialCurl`

When the view controller is presented, one corner of the current view curls up to reveal the modal view underneath. On dismissal, the curled up page unfurls itself back on top of the modal view. A modal view presented using this transition is itself prevented from presenting any additional modal views.

This transition style is supported only if the parent view controller is presenting a full-screen view and you use the `UIModalPresentationFullScreen` modal presentation style. Attempting to use a different form factor for the parent view or a different presentation style triggers an exception.

Available in iOS 3.2 and later.

Declared in `UIViewController.h`.

UIModalPresentationStyle

Presentation styles available when presenting view controllers modally.

```
typedef enum {
    UIModalPresentationFullScreen = 0,
    UIModalPresentationPageSheet,
    UIModalPresentationFormSheet,
    UIModalPresentationCurrentContext,
} UIModalPresentationStyle;
```

Constants

`UIModalPresentationFullScreen`

The presented view covers the screen, taking into account the value of the `wantsFullScreenLayout` (page 762) property.

Available in iOS 3.2 and later.

Declared in `UIViewController.h`.

`UIModalPresentationPageSheet`

The height of the presented view is set to the height of the screen and the view's width is set to the width of the screen in a portrait orientation. Any uncovered areas are dimmed to prevent the user from interacting with them. (In portrait orientations, this option is essentially the same as `UIModalPresentationFullScreen`.)

Available in iOS 3.2 and later.

Declared in `UIViewController.h`.

`UIModalPresentationFormSheet`

The width and height of the presented view are smaller than those of the screen and the view is centered on the screen. If the device is in a landscape orientation and the keyboard is visible, the position of the view is adjusted upward so that the view remains visible. All uncovered areas are dimmed to prevent the user from interacting with them.

Available in iOS 3.2 and later.

Declared in `UIViewController.h`.

`UIModalPresentationCurrentContext`

The view is presented using the same style as its parent view controller.

When presenting a view controller in a popover, this presentation style is supported only if the transition style is `UIModalTransitionStyleCoverVertical`. Attempting to use a different transition style triggers an exception. However, you may use other transition styles (except the partial curl transition) if the parent view controller is not in a popover.

Available in iOS 3.2 and later.

Declared in `UIViewController.h`.

UIWebView Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding UIScrollViewDelegate NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 789) method to begin loading web content, the `stopLoading` (page 789) method to stop loading, and the `loading` (page 786) 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 787) and `goForward` (page 787) methods as actions for buttons. Use the `canGoBack` (page 784) and `canGoForward` (page 784) 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 785) property to `NO` to turn off this default behavior.

You can also use the `scalesPageToFit` (page 787) 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 785) property to an object conforming to the `UIWebViewDelegate` protocol if you want to track the loading of web content.

Read *Safari Web Content Guide* for how to create web content that is compatible with and optimized for displaying in Safari on iPhone and your web views.

Important: You should not embed UIWebView objects in UIScrollView object. If you do so, unexpected behavior can result because touch events for the two objects can be mixed up and wrongly handled.

Supported File Formats

In addition to HTML content, `UIWebView` objects can be used to display other content types. For more information, see *Using UIWebView to display select document types*.

Subclassing Notes

The UIWebView class should not be subclassed.

Tasks

Setting the Delegate

`delegate` (page 785) *property*
The receiver's delegate.

Loading Content

- `loadData:MIMEType:textEncodingName:baseURL:` (page 788)
Sets the main page contents, MIME type, content encoding, and base URL.
- `loadHTMLString:baseURL:` (page 788)
Sets the main page content and base URL.

`request` (page 786) *property*
The URL request identifying the location of the content to load. (read-only)

`loading` (page 786) *property*
A Boolean value indicating whether the receiver is done loading content. (read-only)

- `stopLoading` (page 789)
Stops the loading of any web content managed by the receiver.
- `reload` (page 789)
Reloads the current page.
- `loadRequest:` (page 789) Deprecated in iOS 3.1
Connects to a given URL by initiating an asynchronous client request.

Moving Back and Forward

`canGoBack` (page 784) *property*
A Boolean value indicating whether the receiver can move backward. (read-only)

`canGoForward` (page 784) *property*
A Boolean value indicating whether the receiver can move forward. (read-only)

- `goBack` (page 787)
Loads the previous location in the back-forward list.

- `goForward` (page 787)
Loads the next location in the back-forward list.

Setting Web Content Properties

`detectsPhoneNumbers` (page 785) *property*
A Boolean value indicating whether telephone number detection is on. (Deprecated. Use `dataDetectorTypes` (page 784) instead.)

`scalesPageToFit` (page 787) *property*
A Boolean value determining whether the webpage scales to fit the view and the user can change the scale.

Running JavaScript

- `stringByEvaluatingJavaScriptFromString:` (page 790)
Returns the result of running a script.

Detecting Types of Data

`dataDetectorTypes` (page 784) *property*
The types of data converted to clickable URLs in the web view's content.

Managing Media Playback

`allowsInlineMediaPlayback` (page 783) *property*
A Boolean value that determines whether HTML5 videos play inline or use the native full-screen controller.

`mediaPlaybackRequiresUserAction` (page 786) *property*
A Boolean value that determines whether HTML5 videos can play automatically or require the user to start playing them.

Properties

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

allowsInlineMediaPlayback

A Boolean value that determines whether HTML5 videos play inline or use the native full-screen controller.

```
@property (nonatomic) BOOL allowsInlineMediaPlayback
```

Discussion
The default value on iPhone is `NO`.

Availability
Available in iOS 4.0 and later.

Declared In
`UIWebView.h`

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 iOS 2.0 and later.

See Also
`@property canGoForward` (page 784)

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 iOS 2.0 and later.

See Also
`@property canGoBack` (page 784)

Declared In
`UIWebView.h`

dataDetectorTypes

The types of data converted to clickable URLs in the web view's content.

```
@property(nonatomic) UIDataDetectorTypes dataDetectorTypes
```

Discussion
You can use this property to specify the types of data (phone numbers, http links, email address, and so on) that should be automatically converted to clickable URLs in the web view. When clicked, the web view opens the application responsible for handling the URL type and passes it the URL.

Availability
Available in iOS 3.0 and later.

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. See *UIWebViewDelegate Protocol Reference* for the optional methods this delegate may implement.

Important: Before releasing an instance of `UIWebView` for which you have set a delegate, you must first set its delegate property to `nil`. This can be done, for example, in your dealloc method.

Availability
Available in iOS 2.0 and later.

Declared In
`UIWebView.h`

detectsPhoneNumbers

A Boolean value indicating whether telephone number detection is on. (Deprecated in iOS 3.0. Use `dataDetectorTypes` (page 784) instead.)

```
@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.

Special Considerations

The functionality provided by this property has been superseded by the `dataDetectorTypes` (page 784) property.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 3.0.

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 still loading content; otherwise, NO.

Availability
Available in iOS 2.0 and later.

See Also
- @property request (page 786)
- stopLoading (page 789)
- loadRequest: (page 789)
- reload (page 789)

Declared In
UIWebView.h

mediaPlaybackRequiresUserAction

A Boolean value that determines whether HTML5 videos can play automatically or require the user to start playing them.

```
@property (nonatomic) BOOL mediaPlaybackRequiresUserAction
```

Discussion
The default value on both iPad and iPhone is YES.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
- loadRequest: (page 789)

- `stopLoading` (page 789)
  `@property loading` (page 786)
- `reload` (page 789)

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 iOS 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 iOS 2.0 and later.

See Also
- `goBack` (page 787)

Declared In
`UIWebView.h`

goForward

Loads the next location in the back-forward list.

```
- (void)goForward
```

Availability
Available in iOS 2.0 and later.

See Also
- goBack (page 787)

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 iOS 2.0 and later.

See Also
- loadHTMLString:baseURL: (page 788)

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 iOS 2.0 and later.

See Also
- loadData:MIMEType:textEncodingName:baseURL: (page 788)

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 789) method. To see whether the receiver is done loading the content, use the `loading` (page 786) property.

Availability

Available in iOS 2.0 and later.

See Also

`@property request` (page 786)
`- stopLoading` (page 789)
`@property loading` (page 786)
`- reload` (page 789)

Declared In

`UIWebView.h`

reload

Reloads the current page.

```
- (void)reload
```

Availability

Available in iOS 2.0 and later.

See Also

`@property request` (page 786)
`@property loading` (page 786)
`- loadRequest:` (page 789)
`- stopLoading` (page 789)

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 iOS 2.0 and later.

See Also
`@property request` (page 786)
`@property loading` (page 786)
`- loadRequest:` (page 789)
`- reload` (page 789)

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 10 seconds for each top-level entry point. If your script executes for more than 10 seconds, the web view 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. The web view raises an exception if you exceed this limit on the total memory allocation for JavaScript.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared in `UIWebView.h`.

`UIWebViewNavigationTypeFormSubmitted`
User submitted a form.

Available in iOS 2.0 and later.

Declared in `UIWebView.h`.

`UIWebViewNavigationTypeBackForward`
User tapped the back or forward button.

Available in iOS 2.0 and later.

Declared in `UIWebView.h`.

`UIWebViewNavigationTypeReload`
User tapped the reload button.

Available in iOS 2.0 and later.

Declared in `UIWebView.h`.

`UIWebViewNavigationTypeFormResubmitted`
User resubmitted a form.

Available in iOS 2.0 and later.

Declared in `UIWebView.h`.

`UIWebViewNavigationTypeOther`
Some other action occurred.

Available in iOS 2.0 and later.

Declared in `UIWebView.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIWebView.h`

UIWindow Class Reference

Inherits from	UIView : UIResponder : NSObject
Conforms to	NSCoding (UIView) NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIWindow.h
Related sample code	GKRocket GKTank MoviePlayer ScrollViewSuite SpeakHere

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 iOS application.

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

Tasks

Configuring Windows

`windowLevel` (page 796) *property*
The receiver's window level.

`rootViewController` (page 795) *property*
The root view controller for the window.

`screen` (page 795) *property* Deprecated in iOS 3.0
The screen on which the window is currently displayed.

Making Windows Key

`keyWindow` (page 794) *property*
A Boolean value that indicates whether the receiver is the key window for the application. (read-only)

- `makeKeyAndVisible` (page 798)
Makes the receiver the key window and makes that window visible.

- `becomeKeyWindow` (page 796)
Invoked automatically to inform the receiver that it has become the key window; never invoke this method directly.

- `makeKeyWindow` (page 799)
Makes the receiver the main window.

- `resignKeyWindow` (page 799)
Invoked automatically when the window resigns key window status; never invoke this method directly.

Converting Coordinates

- `convertPoint:toWindow:` (page 797)
Converts a point from the receiver's coordinate system to that of another window.

- `convertPoint:fromWindow:` (page 796)
Converts a point from the coordinate system of a given window to that of the receiver.

- `convertRect:toWindow:` (page 798)
Converts a rectangle from the receiver's coordinate system to that of another window.

- `convertRect:fromWindow:` (page 797)
Converts a rectangle from the coordinate system of another window to that of the receiver.

Sending Events

- `sendEvent:` (page 800)
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 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 iOS 2.0 and later.

See Also
- `makeKeyAndVisible` (page 798)
- `becomeKeyWindow` (page 796)
- `makeKeyWindow` (page 799)
- `resignKeyWindow` (page 799)

Declared In
`UIWindow.h`

rootViewController

The root view controller for the window.

```
@property(nonatomic,retain) UIViewController *rootViewController
```

Discussion
The root view controller provides the content view of the window. Assigning a view controller to this property (either programmatically or using Interface Builder) installs the view controller's view as the content view of the window. If the window has an existing view hierarchy, the old views are removed before the new ones are installed.

The default value of this property is `nil`.

Availability
Available in iOS 4.0 and later.

Declared In
`UIWindow.h`

screen

The screen on which the window is currently displayed.

```
@property (nonatomic,retain) UIScreen *screen
```

Discussion
By default, all windows are created on the main screen. If additional screens are attached to the device, assigning a different screen object to this property causes the window to be displayed on the new screen.

Moving windows from screen to screen is a relatively expensive operation and should not be done in performance-sensitive code. Instead, it is recommended that you change the screen before displaying the window the first time. Changing the screen of a window that has not yet been ordered onto the screen has no significant additional cost.

Availability
Available in iOS 3.2 and later.

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 800) for a list of possible values. The default value is `0.0`.

Availability
Available in iOS 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 796) message to that object if it responds, and posts `UIWindowDidBecomeKeyNotification` (page 803) to the default notification center.

Availability
Available in iOS 2.0 and later.

See Also
`@property keyWindow` (page 794)
`- makeKeyAndVisible` (page 798)
`- makeKeyWindow` (page 799)
`- resignKeyWindow` (page 799)

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 `nil`, this method converts the point from the logical coordinate system of the screen, which is measured in points.

Return Value

The point converted to the coordinate system of the receiver.

Availability

Available in iOS 2.0 and later.

See Also

`- convertPoint:toWindow:` (page 797)

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 logical coordinate system of the receiver.

window

The window into whose coordinate system *point* is to be converted. If `nil`, this method converts the point to the logical coordinate system of the screen, which is measured in points.

Return Value

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

Availability

Available in iOS 2.0 and later.

See Also

`- convertPoint:fromWindow:` (page 796)

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 the rectangle from the logical coordinate system of the screen, which is measured in points.

Return Value
The converted rectangle.

Availability
Available in iOS 2.0 and later.

See Also
`- convertRect:toWindow:` (page 798)

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 the rectangle to the logical coordinate system of the screen, which is measured in points.

Return Value
The converted rectangle.

Availability
Available in iOS 2.0 and later.

See Also
`- convertRect:fromWindow:` (page 797)

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 703)`UIView` property.

Availability
Available in iOS 2.0 and later.

See Also
`@property keyWindow` (page 794)
`- becomeKeyWindow` (page 796)
`- makeKeyWindow` (page 799)
`- resignKeyWindow` (page 799)

Declared In
`UIWindow.h`

makeKeyWindow

Makes the receiver the main window.

```
- (void)makeKeyWindow
```

Availability
Available in iOS 2.0 and later.

See Also
`@property keyWindow` (page 794)
`- makeKeyAndVisible` (page 798)
`- becomeKeyWindow` (page 796)
`- resignKeyWindow` (page 799)

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 799) to the receiver's first responder and posts `UIWindowDidResignKeyNotification` (page 803) to the default notification center.

Availability
Available in iOS 2.0 and later.

See Also
`@property keyWindow` (page 794)
`- makeKeyAndVisible` (page 798)
`- becomeKeyWindow` (page 796)
`- makeKeyWindow` (page 799)

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 iOS 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 iOS 2.0 and later.

Declared in `UIWindow.h`.

`UIWindowLevelAlert`

The level for an alert view.

Available in iOS 2.0 and later.

Declared in `UIWindow.h`.

`UIWindowLevelStatusBar`

The level for a status window.

Available in iOS 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 UIKeyboardFrameBeginUserInfoKey;
NSString *const UIKeyboardFrameEndUserInfoKey;
NSString *const UIKeyboardAnimationDurationUserInfoKey;
NSString *const UIKeyboardAnimationCurveUserInfoKey;

// Deprecated in iOS 3.2 and later.
NSString *const UIKeyboardCenterBeginUserInfoKey;
NSString *const UIKeyboardCenterEndUserInfoKey;
NSString *const UIKeyboardBoundsUserInfoKey;
```

Constants

`UIKeyboardFrameBeginUserInfoKey`

The key for an `NSValue` object containing a `CGRect` that identifies the start frame of the keyboard in screen coordinates. These coordinates do not take into account any rotation factors applied to the window's contents as a result of interface orientation changes. Thus, you may need to convert the rectangle to window coordinates (using the `convertRect:fromWindow:` (page 797) method) or to view coordinates (using the `convertRect:fromView:` (page 724) method) before using it.

Available in iOS 3.2 and later.

Declared in `UIWindow.h`.

`UIKeyboardFrameEndUserInfoKey`

The key for an `NSValue` object containing a `CGRect` that identifies the end frame of the keyboard in screen coordinates. These coordinates do not take into account any rotation factors applied to the window's contents as a result of interface orientation changes. Thus, you may need to convert the rectangle to window coordinates (using the `convertRect:fromWindow:` (page 797) method) or to view coordinates (using the `convertRect:fromView:` (page 724) method) before using it.

Available in iOS 3.2 and later.

Declared in `UIWindow.h`.

`UIKeyboardAnimationCurveUserInfoKey`

The key for an `NSValue` object containing a `UIViewAnimationCurve` (page 739) constant that defines how the keyboard will be animated onto or off the screen.

Available in iOS 3.0 and later.

Declared in `UIWindow.h`.

`UIKeyboardAnimationDurationUserInfoKey`

The key for an `NSValue` object containing a `double` that identifies the duration of the animation in seconds.

Available in iOS 3.0 and later.

Declared in `UIWindow.h`.

`UIKeyboardCenterBeginUserInfoKey`

The key for an `NSValue` object containing a `CGPoint` that is the center of the keyboard in window coordinates before animation. These coordinates actually take into account any rotation factors applied to the window's contents as a result of interface orientation changes. Thus, the center point of the keyboard is different in portrait versus landscape orientations.

Use the `UIKeyboardFrameBeginUserInfoKey` key instead.

Available in iOS 2.0 and later.

Deprecated in iOS 3.2.

Declared in `UIWindow.h`.

`UIKeyboardCenterEndUserInfoKey`

The key for an `NSValue` object containing a `CGPoint` that is the center of the keyboard in window coordinates after animation. These coordinates take into account any rotation factors applied to the window's contents as a result of interface orientation changes. Thus, the center point of the keyboard is different in portrait versus landscape orientations.

Use the `UIKeyboardFrameEndUserInfoKey` key instead.

Available in iOS 2.0 and later.

Deprecated in iOS 3.2.

Declared in `UIWindow.h`.

`UIKeyboardBoundsUserInfoKey`

The key for an `NSValue` object containing a `CGRect` that identifies the bounds rectangle of the keyboard in window coordinates. This value is sufficient for obtaining the size of the keyboard. If you want to get the origin of the keyboard on the screen (before or after animation) use the values obtained from the user info dictionary through the `UIKeyboardCenterBeginUserInfoKey` (page 801) or `UIKeyboardCenterEndUserInfoKey` (page 802) constants.

Use the `UIKeyboardFrameBeginUserInfoKey` or `UIKeyboardFrameEndUserInfoKey` key instead.

Available in iOS 2.0 and later.

Deprecated in iOS 3.2.

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 iOS 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 iOS 2.0 and later.

Declared In
`UIWindow.h`

UIWindowDidBecomeKeyNotification

Posted whenever a `UIWindow` 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 iOS 2.0 and later.

Declared In
`UIWindow.h`

UIWindowDidResignKeyNotification

Posted whenever a `UIWindow` 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 iOS 2.0 and later.

Declared In
`UIWindow.h`

UIKeyboardWillShowNotification

Posted before a `UIWindow` 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 800) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iOS 2.0 and later.

Declared In
`UIWindow.h`

UIKeyboardDidShowNotification

Posted after a `UIWindow` 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 800) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iOS 2.0 and later.

Declared In
`UIWindow.h`

UIKeyboardWillHideNotification

Posted before a `UIWindow` 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 800) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iOS 2.0 and later.

Declared In
`UIWindow.h`

UIKeyboardDidHideNotification

Posted after a `UIWindow` 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 800) to get the location and size of the keyboard from the *`userInfo`* dictionary.

Availability
Available in iOS 2.0 and later.

Declared In
`UIWindow.h`

Protocols

UIAccelerometerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 807)
Delivers the latest acceleration data to the delegate.

Instance Methods

accelerometer:didAccelerate:

Delivers the latest acceleration data to the delegate.

```
- (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 iOS 2.0 and later.

Declared In
`UIAccelerometer.h`

UIAccessibility Protocol Reference

(informal protocol)

Adopted by	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Declared in	UIAccessibility.h
Companion guide	Accessibility Programming Guide for iOS

Overview

The `UIAccessibility` informal protocol provides accessibility information about an application's user interface elements. Assistive applications, such as VoiceOver, convey this information to users with disabilities to help them use the application.

Standard UIKit controls and views implement the `UIAccessibility` methods and are therefore accessible to assistive applications by default. This means that if your application uses only standard controls and views, such as `UIButton`, `UISegmentedControl`, and `UITableView`, you need only supply application-specific details when the default values are incomplete. You can do this by setting these values in Interface Builder or by setting the properties in this informal protocol.

The `UIAccessibility` informal protocol is also implemented by the `UIAccessibilityElement` class, which represents custom user interface objects. If you create a completely custom `UIView` subclass, you might need to create an instance of `UIAccessibilityElement` to represent it. In this case, you would support all the `UIAccessibility` properties to correctly set and return the accessibility element's properties.

Tasks

Determining Accessibility

`isAccessibilityElement` (page 812) *property*
A Boolean value indicating whether the receiver is an accessibility element that an assistive application can access.

Configuring an Accessibility Element

`accessibilityLabel` (page 811) *property*
A succinct label that identifies the accessibility element, in a localized string.

`accessibilityHint` (page 810) *property*
A brief description of the result of performing an action on the accessibility element, in a localized string.

`accessibilityValue` (page 812) *property*
The value of the accessibility element, in a localized string.

`accessibilityTraits` (page 812) *property*
The combination of accessibility traits that best characterize the accessibility element.

`accessibilityFrame` (page 810) *property*
The frame of the accessibility element, in screen coordinates.

`accessibilityLanguage` (page 811) *property*
The language in which to speak the accessibility element's label, value, and hint.

Properties

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

accessibilityFrame

The frame of the accessibility element, in screen coordinates.

```
@property(nonatomic) CGRect accessibilityFrame
```

Discussion
The default value for this property is `CGRectZero` unless the receiver is a `UIView` or subclass of `UIView`, in which case the value is the frame of the view.

You must set this property for an accessibility element that represents an object that is not a subclass of `UIView`, because the screen coordinates of such an object are not already known. (You do not have to set this property for an accessibility element that represents a subclass of `UIView` because such an object's screen coordinates are already known.)

Declared In
`UIAccessibility.h`

accessibilityHint

A brief description of the result of performing an action on the accessibility element, in a localized string.

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

Discussion
The default value for this property is `nil` unless the receiver is a UIKit control, in which case the value is a system-provided hint based on the type of control.

An accessibility hint helps users understand what will happen when they perform an action on the accessibility element, when that result is not obvious from the accessibility label. For example, if you provide an Add button in your application, the button's accessibility label helps users understand that tapping the button adds values in the application. If, on the other hand, your application allows users to play a song by tapping

its title in a list of song titles, the accessibility label for the list row does not tell users this. To help an assistive application provide this information to users with disabilities, an appropriate hint for the list row would be "Plays the song."

Follow these guidelines to create a hint for an accessibility element:

- The hint should be a very brief phrase that begins with a plural verb that names the results of the action, such as "Plays the song" or "Purchases the item."

  Avoid beginning the phrase with a singular verb, because this can make the hint sound like a command. For example, do not create a hint such as "Play the song" or "Purchase the item."
- Don't repeat the action type in the hint. For example, do not create hints such as "Tap to play the song" or "Tapping plays the song."
- Don't repeat the control or view type in the hint. For example, do not create hints such as "Plays the song in the row" or "Button that adds a contact name."

Declared In
`UIAccessibility.h`

accessibilityLabel

A succinct label that identifies the accessibility element, in a localized string.

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

Discussion
The default value for this property is `nil` unless the receiver is a UIKit control, in which case the value is a label derived from the control's title.

> **Note:** If you supply `UIImage` objects to display in a `UISegmentedControl`, you can set this property on each image to ensure that the segments are properly accessible.

If you implement a custom control or view, or if you display a custom icon on a UIKit control, you should set this property to make sure your accessibility elements have appropriate labels. If an accessibility element does not display a descriptive label, set this property to supply a short, localized label that succinctly identifies the element. For example, a "Play music" button might display an icon that shows sighted users what it does. To be accessible, however, the button should have the accessibility label "Play" or "Play music" so that an assistive application can provide this information to users with disabilities. Note, however, that the label should never include the control type (such as "button") because this information is contained in the traits associated with the accessibility element.

Declared In
`UIAccessibility.h`

accessibilityLanguage

The language in which to speak the accessibility element's label, value, and hint.

```
@property(nonatomic, retain) NSString *accessibilityLanguage
```

Discussion
The default value for this property is `nil`. If no language is set, the user's current language setting is used.

If you need to set this property, be sure to use a language ID tag that follows the format defined in the BCP 47 specification (a draft of this specification is available at http://www.rfc-editor.org/).

Declared In
`UIAccessibility.h`

accessibilityTraits

The combination of accessibility traits that best characterize the accessibility element.

```
@property(nonatomic) UIAccessibilityTraits accessibilityTraits
```

Discussion
The default value for this property is `UIAccessibilityTraitNone` (page 814) unless the receiver is a UIKit control, in which case the value is the standard set of traits associated with the control.

If you implement a custom control or view, you need to select all the accessibility traits that best characterize the object and OR them together with its superclass's traits (in other words, with `super.accessibilityTraits`). See "Accessibility Traits" (page 813) for a complete list of traits.

Declared In
`UIAccessibility.h`

accessibilityValue

The value of the accessibility element, in a localized string.

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

Discussion
The default value for this property is `nil` unless the receiver is a UIKit control, in which case value of the property represents the value of the control, when it differs from the label.

When an accessibility element has a static label, but a dynamic value, you should set this property to return the value. For example, although an accessibility element that represents a text field might have the label "Message," its value is the text currently inside the text field.

Declared In
`UIAccessibility.h`

isAccessibilityElement

A Boolean value indicating whether the receiver is an accessibility element that an assistive application can access.

```
@property(nonatomic) BOOL isAccessibilityElement
```

Discussion
The default value for this property is `NO` unless the receiver is a standard UIKit control, in which case the value is `YES`.

Assistive applications can only get information about objects that are represented by accessibility elements. Therefore, if you implement a custom control or view that should be accessible to users with disabilities, you should set this property to `YES`. The only exception to this is a view that merely serves as a container for other items that should be accessible. Such a view should implement the `UIAccessibilityContainer` protocol and set this property to `NO`.

Declared In
`UIAccessibility.h`

Constants

UIAccessibilityTraits

A mask that contains the OR combination of the accessibility traits that best characterize an accessibility element.

```
typedef uint64_t UIAccessibilityTraits;
```

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityConstants.h`

Accessibility Traits

Accessibility traits that tell an assistive application how an accessibility element behaves or should be treated.

```
UIAccessibilityTraits UIAccessibilityTraitNone;
UIAccessibilityTraits UIAccessibilityTraitButton;
UIAccessibilityTraits UIAccessibilityTraitLink;
UIAccessibilityTraits UIAccessibilityTraitSearchField;
UIAccessibilityTraits UIAccessibilityTraitImage;
UIAccessibilityTraits UIAccessibilityTraitSelected;
UIAccessibilityTraits UIAccessibilityTraitPlaysSound;
UIAccessibilityTraits UIAccessibilityTraitKeyboardKey;
UIAccessibilityTraits UIAccessibilityTraitStaticText;
UIAccessibilityTraits UIAccessibilityTraitSummaryElement;
UIAccessibilityTraits UIAccessibilityTraitNotEnabled;
UIAccessibilityTraits UIAccessibilityTraitUpdatesFrequently;
UIAccessibilityTraits UIAccessibilityTraitStartsMediaSession;
UIAccessibilityTraits UIAccessibilityTraitAdjustable;
```

Constants

`UIAccessibilityTraitNone`

The accessibility element has no traits.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitButton`

The accessibility element should be treated as a button.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitLink`

The accessibility element should be treated as a link.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitSearchField`

The accessibility element should be treated as a search field.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitImage`

The accessibility element should be treated as an image.

This trait can be combined with the button or link traits.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitSelected`

The accessibility element is currently selected.

You can use this trait to characterize an accessibility element that represents, for example, a selected table row or a selected segment in a segmented control.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitPlaysSound`

The accessibility element plays its own sound when activated.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitKeyboardKey`
The accessibility element behaves as a keyboard key.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitStaticText`
The accessibility element should be treated as static text that cannot change.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitSummaryElement`
The accessibility element provides summary information when the application starts.

You can use this trait to characterize an accessibility element that provides a summary of current conditions, settings, or state, such as the current temperature in the Weather application.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitNotEnabled`
The accessibility element is not enabled and does not respond to user interaction.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitUpdatesFrequently`
The accessibility element frequently updates its label or value.

You can use this trait to characterize an accessibility element that updates its label or value too often to send update notifications. Including this trait allows an assistive application to avoid handling continual notifications and, instead, poll for changes when it needs updated information. For example, you might use this trait to characterize the readout of a stopwatch.

Available in iOS 3.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitStartsMediaSession`
The accessibility element starts a media session when it is activated.

You can use this trait to silence the audio output of an assistive technology, such as VoiceOver, during a media session that should not be interrupted. For example, you might use this trait to silence VoiceOver speech while the user is recording audio.

Available in iOS 4.0 and later.

Declared in `UIAccessibilityConstants.h`.

`UIAccessibilityTraitAdjustable`
The accessibility element allows continuous adjustment through a range of values.

You can use this trait to characterize an accessibility element that users can adjust in a continuous manner, such as a slider or a picker view. If you specify this trait on an accessibility element, you must also implement the `accessibilityIncrement` (page 820) and `accessibilityDecrement` (page 819) methods in the `UIAccessibilityAction` protocol.

Available in iOS 4.0 and later.

Declared in `UIAccessibilityConstants.h`.

UIAccessibilityNotifications

A notification that an accessible application can send.

```
typedef uint32_t UIAccessibilityNotifications;
```

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityConstants.h`

Notifications

UIAccessibilityAnnouncementNotification

Posted by an application when an announcement needs to be conveyed to the assistive technology. This notification includes a parameter, which is an `NSString` object that contains the announcement. An assistive technology outputs the announcement string contained in the parameter.

Use this notification to provide accessibility information about events that do not update the application user interface (UI), or that update the UI only briefly.

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibilityConstants.h`

UIAccessibilityLayoutChangedNotification

Posted by an application when the layout of a screen changes, such as when an element appears or disappears. This notification does not include a parameter.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityConstants.h`

UIAccessibilityScreenChangedNotification

Posted by an application when a new view appears that comprises a major portion of the screen. This notification does not include a parameter.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibilityConstants.h`

UIAccessibilityVoiceOverStatusChanged

Posted by UIKit when VoiceOver starts or stops. This notification does not include a parameter.

You can use this notification to customize your application's user interface (UI) for VoiceOver users. For example, if you display a UI element that briefly overlays other parts of your UI, you can make the display persistent for VoiceOver users, but allow it to disappear as designed for users who are not using VoiceOver. Note that you can also use `UIAccessibilityIsVoiceOverRunning` to determine whether VoiceOver is currently running.

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibility.h`

UIAccessibilityAction Protocol Reference

(informal protocol)

Adopted by	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Declared in	UIAccessibility.h
Companion guide	Accessibility Programming Guide for iOS

Overview

The `UIAccessibilityAction` informal protocol provides a way for accessibility elements to support specific actions. For example, for UI elements such as sliders and picker views, users can select a value by manipulating the element through a range of values. To make such elements accessible, you first need to characterize it by including the `UIAccessibilityTraitAdjustable` trait. Then, you must implement the methods of the `UIAccessibilityAction` protocol. When you do this, assistive technology users can adjust the element using technology-specific gestures.

Tasks

Performing the Action

- `accessibilityIncrement` (page 820)
  Adjusts the accessibility element so that its content is increased.
- `accessibilityDecrement` (page 819)
  Adjusts the accessibility element so that its content is decreased.

Instance Methods

accessibilityDecrement

Adjusts the accessibility element so that its content is decreased.

```
- (void)accessibilityDecrement
```

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibility.h`

accessibilityIncrement

Adjusts the accessibility element so that its content is increased.

```
- (void)accessibilityIncrement
```

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibility.h`

UIAccessibilityContainer Protocol Reference

(informal protocol)

Adopted by	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Declared in	UIAccessibility.h
Companion guide	Accessibility Programming Guide for iOS

Overview

The `UIAccessibilityContainer` informal protocol provides a way for `UIView` subclasses to make selected components accessible as separate elements. For example, a view might contain icons or drawn text that, to end users, appear and function as separate items. But because these components are not implemented as instances of `UIView`, they are not automatically accessible to users with disabilities. Therefore, such a container view should implement the `UIAccessibilityContainer` methods to supply accessibility information about these components to assistive applications such as VoiceOver.

A view that implements the `UIAccessibilityContainer` informal protocol uses the `UIAccessibilityElement` method `initWithAccessibilityContainer:` (page 77) to create an accessibility element to represent each non-view component that needs to be accessible to users with disabilities. Note, however, that the container view itself is not an accessibility element because users interact with the contents, not with the container. This means that a container view that implements the `UIAccessibilityContainer` methods must set to `NO` the `isAccessibilityElement` property of the `UIAccessibility` informal protocol.

The order of accessibility elements within the container view should be the same as the order in which the represented elements are presented to the user, from top-left to bottom-right.

Tasks

Providing Information About Accessibility Elements

- `accessibilityElementCount` (page 822)
  Returns the number of accessibility elements in the container.
- `accessibilityElementAtIndex:` (page 822)
  Returns the accessibility element at the specified index.
- `indexOfAccessibilityElement:` (page 822)
  Returns the index of the specified accessibility element.

Instance Methods

accessibilityElementAtIndex:

Returns the accessibility element at the specified index.

```
- (id)accessibilityElementAtIndex:(NSInteger)index
```

Parameters

index

The index of the accessibility element.

Return Value

The accessibility element at the specified index, or `nil` if none exists.

Availability

Available in iOS 3.0 and later.

See Also

`- indexOfAccessibilityElement:` (page 822)

Declared In

`UIAccessibility.h`

accessibilityElementCount

Returns the number of accessibility elements in the container.

```
- (NSInteger)accessibilityElementCount
```

Return Value

The number of accessibility elements in the container. By default, this method returns `0`.

Availability

Available in iOS 3.0 and later.

Declared In

`UIAccessibility.h`

indexOfAccessibilityElement:

Returns the index of the specified accessibility element.

```
- (NSInteger)indexOfAccessibilityElement:(id)element
```

Parameters

element

The accessibility element.

Return Value

The index of the specified accessibility element, or `NSNotFound` if the element does not exist.

Availability
Available in iOS 3.0 and later.

See Also
`- accessibilityElementAtIndex:` (page 822)

Declared In
`UIAccessibility.h`

UIAccessibilityFocus Protocol Reference

(informal protocol)

Adopted by	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Declared in	UIAccessibility.h
Companion guide	Accessibility Programming Guide for iOS

Overview

The `UIAccessibilityFocus` informal protocol provides a way to find out whether an assistive technology, such as VoiceOver, is focused on an accessible element.

VoiceOver and other assistive technologies place a virtual focus on elements, which allows users to inspect an element without activating it. If you know the current location of the virtual focus, you can optimize the user experience for assistive technology users. For example, if your application expects people to tap once to select an object and then double-tap to activate it, VoiceOver users must make an extra tap to focus VoiceOver on the object before tapping to select it. To improve the VoiceOver user's experience, you can move selection to an element at the same time VoiceOver focuses on the element. In this way, the user can activate the element without having to tap again to select the element.

Tasks

Getting Focus Information

- `accessibilityElementDidBecomeFocused` (page 826)
  Sent after an assistive technology has set its virtual focus on the accessibility element.
- `accessibilityElementDidLoseFocus` (page 826)
  Sent after an assistive technology has removed its virtual focus from an accessibility element.
- `accessibilityElementIsFocused` (page 826)
  Returns a Boolean value indicating whether an assistive technology is focused on the accessibility element.

Instance Methods

accessibilityElementDidBecomeFocused

Sent after an assistive technology has set its virtual focus on the accessibility element.

```
- (void)accessibilityElementDidBecomeFocused
```

Discussion
Override `accessibilityElementDidBecomeFocused` if you need to know when an assistive technology has set its virtual focus on an accessibility element.

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibility.h`

accessibilityElementDidLoseFocus

Sent after an assistive technology has removed its virtual focus from an accessibility element.

```
- (void)accessibilityElementDidLoseFocus
```

Discussion
Override `accessibilityElementDidLoseFocus` if you need to know when an assistive technology has removed its virtual focus from an accessibility element. Note that `accessibilityElementDidLoseFocus` is sent before `accessibilityElementDidBecomeFocused` (page 826).

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibility.h`

accessibilityElementIsFocused

Returns a Boolean value indicating whether an assistive technology is focused on the accessibility element.

```
- (BOOL)accessibilityElementIsFocused
```

Return Value
`YES` if an assistive technology is virtually focused on the element; otherwise, `NO`.

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibility.h`

UIActionSheetDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Framework/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIAlert.h
Related sample code	ToolbarSearch WiTap

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 82) 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 828) 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 828) delegate method is invoked.

Optionally, you can implement the `actionSheetCancel:` (page 829) 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 828).

Tasks

Responding to Actions

`- actionSheet:clickedButtonAtIndex:` (page 828)
Sent to the delegate when the user clicks a button on an action sheet.

Customizing Behavior

- `willPresentActionSheet:` (page 830)
  Sent to the delegate before an action sheet is presented to the user.
- `didPresentActionSheet:` (page 830)
  Sent to the delegate after an action sheet is presented to the user.
- `actionSheet:willDismissWithButtonIndex:` (page 829)
  Sent to the delegate before an action sheet is dismissed.
- `actionSheet:didDismissWithButtonIndex:` (page 828)
  Sent to the delegate after an action sheet is dismissed from the screen.

Canceling

- `actionSheetCancel:` (page 829)
  Sent to the delegate before an action sheet is canceled.

Instance Methods

actionSheet:clickedButtonAtIndex:

Sent to the delegate when the user clicks a button on an action sheet.

```
- (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 iOS 2.0 and later.

Declared In
`UIActionSheet.h`

actionSheet:didDismissWithButtonIndex:

Sent to the delegate after an action sheet is dismissed from the screen.

```
- (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 iOS 2.0 and later.

See Also
- actionSheet:willDismissWithButtonIndex: (page 829)

Declared In
UIActionSheet.h

actionSheet:willDismissWithButtonIndex:

Sent to the delegate before an action sheet is dismissed.

```
- (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 iOS 2.0 and later.

See Also
- actionSheet:didDismissWithButtonIndex: (page 828)

Declared In
UIActionSheet.h

actionSheetCancel:

Sent to the delegate before an action sheet is canceled.

```
- (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 829) and `actionSheet:didDismissWithButtonIndex:` (page 828) methods are invoked after this method.

Availability

Available in iOS 2.0 and later.

Declared In

`UIActionSheet.h`

didPresentActionSheet:

Sent to the delegate after an action sheet is presented to the user.

```
- (void)didPresentActionSheet:(UIActionSheet *)actionSheet
```

Parameters

actionSheet

The action sheet that was displayed.

Availability

Available in iOS 2.0 and later.

See Also

`- willPresentActionSheet:` (page 830)

Declared In

`UIActionSheet.h`

willPresentActionSheet:

Sent to the delegate before an action sheet is presented to the user.

```
- (void)willPresentActionSheet:(UIActionSheet *)actionSheet
```

Parameters

actionSheet

The action sheet that is about to be displayed.

Availability

Available in iOS 2.0 and later.

See Also

`- didPresentActionSheet:` (page 830)

Declared In

`UIActionSheet.h`

UIAlertViewDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Framework/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIAlert.h
Related sample code	GKRocket GKTank

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 99) 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 832) 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 832) delegate method is invoked.

Optionally, you can implement the `alertViewCancel:` (page 833) 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 832).

Tasks

Responding to Actions

`- alertView:clickedButtonAtIndex:` (page 832)
Sent to the delegate when the user clicks a button on an alert view.

Customizing Behavior

- `willPresentAlertView:` (page 834)
  Sent to the delegate before a model view is presented to the user.
- `didPresentAlertView:` (page 834)
  Sent to the delegate after an alert view is presented to the user.
- `alertView:willDismissWithButtonIndex:` (page 833)
  Sent to the delegate before an alert view is dismissed.
- `alertView:didDismissWithButtonIndex:` (page 832)
  Sent to the delegate after an alert view is dismissed from the screen.

Canceling

- `alertViewCancel:` (page 833)
  Sent to the delegate before an alert view is canceled.

Instance Methods

alertView:clickedButtonAtIndex:

Sent to the delegate when the user clicks a button on an alert view.

```
- (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 iOS 2.0 and later.

Declared In
`UIAlertView.h`

alertView:didDismissWithButtonIndex:

Sent to the delegate after an alert view is dismissed from the screen.

```
- (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 iOS 2.0 and later.

See Also

- alertView:willDismissWithButtonIndex: (page 833)

Declared In

UIAlertView.h

alertView:willDismissWithButtonIndex:

Sent to the delegate before an alert view is dismissed.

```
- (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 iOS 2.0 and later.

See Also

- alertView:didDismissWithButtonIndex: (page 832)

Declared In

UIAlertView.h

alertViewCancel:

Sent to the delegate before an alert view is canceled.

```
- (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 833) and `alertView:didDismissWithButtonIndex:` (page 832) methods are invoked after this method.

Availability

Available in iOS 2.0 and later.

Declared In

`UIAlertView.h`

didPresentAlertView:

Sent to the delegate after an alert view is presented to the user.

```
- (void)didPresentAlertView:(UIAlertView *)alertView
```

Parameters

alertView

The alert view that was displayed.

Availability

Available in iOS 2.0 and later.

See Also

`- willPresentAlertView:` (page 834)

Declared In

`UIAlertView.h`

willPresentAlertView:

Sent to the delegate before a model view is presented to the user.

```
- (void)willPresentAlertView:(UIAlertView *)alertView
```

Parameters

alertView

The alert view that is about to be displayed.

Availability

Available in iOS 2.0 and later.

See Also

`- didPresentAlertView:` (page 834)

Declared In

`UIAlertView.h`

UIApplicationDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIApplication.h
Companion guides	iOS Application Programming Guide Local and Push Notification Programming Guide
Related sample code	GKRocket GKTank ScrollViewSuite SpeakHere WiTap

Overview

The `UIApplicationDelegate` protocol declares methods that are implemented by the delegate of the singleton `UIApplication` object. These methods provide you with information about key events in an application's execution such as when it finished launching, when it is about to be terminated, when memory is low, and when important changes occur. Implementing these methods gives you a chance to respond to these system events and respond appropriately.

One of the main jobs of the application delegate is to track the state transitions the application goes through while it is running. Prior to iOS 4.0, applications were either active, inactive, or not running. In iOS 4.0 and later, applications can also be running in the background or suspended. All of these transitions require a response from your application to ensure that it is doing the right thing. For example, a background application would need to stop updating its user interface. You provide the response to these transitions using the methods of the application delegate.

Launch time is also a particularly important point in an application's life cycle. In addition to the user launching an application by tapping its icon, an application can be launched in order to respond to a specific type of event. For example, it could be launched in response to an incoming push notification, it could be asked to open a file, or it could be launched to handle some background event that it had requested. In all of these cases, the options dictionary passed to the `application:didFinishLaunchingWithOptions:` (page 839) method provides information about the reason for the launch.

In situations where the application is already running, the methods of the application delegate are called in response to key changes. Although the methods of this protocol are optional, most or all of them should be implemented.

For more information about the launch cycle of an application and how you manage state transitions using the methods of the application delegate, see *iOS Application Programming Guide*. For more information about the `UIApplication` singleton class, see *UIApplication Class Reference*.

Tasks

Monitoring Application State Changes

- `application:didFinishLaunchingWithOptions:` (page 839)
  Tells the delegate when the application has launched and may have additional launch options to handle.
- `applicationDidBecomeActive:` (page 845)
  Tells the delegate that the application has become active.
- `applicationWillResignActive:` (page 849)
  Tells the delegate that the application is about to become inactive.
- `applicationDidEnterBackground:` (page 845)
  Tells the delegate that the application is now in the background.
- `applicationWillEnterForeground:` (page 848)
  Tells the delegate that the application is about to enter the foreground.
- `applicationWillTerminate:` (page 849)
  Tells the delegate when the application is about to terminate.
- `applicationDidFinishLaunching:` (page 846)
  Tells the delegate when the application has finished launching.

Opening a URL Resource

- `application:handleOpenURL:` (page 843)
  Asks the delegate to open a resource identified by URL.

Managing Status Bar Changes

- `application:willChangeStatusBarOrientation:duration:` (page 844)
  Tells the delegate when the interface orientation of the status bar is about to change.
- `application:didChangeStatusBarOrientation:` (page 838)
  Tells the delegate when the interface orientation of the status bar has changed.
- `application:willChangeStatusBarFrame:` (page 843)
  Tells the delegate when the frame of the status bar is about to change.
- `application:didChangeStatusBarFrame:` (page 837)
  Tells the delegate when the frame of the status bar has changed.

Responding to System Notifications

- `applicationDidReceiveMemoryWarning:` (page 846)
  Tells the delegate when the application receives a memory warning from the system.
- `applicationSignificantTimeChange:` (page 848)
  Tells the delegate when there is a significant change in the time.

Handling Remote Notifications

- `application:didReceiveRemoteNotification:` (page 841)
  Sent to the delegate when a running application receives a remote notification.
- `application:didRegisterForRemoteNotificationsWithDeviceToken:` (page 842)
  Sent to the delegate when the application successfully registers with Apple Push Service (APS).
- `application:didFailToRegisterForRemoteNotificationsWithError:` (page 838)
  Sent to the delegate when Apple Push Service cannot successfully complete the registration process.

Handling Local Notifications

- `application:didReceiveLocalNotification:` (page 840) *required method*
  Sent to the delegate when a running application receives a local notification. (required)

Responding to Content Protection Changes

- `applicationProtectedDataWillBecomeUnavailable:` (page 847) *required method*
  Tells the delegate that the protected files are about to become unavailable. (required)
- `applicationProtectedDataDidBecomeAvailable:` (page 847) *required method*
  Tells the delegate that protected files are available now. (required)

Instance Methods

application:didChangeStatusBarFrame:

Tells the delegate when the frame of the status bar has changed.

```
- (void)application:(UIApplication *)application
    didChangeStatusBarFrame:(CGRect)oldStatusBarFrame
```

Parameters

application
The delegating application object.

oldStatusBarFrame
The previous frame of the status bar, in screen coordinates.

Discussion
After calling this method, the application also posts a `UIApplicationDidChangeStatusBarFrameNotification` (page 135) notification to give interested objects a chance to respond to the change.

Availability
Available in iOS 2.0 and later.

See Also
`- application:willChangeStatusBarFrame:` (page 843)

Declared In
`UIApplication.h`

application:didChangeStatusBarOrientation:

Tells the delegate when the interface orientation of the status bar has changed.

```
- (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 "Monitoring Application State Changes" (page 836) for details.

Discussion
The delegate can get the current device orientation from the shared `UIDevice` object.

After calling this method, the application also posts a `UIApplicationDidChangeStatusBarOrientationNotification` (page 135) notification to give interested objects a chance to respond to the change.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

application:didFailToRegisterForRemoteNotificationsWithError:

Sent to the delegate when Apple Push Service cannot successfully complete the registration process.

```
- (void)application:(UIApplication *)application
    didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
```

Parameters

application
The application that initiated the remote-notification registration process.

error

An NSError object that encapsulates information why registration did not succeed. The application can choose to display this information to the user.

Discussion
The delegate receives this message after the registerForRemoteNotificationTypes: (page 122) method of UIApplication is invoked and there is an error in the registration process.

For more information about how to implement push notifications in your application, see *Local and Push Notification Programming Guide*.

Availability
Available in iOS 3.0 and later.

See Also
- application:didReceiveRemoteNotification: (page 841)
- application:didRegisterForRemoteNotificationsWithDeviceToken: (page 842)

Declared In
UIApplication.h

application:didFinishLaunchingWithOptions:

Tells the delegate when the application has launched and may have additional launch options to handle.

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
```

Parameters

application

The delegating application object.

launchOptions

A dictionary indicating the reason the application was launched (if any). The contents of this dictionary may be empty in situations where the user launched the application directly. See "Launch Options Keys" in *UIApplication Class Reference* for descriptions of these keys.

Return Value
NO if the application cannot handle the URL resource, otherwise return YES. The return value is ignored if the application is launched as a result of a remote notification.

Discussion
You should use this method to initialize your application and prepare it for running. It is called after your application has been launched and its main nib file has been loaded. At the time this method is called, your application is in the inactive state. At some point after this method returns, a subsequent delegate method is called to move your application to the active (foreground) state or the background state.

Note: It is highly recommended that you use this method to initialize your application and not the applicationDidFinishLaunching: (page 846) method.

If your application was launched by the system for a specific reason, the *launchOptions* dictionary contains data indicating the reason for the launch. The options dictionary typically contains keys for the following types of events:

- The user launched the application in response to the arrival of a remote notification. In this case, the dictionary contains the notification payload dictionary described in the `application:didReceiveRemoteNotification:` (page 841) method. (Key: `UIApplicationLaunchOptionsRemoteNotificationKey` (page 131))
- The user launched the application in response to the arrival of a local notification. In this case, the dictionary contains the local notification. (Key: `UIApplicationLaunchOptionsLocalNotificationKey` (page 132))
- Another application opened a URL that is owned by your application. In this case, the dictionary contains the bundle ID of the application and the URL to open. (Keys: `UIApplicationLaunchOptionsSourceApplicationKey` (page 131) and `UIApplicationLaunchOptionsURLKey` (page 131))
- The system asked your application to open a file that it claims to support. In this case, the dictionary contains a URL for the file to open. (Key: `UIApplicationLaunchOptionsURLKey` (page 131))
- The application tracks location updates in the background, was purged, and has now been relaunched. In this case, the dictionary contains a key indicating that the application was relaunched because of a new location event. (Key: `UIApplicationLaunchOptionsLocationKey` (page 132))
- The application is launched to open a document via the document-interaction controller (`UIDocumentInteractionController`). In this case the dictionary contains an annotation property list supplied by the originating application to communicate information to the receiving application. (Key: `UIApplicationLaunchOptionsAnnotationKey` (page 131))

Objects that are not the application delegate can access the same *launchOptions* dictionary values by observing the notification named `UIApplicationDidFinishLaunchingNotification` (page 136) and accessing the notification's `userInfo` dictionary. That notification is sent shortly after this method returns.

Availability
Available in iOS 3.0 and later.

Declared In
`UIApplication.h`

application:didReceiveLocalNotification:

Sent to the delegate when a running application receives a local notification. (required)

```
- (void)application:(UIApplication *)application
    didReceiveLocalNotification:(UILocalNotification *)notification
```

Parameters

application
The application that received the local notification.

notification
A local notification that encapsulates details about the notification, potentially including custom data.

Discussion
Local notifications are similar to remote push notifications, but differ in that they are scheduled, displayed, and received entirely on the same device. An application can create and schedule a local notification, and the operating system then delivers it at the schedule date and time. If it delivers it when the application is not active in the foreground, it displays an alert, badges the application icon, or plays a sound—whatever is

specified in the `UILocalNotification` object. If the application is running in the foreground, there is no alert, badging, or sound; instead, the `application:didReceiveLocalNotification:` method is called if the delegate implements it.

The delegate can implement this method if it wants to be notified that a local notification occurred. For example, if the application is a calendar application, it can enumerate its list of calendar events to determine which ones have due dates that have transpired or are about to transpire soon. It can also reset the application icon badge number, and it can access any custom data in the local-notification object's `userInfo` dictionary.

This method is invoked after `application:didFinishLaunchingWithOptions:` (page 839) (if that method is implemented).

Availability
Available in iOS 4.0 and later.

See Also
`- application:didReceiveRemoteNotification:` (page 841)

Declared In
`UIApplication.h`

application:didReceiveRemoteNotification:

Sent to the delegate when a running application receives a remote notification.

```
- (void)application:(UIApplication *)application
    didReceiveRemoteNotification:(NSDictionary *)userInfo
```

Parameters

application
The application that received the remote notification.

userInfo
A dictionary that contains information related to the remote notification, potentially including a badge number for the application icon, an alert sound, an alert message to display to the user, a notification identifier, and custom data. The provider originates it as a JSON-defined dictionary that iOS converts to an `NSDictionary` object; the dictionary may contain only property-list objects plus `NSNull`.

Discussion
The delegate receives this message when the application is running and a remote notification arrives for it. In response, the application typically connects with its provider and downloads the data waiting for it. It may also process the notification in any other way it deems useful.

The `userInfo` dictionary contains another dictionary that you can obtain using the `aps` key. You can access the contents of the `aps` dictionary—though you shouldn't need to in most cases—using the following keys:

`alert`—The value may either be a string for the alert message or a dictionary with two keys: `body` and `show-view`. The value of the former is the alert message and the latter is a Boolean (`false` or `true`). If `false`, the alert's View button is not shown. The default is to show the View button which, if the user taps it, launches the application.

`badge`—A number indicating the quantity of data items to download from the provider. This number is to be displayed on the application icon. The absence of a `badge` property indicates that any number currently badging the icon should be removed.

`sound`—The name of a sound file in the application bundle to play as an alert sound. If "default" is specified, the default sound should be played.

The `userInfo` dictionary may also have custom data defined by the provider according to the JSON schema. The properties for custom data should be specified at the same level as the `aps` dictionary. However, custom-defined properties should not be used for mass data transport because there is a strict size limit per notification (256 bytes) and delivery is not guaranteed.

If you implement `application:didFinishLaunchingWithOptions:` (page 839) to handle an incoming push notification that causes the launch of the application, this method is not invoked for that push notification.

For more information about how to implement push notifications in your application, see *Local and Push Notification Programming Guide*.

Availability
Available in iOS 3.0 and later.

See Also
- `application:didRegisterForRemoteNotificationsWithDeviceToken:` (page 842)
- `application:didFailToRegisterForRemoteNotificationsWithError:` (page 838)

Declared In
`UIApplication.h`

application:didRegisterForRemoteNotificationsWithDeviceToken:

Sent to the delegate when the application successfully registers with Apple Push Service (APS).

```
- (void)application:(UIApplication *)application
    didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
```

Parameters

application

The application that initiated the remote-notification registration process.

deviceToken

A token that identifies the device to APS. The token is an opaque data type because that is the form that the provider needs to submit to the APS servers when it sends a notification to a device. The APS servers require a binary format for performance reasons.

Note that the device token is different from the `uniqueIdentifier` (page 244) property of `UIDevice` because, for security and privacy reasons, it must change when the device is wiped.

Discussion
The delegate receives this message after the `registerForRemoteNotificationTypes:` (page 122) method of `UIApplication` is invoked and there is no error in the registration process. After receiving the device token, the application should connect with its provider and give the token to it. APS only pushes notifications to the application's device that are accompanied with this token. This method could be called in other rare circumstances, such as when the user launches an application after having restored a device from data that is not the device's backup data. In this exceptional case, the application won't know the new device's token until the user launches it.

For more information about how to implement push notifications in your application, see *Local and Push Notification Programming Guide*.

Availability
Available in iOS 3.0 and later.

See Also
- `application:didReceiveRemoteNotification:` (page 841)
- `application:didFailToRegisterForRemoteNotificationsWithError:` (page 838)

Declared In
`UIApplication.h`

application:handleOpenURL:

Asks the delegate to open a resource identified by URL.

```
- (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 *iOS Application 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
This method is not called if the delegate returns `NO` from its implementation of the `application:didFinishLaunchingWithOptions:` (page 839) method. if your application implements the `applicationDidFinishLaunching:` (page 846) method instead of `application:didFinishLaunchingWithOptions:`, this method is called to open the specified URL after the application has been initialized.

If a URL arrives while your application is suspended or running in the background, the system moves your application to the foreground prior to calling this method.

There is no equivalent notification for this delegation method.

Availability
Available in iOS 2.0 and later.

See Also
`openURL:` (page 121) (`UIApplication`)
- `application:didFinishLaunchingWithOptions:` (page 839)

Declared In
`UIApplication.h`

application:willChangeStatusBarFrame:

Tells the delegate when the frame of the status bar is about to change.

```
- (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 126) message and is about to change the interface orientation.

After calling this method, the application also posts a `UIApplicationWillChangeStatusBarFrameNotification` (page 137) notification to give interested objects a chance to respond to the change.

Availability

Available in iOS 2.0 and later.

See Also

`- application:didChangeStatusBarFrame:` (page 837)

Declared In

`UIApplication.h`

application:willChangeStatusBarOrientation:duration:

Tells the delegate when the interface orientation of the status bar is about to change.

```
- (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 "Monitoring Application State Changes" (page 836) 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.

After calling this method, the application also posts a `UIApplicationWillChangeStatusBarOrientationNotification` (page 137) notification to give interested objects a chance to respond to the change.

Availability

Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

applicationDidBecomeActive:

Tells the delegate that the application has become active.

```
- (void)applicationDidBecomeActive:(UIApplication *)application
```

Parameters

application
The singleton application instance.

Discussion
This method is called to let your application know that it moved from the inactive to active state. This can occur because your application was launched by the user or the system. Applications can also return to the active state if the user chooses to ignore an interruption (such as an incoming phone call or SMS message) that sent the application temporarily to the inactive state.

You should use this method to restart any tasks that were paused (or not yet started) while the application was inactive. For example, you could use it to restart timers or throttle up OpenGL ES frame rates. If your application was previously in the background, you could also use it to refresh your application's user interface.

After calling this method, the application also posts a `UIApplicationDidBecomeActiveNotification` (page 135) notification to give interested objects a chance to respond to the transition.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

applicationDidEnterBackground:

Tells the delegate that the application is now in the background.

```
- (void)applicationDidEnterBackground:(UIApplication *)application
```

Parameters

application
The singleton application instance.

Discussion
In iOS 4.0 and later, this method is called instead of the `applicationWillTerminate:` method when the user quits an application that supports background execution. You should use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later. You should also disable updates to your application's user interface and avoid using some types of shared system resources (such as the user's contacts database). It is also imperative that you avoid using OpenGL ES in the background.

Your implementation of this method has approximately five seconds to perform any tasks and return. If you need additional time to perform any final tasks, you can request additional execution time from the system.

The application also posts a `UIApplicationDidEnterBackgroundNotification` (page 135) notification around the same time it calls this method to give interested objects a chance to respond to the transition.

For more information about how to transition gracefully to the background, and for information about how to start background tasks at quit time, see *iOS Application Programming Guide*.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

applicationDidFinishLaunching:

Tells the delegate when the application has finished launching.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
```

Parameters

application
The delegating application object.

Discussion
This method is used in earlier versions of iOS to initialize the application and prepare it to run. In iOS 3.0 and later, you should use the `application:didFinishLaunchingWithOptions:` (page 839) instead.

Your implementation of this method should create your application's user interface and initialize the application's data structures. If your application persists its state between launches, you would also use this method to restore your application to its previous state.

After calling this method, the application also posts a `UIApplicationDidFinishLaunchingNotification` (page 136) notification to give interested objects a chance to respond to the initialization cycle.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

applicationDidReceiveMemoryWarning:

Tells the delegate when the application receives a memory warning from the system.

```
- (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 iOS 2.0 and later.

See Also
`didReceiveMemoryWarning` (page 763) (UIViewController)
`UIApplicationDidReceiveMemoryWarningNotification` (page 136)

Declared In
`UIApplication.h`

applicationProtectedDataDidBecomeAvailable:

Tells the delegate that protected files are available now. (required)

```
- (void)applicationProtectedDataDidBecomeAvailable:(UIApplication *)application
```

Parameters
application
The delegating application object.

Discussion
On a device that uses content protection, protected files are stored in an encrypted form and made available only while the device is unlocked. This notification lets your application know that the device is now unlocked and that you may access any protected files once again.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

applicationProtectedDataWillBecomeUnavailable:

Tells the delegate that the protected files are about to become unavailable. (required)

```
- (void)applicationProtectedDataWillBecomeUnavailable:(UIApplication *)application
```

Parameters
application
The delegating application object.

Discussion
On a device that uses content protection, protected files are stored in an encrypted form and made available only while the device is unlocked. This notification lets your application know that the device is about to be locked and that any protected files it is currently accessing will be unavailable shortly.

If your application is currently accessing a protected file, you can use this method to release any references to that file. Although it is not an error to access the file while the device is locked, any attempts to do so will fail. Therefore, if your application depends on the file, you might want to take steps to avoid using that file while the device is locked.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

applicationSignificantTimeChange:

Tells the delegate when there is a significant change in the time.

```
- (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.

Prior to calling this method, the application also posts a `UIApplicationSignificantTimeChangeNotification` (page 137) notification to give interested objects a chance to respond to the change.

If your application is currently suspended, this message is queued until your application returns to the foreground, at which point it is delivered. If multiple time changes occur, only the most recent one is delivered.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

applicationWillEnterForeground:

Tells the delegate that the application is about to enter the foreground.

```
- (void)applicationWillEnterForeground:(UIApplication *)application
```

Parameters

application
The singleton application instance.

Discussion
In iOS 4.0 and later, this method is called as part of the transition from the background to the inactive state. You can use this method to undo many of the changes you made to your application upon entering the background. The call to this method is invariably followed by a call to the `applicationDidBecomeActive:` (page 845) method, which then moves the application from the inactive to the active state.

The application also posts a `UIApplicationWillEnterForegroundNotification` (page 137) notification shortly before calling this method to give interested objects a chance to respond to the transition.

Availability
Available in iOS 4.0 and later.

Declared In
`UIApplication.h`

applicationWillResignActive:

Tells the delegate that the application is about to become inactive.

```
- (void)applicationWillResignActive:(UIApplication *)application
```

Parameters

application
The singleton application instance.

Discussion
This method is called to let your application know that it is about to move from the active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state. An application in the inactive state continues to run but does not dispatch incoming events to responders.

You should use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game. An application in the inactive state should do minimal work while it waits to transition to either the active or background state.

After calling this method, the application also posts a `UIApplicationWillResignActiveNotification` (page 138) notification to give interested objects a chance to respond to the transition.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

applicationWillTerminate:

Tells the delegate when the application is about to terminate.

```
- (void)applicationWillTerminate:(UIApplication *)application
```

Parameters

application

The delegating application object.

Discussion

This method lets your application know that it is about to be terminated and purged from memory entirely. You should use this method to perform any final clean-up tasks for your application, such as freeing shared resources, saving user data, invalidating timers, and storing enough application state to reconstitute your application's interface when it is relaunched. Your implementation of this method has approximately five seconds to perform any tasks and return. If the method does not return before time expires, the system may kill the process altogether.

For applications that do not support background execution or are linked against iOS 3.x or earlier, this method is always called when the user quits the application. For applications that support background execution, this method is generally not called when the user quits the application because the application simply moves to the background in that case. However, this method may be called in situations where the application is running in the background (not suspended) and the system needs to terminate it for some reason.

After calling this method, the application also posts a `UIApplicationWillTerminateNotification` (page 138) notification to give interested objects a chance to respond to the transition.

Availability

Available in iOS 2.0 and later.

See Also

- `applicationDidEnterBackground:` (page 845)
- `application:didFinishLaunchingWithOptions:` (page 839)

Declared In

`UIApplication.h`

UIDocumentInteractionControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIDocumentInteractionController.h

Overview

The `UIDocumentInteractionControllerDelegate` protocol includes methods that you use to respond to events in a document interaction controller object. You can use this protocol to determine when document previews are displayed and when a document is about to be opened by another application. You also use it to support commands (such as the copy command) that affect documents.

If you use a document interaction controller to display a document preview, your delegate must implement the `documentInteractionControllerViewControllerForPreview:` (page 856) method. All other methods of the protocol are optional.

For more information about using a document interaction controller object, see *UIDocumentInteractionController Class Reference*.

Tasks

Configuring the Parent View Controller

- `documentInteractionControllerViewControllerForPreview:` (page 856)
  Asks the delegate for the view controller to use when presenting the document preview.
- `documentInteractionControllerViewForPreview:` (page 857)
  Asks the delegate for the view to use as the starting point for animating the display of the document preview.
- `documentInteractionControllerRectForPreview:` (page 856)
  Asks the delegate for the rectangle to use as the starting point for animating the display of the document preview.

Presenting the User Interface

- `documentInteractionControllerWillBeginPreview:` (page 857)
  Tells the delegate that the interaction controller is about to display a preview for its document.
- `documentInteractionControllerDidEndPreview:` (page 855)
  Tells the delegate that the interaction controller dismissed its document preview.
- `documentInteractionControllerWillPresentOptionsMenu:` (page 858)
  Tells the delegate that the interaction controller is about to display an options menu.
- `documentInteractionControllerDidDismissOptionsMenu:` (page 855)
  Tells the delegate that the interaction controller dismissed its options menu.
- `documentInteractionControllerWillPresentOpenInMenu:` (page 858)
  Tells the delegate that the interaction controller is about to display an Open In menu.
- `documentInteractionControllerDidDismissOpenInMenu:` (page 854)
  Tells the delegate that the interaction controller dismissed its Open In menu.

Opening Files

- `documentInteractionController:willBeginSendingToApplication:` (page 854)
  Tells the delegate that the document is about to be opened by the specified application
- `documentInteractionController:didEndSendingToApplication:` (page 853)
  Tells the delegate that the document was handed off to the specified application

Managing Actions

- `documentInteractionController:canPerformAction:` (page 852)
  Asks the delegate whether the specified action can be performed on the target document.
- `documentInteractionController:performAction:` (page 853)
  Tells the delegate to perform the specified action on the target document.

Instance Methods

documentInteractionController:canPerformAction:

Asks the delegate whether the specified action can be performed on the target document.

```
- (BOOL)documentInteractionController:(UIDocumentInteractionController *)controller
    canPerformAction:(SEL)action
```

Parameters

controller
The document interaction controller managing the target document.

action
A selector for the action method in question.

Return Value
`YES` if the specified action is supported on the target document or `NO` if it is not. If you do not implement this method, the return value is assumed to be `NO`.

Discussion
When building the options menu, the document interaction controller calls this method to find out if your application is able to perform the indicated action. (If you implement this method, you must also implement the `documentInteractionController:performAction:` (page 853) method for the given action.)

Currently only the `copy:` action is supported. If your application supports copying the document contents, you should return `YES` and use the `documentInteractionController:performAction:` to perform the copy operation.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionController:didEndSendingToApplication:

Tells the delegate that the document was handed off to the specified application

```
- (void)documentInteractionController:(UIDocumentInteractionController *)controller
    didEndSendingToApplication:(NSString *)application
```

Parameters
controller
The document interaction controller whose document is about to be opened.

application
The bundle identifier of the application that is about to open the document. This value corresponds to the value in the `CFBundleIdentifier` key of the application's `Info.plist` file.

Discussion
This method is called after the document information has been saved for the specified application.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionController:performAction:

Tells the delegate to perform the specified action on the target document.

```
- (BOOL)documentInteractionController:(UIDocumentInteractionController *)controller
    performAction:(SEL)action
```

Parameters
controller
The document interaction controller managing the target document.

action

A selector representing the action to perform. You can invoke this selector directly on the object responsible for performing the action or use it to call the appropriate method.

Return Value
`YES` if the action was performed successfully or `NO` if it was not.

Discussion
Currently only the `copy:` action is supported. Your implementation of this method should write the contents of the document to the pasteboard when the action is performed.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionController:willBeginSendingToApplication:

Tells the delegate that the document is about to be opened by the specified application

```
- (void)documentInteractionController:(UIDocumentInteractionController *)controller
    willBeginSendingToApplication:(NSString *)application
```

Parameters

controller

The document interaction controller whose document is about to be opened.

application

The bundle identifier of the application that is about to open the document. This value corresponds to the value in the `CFBundleIdentifier` key of the application's `Info.plist` file.

Discussion
This method is called when the user chooses to open a document, which could occur from within a document preview. When a document is passed to another application, the contents of the document interaction controller's `annotation` (page 253) property are passed with it. You can use this method to configure the contents of that property or prepare your own application for handing off the document.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerDidDismissOpenInMenu:

Tells the delegate that the interaction controller dismissed its Open In menu.

```
- (void)documentInteractionControllerDidDismissOpenInMenu:(UIDocumentInteractionController
    *)controller
```

Parameters

controller

The document interaction controller that dismissed its menu.

Discussion

You can use this method to remove any additional views or content you placed underneath the Open In menu in your `documentInteractionControllerWillPresentOpenInMenu:` (page 858) method.

Availability

Available in iOS 3.2 and later.

Declared In

`UIDocumentInteractionController.h`

documentInteractionControllerDidDismissOptionsMenu:

Tells the delegate that the interaction controller dismissed its options menu.

```
- (void)documentInteractionControllerDidDismissOptionsMenu:(UIDocumentInteractionController
    *)controller
```

Parameters

controller

The document interaction controller that dismissed its options menu.

Discussion

You can use this method to remove any additional views or content you placed underneath the options menu in your `documentInteractionControllerWillPresentOptionsMenu:` (page 858) method.

Availability

Available in iOS 3.2 and later.

Declared In

`UIDocumentInteractionController.h`

documentInteractionControllerDidEndPreview:

Tells the delegate that the interaction controller dismissed its document preview.

```
- (void)documentInteractionControllerDidEndPreview:(UIDocumentInteractionController
    *)controller
```

Parameters

controller

The document interaction controller that dismissed its document preview.

Discussion

This method is called after the view containing the document preview has been removed from the application's key window. You can use this notification to remove any interface elements you set up behind the preview elements.

Availability

Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerRectForPreview:

Asks the delegate for the rectangle to use as the starting point for animating the display of the document preview.

```
- (CGRect)documentInteractionControllerRectForPreview:(UIDocumentInteractionController
  *)controller
```

Parameters

controller

The document interaction controller requesting the starting rectangle.

Return Value
A rectangle in the coordinate system of the view returned by the `documentInteractionControllerViewForPreview:` (page 857) method.

Discussion
If you do not implement the `documentInteractionControllerViewForPreview:` method, or if you do implement it but return a `nil` value, this method is not called. If you do not implement this method, the starting rectangle is assumed to be the bounds of the view returned by the `documentInteractionControllerViewForPreview:` (page 857) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerViewControllerForPreview:

Asks the delegate for the view controller to use when presenting the document preview.

```
- (UIViewController
  *)documentInteractionControllerViewControllerForPreview:(UIDocumentInteractionController
  *)controller
```

Parameters

controller

The document interaction controller requesting the parent view controller.

Return Value
The view controller to use when presenting the document preview. The return value must not be `nil`.

Discussion
Although technically optional, this method is required if your application attempts to display a preview for a document. The view controller returned by this method is used as the parent for the document preview.

If you return a navigation controller from this method, the document interaction controller is pushed onto the navigation stack using the standard navigation controller animations. If you return any other type of view controller, the document interaction controller is displayed modally, in which case, the view controller you return must be capable of presenting a modal view controller.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerViewForPreview:

Asks the delegate for the view to use as the starting point for animating the display of the document preview.

```
- (UIView
    *)documentInteractionControllerViewForPreview:(UIDocumentInteractionController
     *)controller
```

Parameters

controller

The document interaction controller requesting the starting view.

Return Value
The view to use as the starting point for the animation or `nil` if you want the document preview to fade into place.

Discussion
By default, the starting rectangle for the animation is set to the bounds of the returned view. To specify a different starting rectangle, you must also override the `documentInteractionControllerRectForPreview:` (page 856) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerWillBeginPreview:

Tells the delegate that the interaction controller is about to display a preview for its document.

```
-
    (void)documentInteractionControllerWillBeginPreview:(UIDocumentInteractionController
     *)controller
```

Parameters

controller

The document interaction controller that is about to preview its document.

Discussion
This method is called shortly before the view containing the document preview is presented modally. You can use this notification to set up any additional interface elements behind the preview elements.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerWillPresentOpenInMenu:

Tells the delegate that the interaction controller is about to display an Open In menu.

```
- (void)documentInteractionControllerWillPresentOpenInMenu:(UIDocumentInteractionController
  *)controller
```

Parameters

controller
The document interaction controller that is about to display a menu.

Discussion
The Open In menu is used to select an application for opening the current file. You can use this method to update your user interface in response to displaying the menu.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

documentInteractionControllerWillPresentOptionsMenu:

Tells the delegate that the interaction controller is about to display an options menu.

```
- (void)documentInteractionControllerWillPresentOptionsMenu:(UIDocumentInteractionController
  *)controller
```

Parameters

controller
The document interaction controller that is about to display an options menu.

Discussion
The Open In menu is used to present the user with options for previewing the document, opening it in an application, or copying its contents. You can use this method to update your user interface in response to displaying the menu.

Availability
Available in iOS 3.2 and later.

Declared In
`UIDocumentInteractionController.h`

UIGestureRecognizerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIGestureRecognizer.h
Companion guide	Event Handling Guide for iOS
Related sample code	SimpleGestureRecognizers

Overview

Delegates of a gesture recognizer—that is, an instance of a concrete subclass of `UIGestureRecognizer`—adopt the `UIGestureRecognizerDelegate` protocol to fine-tune an application's gesture-recognition behavior. They receive messages from the gesture recognizer, and their responses to these messages enable them to affect the operation of the gesture recognizer or permit the simultaneous operation of two gesture recognizers.

Tasks

Regulating Gesture Recognition

`- gestureRecognizerShouldBegin:` (page 861)
Asks the delegate if a gesture recognizer should begin interpreting touches.

`- gestureRecognizer:shouldReceiveTouch:` (page 860)
Ask the delegate if a gesture recognizer should receive an object representing a touch.

Controlling Simultaneous Gesture Recognition

`- gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:` (page 860)
Asks the delegate if two gesture recognizers should be allowed to recognize gestures simultaneously.

Instance Methods

gestureRecognizer:shouldReceiveTouch:

Ask the delegate if a gesture recognizer should receive an object representing a touch.

```
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
    shouldReceiveTouch:(UITouch *)touch
```

Parameters

gestureRecognizer

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

touch

A `UITouch` object from the current multi-touch sequence.

Return Value

`YES` (the default) to allow the gesture recognizer to examine the touch object, `NO` to prevent the gesture recognizer from seeing this touch object.

Discussion

This method is called before `touchesBegan:withEvent:` (page 292) is called on the gesture recognizer for a new touch.

Availability

Available in iOS 3.2 and later.

See Also

`- gestureRecognizerShouldBegin:` (page 861)

Declared In

`UIGestureRecognizer.h`

gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:

Asks the delegate if two gesture recognizers should be allowed to recognize gestures simultaneously.

```
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
    shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer
    *)otherGestureRecognizer
```

Parameters

gestureRecognizer

An instance of a subclass of the abstract base class `UIGestureRecognizer`. This is the object sending the message to the delegate.

otherGestureRecognizer

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

Return Value

`YES` to allow both *gestureRecognizer* and *otherGestureRecognizer* to recognize their gestures simultaneously. The default implementation returns `NO`—no two gestures can be recognized simultaneously.

Discussion
This method is called when recognition of a gesture by either *gestureRecognizer* or *otherGestureRecognizer* would block the other gesture recognizer from recognizing its gesture. Note that returning YES is guaranteed to allow simultaneous recognition; returning NO, on the other hand, is not guaranteed to prevent simultaneous recognition because the other gesture recognizer's delegate may return YES.

Availability
Available in iOS 3.2 and later.

Declared In
UIGestureRecognizer.h

gestureRecognizerShouldBegin:

Asks the delegate if a gesture recognizer should begin interpreting touches.

```
- (BOOL)gestureRecognizerShouldBegin:(UIGestureRecognizer *)gestureRecognizer
```

Parameters

gestureRecognizer
An instance of a subclass of the abstract base class UIGestureRecognizer. This gesture-recognizer object is about to begin processing touches to determine if its gesture is occurring.

Return Value
YES (the default) to tell the gesture recognizer to proceed with interpreting touches, NO to prevent it from attempting to recognize its gesture.

Discussion
This method is called when a gesture recognizer attempts to transition out of the UIGestureRecognizerStatePossible (page 296) state. Returning NO causes the gesture recognizer to transition to the UIGestureRecognizerStateFailed (page 297) state.

Availability
Available in iOS 3.2 and later.

See Also
- gestureRecognizer:shouldReceiveTouch: (page 860)

Declared In
UIGestureRecognizer.h

UIImagePickerControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	

Overview

The `UIImagePickerControllerDelegate` protocol defines methods that your delegate object must implement to interact with the image picker interface. The methods of this protocol notify your delegate when the user either picks an image or movie, or cancels the picker operation.

The delegate methods are responsible for dismissing the picker when the operation completes. To dismiss the picker, call the `dismissModalViewControllerAnimated:` method of the parent controller responsible for displaying the `UIImagePickerController` object.

To save a still image to the user's Saved Photos album, use the `UIImageWriteToSavedPhotosAlbum` (page 1045) function. To save a movie to the user's Saved Photos album, use the `UISaveVideoAtPathToSavedPhotosAlbum` (page 1049) function.

Tasks

Closing the Picker

- `imagePickerController:didFinishPickingMediaWithInfo:` (page 864)
  Tells the delegate that the user picked a still image or movie.
- `imagePickerController:didFinishPickingImage:editingInfo:` (page 864)
  Tells the delegate that the user picked an image. (Deprecated. Use `imagePickerController:didFinishPickingMediaWithInfo:` (page 864) instead.)
- `imagePickerControllerDidCancel:` (page 865)
  Tells the delegate that the user cancelled the pick operation.

Instance Methods

imagePickerController:didFinishPickingImage:editingInfo:

Tells the delegate that the user picked an image. (Deprecated in iOS 3.0. Use `imagePickerController:didFinishPickingMediaWithInfo:` (page 864) instead.)

```
- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingImage:(UIImage *)image
    editingInfo:(NSDictionary *)editingInfo
```

Parameters

picker

The controller object managing the image picker interface.

image

The image that the user picked. If user editing is enabled, this may be a cropped and adjusted version of the original image. In this case, the original image, and the 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 866).

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 available in the `editingInfo` parameter. If you don't need the editing information, simply use the image in the *image* parameter as is.

Special Considerations

This deprecated method supports picking only still pictures. The replacement method, `imagePickerController:didFinishPickingMediaWithInfo:` (page 864), supports picking movies as well as still pictures.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 3.0.

Declared In

`UIImagePickerController.h`

imagePickerController:didFinishPickingMediaWithInfo:

Tells the delegate that the user picked a still image or movie.

```
- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingMediaWithInfo:(NSDictionary *)info
```

Parameters

picker

The controller object managing the image picker interface.

info

A dictionary containing the original image and the edited image, if an image was picked; or a filesystem URL for the movie, if a movie was picked. The dictionary also contains any relevant editing information. The keys for this dictionary are listed in "Editing Information Keys" (page 866).

Discussion

Your delegate object's implementation of this method should pass the specified media on to any custom code that needs it, and should then dismiss the picker view.

When editing is enabled, the image picker view presents the user with a preview of the currently selected image or movie 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 or movie, the editing information is available in the `info` parameter. The original image is also returned in the `info` parameter.

If you set the image picker's `showsCameraControls` (page 322) property to `NO` and provide your own custom controls, you can take multiple pictures before dismissing the image picker interface. However, if you set that property to `YES`, your delegate must dismiss the image picker interface after the user takes one picture or cancels the operation.

Implementation of this method is optional, but expected.

Availability

Available in iOS 3.0 and later.

Declared In

`UIImagePickerController.h`

imagePickerControllerDidCancel:

Tells the delegate that the user cancelled the pick operation.

```
- (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 by calling the `dismissModalViewControllerAnimated:` method of the parent view controller.

Implementation of this method is optional, but expected.

Availability

Available in iOS 2.0 and later.

Declared In

`UIImagePickerController.h`

Constants

Editing Information Keys

Keys for the editing information dictionary passed to the delegate.

```
NSString *const UIImagePickerControllerMediaType;
NSString *const UIImagePickerControllerOriginalImage;
NSString *const UIImagePickerControllerEditedImage;
NSString *const UIImagePickerControllerCropRect;
NSString *const UIImagePickerControllerMediaURL;
```

Constants

`UIImagePickerControllerMediaType`

Specifies the media type selected by the user.

The value for this key is an `NSString` object containing a type code such as `kUTTypeImage` or `kUTTypeMovie`.

Available in iOS 3.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerOriginalImage`

Specifies the original, uncropped image selected by the user.

The value for this key is a `UIImage` object.

Available in iOS 2.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerEditedImage`

Specifies an image edited by the user.

The value for this key is a `UIImage` object.

Available in iOS 3.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerCropRect`

Specifies the cropping rectangle that was applied to the original image.

The value for this key is an `NSValue` object containing a `CGRect` opaque type.

Available in iOS 2.0 and later.

Declared in `UIImagePickerController.h`.

`UIImagePickerControllerMediaURL`

Specifies the filesystem URL for the movie.

The value for this key is an `NSURL` object.

Available in iOS 3.0 and later.

Declared in `UIImagePickerController.h`.

UIKeyInput Protocol Reference

Conforms to	UITextInputTraits
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h

Overview

A subclass of `UIResponder` can adopt this protocol to implement simple text entry. When instances of this subclass are the first responder, the system keyboard is displayed.

Only a small subset of the available keyboards and languages are available to classes that adopt this protocol.

Tasks

Inserting and Deleting Text

- `insertText:` (page 868) *required method*
  Insert a character into the displayed text. (required)
- `deleteBackward` (page 867) *required method*
  Delete a character from the displayed text. (required)
- `hasText` (page 868) *required method*
  A Boolean value that indicates whether the text-entry objects has any text. (required)

Instance Methods

deleteBackward

Delete a character from the displayed text. (required)

```
- (void)deleteBackward
```

Discussion
Remove the character just before the cursor from your class's backing store and redisplay the text.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

hasText

A Boolean value that indicates whether the text-entry objects has any text. (required)

```
- (BOOL)hasText
```

Return Value
`YES` if the backing store has textual content, `NO` otherwise.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

insertText:

Insert a character into the displayed text. (required)

```
- (void)insertText:(NSString *)text
```

Parameters
text
A string object representing the character typed on the system keyboard.

Discussion
Add the character `text` to your class's backing store at the index corresponding to the cursor and redisplay the text.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

UINavigationBarDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 871) and `navigationBar:shouldPopItem:` (page 871) 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 870) method to update the view below the navigation bar to reflect the new item. Similarly, you implement the `navigationBar:didPopItem:` (page 870) method to replace the view below the navigation bar.

Tasks

Pushing Items

- `navigationBar:shouldPushItem:` (page 871)
  Returns a Boolean value indicating whether the navigation bar should push an item.
- `navigationBar:didPushItem:` (page 870)
  Tells the delegate that an item was pushed onto the navigation bar.

Popping Items

- `navigationBar:shouldPopItem:` (page 871)
  Returns a Boolean value indicating whether the navigation bar should pop an item.

Instance Methods

navigationBar:didPopItem:

Tells the delegate that an item was popped from the navigation bar.

```
- (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 iOS 2.0 and later.

See Also
- navigationBar:shouldPopItem: (page 871)

Declared In
UINavigationBar.h

navigationBar:didPushItem:

Tells the delegate that an item was pushed onto the navigation bar.

```
- (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 iOS 2.0 and later.

See Also
- navigationBar:shouldPushItem: (page 871)

Declared In
UINavigationBar.h

navigationBar:shouldPopItem:

Returns a Boolean value indicating whether the navigation bar should pop an item.

```
- (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 iOS 2.0 and later.

See Also
- navigationBar:didPopItem: (page 870)

Declared In
UINavigationBar.h

navigationBar:shouldPushItem:

Returns a Boolean value indicating whether the navigation bar should push an item.

```
- (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 iOS 2.0 and later.

See Also
`- navigationBar:didPushItem:` (page 870)

Declared In
`UINavigationBar.h`

UINavigationControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Framework/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UINavigationController.h

Overview

The `UINavigationControllerDelegate` protocol defines methods a navigation controller delegate can implement to change the behavior when view controllers are pushed and popped from the stack of a navigation controller.

Tasks

Customizing Behavior

`- navigationController:willShowViewController:animated:` (page 874)
Sent to the receiver just before the navigation controller displays a view controller's view and navigation item properties.

`- navigationController:didShowViewController:animated:` (page 873)
Sent to the receiver just after the navigation controller displays a view controller's view and navigation item properties.

Instance Methods

navigationController:didShowViewController:animated:

Sent to the receiver just after the navigation controller displays a view controller's view and navigation item properties.

```
- (void)navigationController:(UINavigationController *)navigationController
    didShowViewController:(UIViewController *)viewController  animated:(BOOL)animated
```

Parameters

navigationController
The navigation controller that is showing the properties of a view controller.

viewController
The view controller whose view and navigation item properties are being shown.

animated
YES to animate the transition; otherwise, NO.

Availability
Available in iOS 2.0 and later.

See Also
- navigationController:willShowViewController:animated: (page 874)

Declared In
UINavigationController.h

navigationController:willShowViewController:animated:

Sent to the receiver just before the navigation controller displays a view controller's view and navigation item properties.

```
- (void)navigationController:(UINavigationController *)navigationController
    willShowViewController:(UIViewController *)viewController
    animated:(BOOL)animated
```

Parameters

navigationController
The navigation controller that is showing the properties of a view controller.

viewController
The view controller whose view and navigation item properties are being shown.

animated
YES to animate the transition; otherwise, NO.

Availability
Available in iOS 2.0 and later.

See Also
- navigationController:didShowViewController:animated: (page 873)

Declared In
UINavigationController.h

UIPickerViewAccessibilityDelegate Protocol Reference

Conforms to	UIPickerViewDelegate
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 4.0 and later.
Declared in	UIAccessibilityAdditions.h

Overview

The `UIPickerViewAccessibilityDelegate` protocol defines methods you can implement to provide accessibility information for individual components of a picker view.

Tasks

Providing Descriptive Information

`- pickerView:accessibilityLabelForComponent:` (page 876)
Returns a label that identifies the picker view component.

`- pickerView:accessibilityHintForComponent:` (page 875)
Returns a hint that describes the result of performing an action on the picker view component.

Instance Methods

pickerView:accessibilityHintForComponent:

Returns a hint that describes the result of performing an action on the picker view component.

```
- (NSString *)pickerView:(UIPickerView *)pickerView
    accessibilityHintForComponent:(NSInteger)component
```

Return Value
A brief description, in a localized string, of the result of performing an action on the picker view component.

Discussion
Implement this optional method to ensure that the accessibility element representing the picker view provides an appropriate hint for each component. For in-depth information on how to create an appropriate hint, see "Guidelines for Creating Hints" in *Accessibility Programming Guide for iOS*.

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibilityAdditions.h`

pickerView:accessibilityLabelForComponent:

Returns a label that identifies the picker view component.

```
- (NSString *)pickerView:(UIPickerView *)pickerView
    accessibilityLabelForComponent:(NSInteger)component
```

Return Value
A succinct label, in a localized string, that identifies the picker view component.

Discussion
Implement this optional method to ensure that the accessibility element representing the picker view provides an appropriate label for each component. For in-depth information on how to create an appropriate label, see "Crafting Useful Labels and Hints" in *Accessibility Programming Guide for iOS*.

Availability
Available in iOS 4.0 and later.

Declared In
`UIAccessibilityAdditions.h`

UIPickerViewDataSource Protocol Reference

Conforms to NSObject

Framework /System/Library/Frameworks/UIKit.framework

Availability Available in iOS 2.0 and later.

Declared in

Overview

The `UIPickerViewDataSource` protocol must be adopted by an object that mediates between a `UIPickerView` object and your application's data model for that picker view. The data source provides the picker view with the number of components, and the number of rows in each component, for displaying the picker view data. Both methods in this protocol are required.

Tasks

Providing Counts for the Picker View

`- numberOfComponentsInPickerView:` (page 877) *required method*
Called by the picker view when it needs the number of components. (required)

`- pickerView:numberOfRowsInComponent:` (page 878) *required method*
Called by the picker view when it needs the number of rows for a specified component. (required)

Instance Methods

numberOfComponentsInPickerView:

Called by the picker view when it needs the number of components. (required)

```
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
```

Parameters

pickerView
The picker view requesting the data.

Return Value
The number of components (or "columns") that the picker view should display.

Availability
Available in iOS 2.0 and later.

Declared In
`UIPickerView.h`

pickerView:numberOfRowsInComponent:

Called by the picker view when it needs the number of rows for a specified component. (required)

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
    numberOfRowsInComponent:(NSInteger)component
```

Parameters

pickerView
The picker view requesting the data.

component
A zero-indexed number identifying a component of *pickerView*. Components are numbered left-to-right.

Return Value
The number of rows for the component.

Availability
Available in iOS 2.0 and later.

Declared In
`UIPickerView.h`

UIPickerViewDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 this 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 *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 880)
Called by the picker view when it needs the row height to use for drawing row content.

`- pickerView:widthForComponent:` (page 882)
Called by the picker view when it needs the row width to use for drawing row content.

Setting the Content of Component Rows

The methods in this group are marked `@optional`. However, to use a picker view, you must implement either the `pickerView:titleForRow:forComponent:` (page 881) or the `pickerView:viewForRow:forComponent:reusingView:` (page 881) method to provide the content of component rows.

`- pickerView:titleForRow:forComponent:` (page 881)
Called by the picker view when it needs the title to use for a given row in a given component.

- pickerView:viewForRow:forComponent:reusingView: (page 881)
Called by the picker view when it needs the view to use for a given row in a given component.

Responding to Row Selection

- pickerView:didSelectRow:inComponent: (page 880)
Called by the picker view when the user selects a row in a component.

Instance Methods

pickerView:didSelectRow:inComponent:

Called by the picker view when the user selects a row in a component.

```
- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
    inComponent:(NSInteger)component
```

Parameters

pickerView
An object representing the picker view requesting the data.

row
A zero-indexed number identifying a row of *component*. Rows are numbered top-to-bottom.

component
A zero-indexed number identifying a component of *pickerView*. Components are numbered left-to-right.

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 iOS 2.0 and later.

Declared In
UIPickerView.h

pickerView:rowHeightForComponent:

Called by the picker view when it needs the row height to use for drawing row content.

```
- (CGFloat)pickerView:(UIPickerView *)pickerView
    rowHeightForComponent:(NSInteger)component
```

Parameters

pickerView
The picker view requesting this information.

component
A zero-indexed number identifying a component of *pickerView*. Components are numbered left-to-right.

Return Value
A float value indicating the height of the row in points.

Availability
Available in iOS 2.0 and later.

Declared In
UIPickerView.h

pickerView:titleForRow:forComponent:

Called by the picker view when it needs the title to use for a given row in a given component.

```
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
    forComponent:(NSInteger)component
```

Parameters
pickerView
An object representing the picker view requesting the data.

row
A zero-indexed number identifying a row of *component*. Rows are numbered top-to-bottom.

component
A zero-indexed number identifying a component of *pickerView*. Components are numbered left-to-right.

Return Value
The string to use as the title of the indicated component row.

Availability
Available in iOS 2.0 and later.

Declared In
UIPickerView.h

pickerView:viewForRow:forComponent:reusingView:

Called by the picker view when it needs the view to use for a given row in a given component.

```
- (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
A zero-indexed number identifying a row of *component*. Rows are numbered top-to-bottom.

component
A zero-indexed number identifying a component of *pickerView*. Components are numbered left-to-right.

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 iOS 2.0 and later.

Declared In
`UIPickerView.h`

pickerView:widthForComponent:

Called by the picker view when it needs the row width to use for drawing row content.

```
- (CGFloat)pickerView:(UIPickerView *)pickerView
    widthForComponent:(NSInteger)component
```

Parameters
pickerView
The picker view requesting this information.

component
A zero-indexed number identifying a component of the picker view. Components are numbered left-to-right.

Return Value
A float value indicating the width of the row in points.

Availability
Available in iOS 2.0 and later.

Declared In
`UIPickerView.h`

UIPopoverControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UIPopoverController.h
Related sample code	ToolbarSearch

Overview

The `UIPopoverControllerDelegate` protocol defines the methods you can implement for the delegate of a `UIPopoverController` object. Popover controllers notify their delegate whenever user interactions would cause the dismissal of the popover and, in some cases, give the user a chance to prevent that dismissal.

For more information about the `UIPopoverController` class, see *UIPopoverController Class Reference*.

Tasks

MethodGroup

- `popoverControllerShouldDismissPopover:` (page 884)
  Asks the delegate if the popover should be dismissed.
- `popoverControllerDidDismissPopover:` (page 883)
  Tells the delegate that the popover was dismissed.

Instance Methods

popoverControllerDidDismissPopover:

Tells the delegate that the popover was dismissed.

```
- (void)popoverControllerDidDismissPopover:(UIPopoverController *)popoverController
```

Parameters

popoverController

The popover controller that was dismissed.

Discussion

The popover controller does not call this method in response to programmatic calls to the `dismissPopoverAnimated:` (page 447) method. If you dismiss the popover programmatically, you should perform any cleanup actions immediately after calling the `dismissPopoverAnimated:` method.

You can use this method to incorporate any changes from the popover's content view controller back into your application. If you do not plan to use the object in the *popoverController* parameter again, it is safe to release it from this method.

Availability

Available in iOS 3.2 and later.

Declared In

`UIPopoverController.h`

popoverControllerShouldDismissPopover:

Asks the delegate if the popover should be dismissed.

```
- (BOOL)popoverControllerShouldDismissPopover:(UIPopoverController
    *)popoverController
```

Parameters

popoverController

The popover controller to be dismissed.

Return Value

`YES` if the popover should be dismissed or `NO` if it should remain visible.

Discussion

This method is called in response to user-initiated attempts to dismiss the popover. It is not called when you dismiss the popover using the `dismissPopoverAnimated:` (page 447) method of the popover controller.

If you do not implement this method in your delegate, the default return value is assumed to be `YES`.

Availability

Available in iOS 3.2 and later.

Declared In

`UIPopoverController.h`

UIResponderStandardEditActions Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/UIKit.framework
Declared in	UIResponder.h

Overview

The `UIResponderStandardEditActions` informal protocol declares methods that responder classes should override to handle common editing commands invoked in the user interface, such as Copy, Paste, and Select.

Although this is an informal protocol—that is, a category declared on `NSObject`—it is recommended that responder classes (that is, immediate or distant ancestors of `UIResponder`) override its methods. Starting with the first responder, `UIResponder` looks for a responder object that can handle the method, and works up the responder chain from there. Responder classes may also implement the `canPerformAction:withSender:` (page 461) method of `UIResponder` to disable or enable user-interface commands based on the context. The `copy:` (page 886), `cut:` (page 886), `delete:` (page 887), `paste:` (page 887), `select:` (page 888), and `selectAll:` (page 888) methods are invoked when users tap the corresponding command in the menu managed by the `UIMenuController` shared instance.

Tasks

Handling Copy, Cut, Delete, and Paste Commands

- `copy:` (page 886) *required method*
  Copy the selection to the pasteboard. (required)
- `cut:` (page 886) *required method*
  Remove the selection from the user interface and write it to the pasteboard. (required)
- `delete:` (page 887) *required method*
  Remove the selection from the user interface. (required)
- `paste:` (page 887) *required method*
  Read data from the pasteboard and display it in the user interface. (required)

Handling Selection Commands

- `select:` (page 888) *required method*
  Select the next object the user taps. (required)

- selectAll: (page 888) *required method*
  Select all objects in the current view. (required)

Instance Methods

copy:

Copy the selection to the pasteboard. (required)

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

Parameters

sender
The object calling this method.

Discussion
This method is invoked when the user taps the Copy command of the editing menu. A subclass of `UIResponder` typically implements this method. Using the methods of the `UIPasteboard` class, it should convert the selection into an appropriate object (if necessary) and write that object to a pasteboard. The command travels from the first responder up the responder chain until it is handled; it is ignored if no responder handles it. If a responder doesn't handle the command in the current context, it should pass it to the next responder.

See Also
- cut: (page 886)
- paste: (page 887)

Availability
Available in iOS 3.0 and later.

Declared In
`UIResponder.h`

cut:

Remove the selection from the user interface and write it to the pasteboard. (required)

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

Parameters

sender
The object calling this method.

Discussion
This method is invoked when the user taps the Cut command of the editing menu. A subclass of `UIResponder` typically implements this method. Using the methods of the `UIPasteboard` class, it should convert the selection into an appropriate object (if necessary) and write that object to a pasteboard. It should also remove the selected object from the user interface and, if applicable, from the application's data model. The command travels from the first responder up the responder chain until it is handled; it is ignored if no responder handles it. If a responder doesn't handle the command in the current context, it should pass it to the next responder.

See Also
- `copy:` (page 886)
- `paste:` (page 887)

Availability
Available in iOS 3.0 and later.

Declared In
`UIResponder.h`

delete:

Remove the selection from the user interface. (required)

```
- (void)delete:(id)sender;
```

Parameters

sender
The object calling this method.

Discussion
This method is invoked when the user taps the Delete command of the editing menu. A subclass of `UIResponder` typically implements this method by removing the selected object from the user interface and, if applicable, from the application's data model. It should not write any data to the pasteboard. The command travels from the first responder up the responder chain until it is handled; it is ignored if no responder handles it. If a responder doesn't handle the command in the current context, it should pass it to the next responder.

See Also
- `cut:` (page 886)

Availability
Available in iOS 3.2 and later.

Declared In
`UIResponder.h`

paste:

Read data from the pasteboard and display it in the user interface. (required)

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

Parameters

sender
The object calling this method.

Discussion
This method is invoked when the user taps the Paste command of the editing menu. A subclass of `UIResponder` typically implements this method. Using the methods of the `UIPasteboard` class, it should read the data in the pasteboard, convert the data into an appropriate internal representation (if necessary),

and display it in the user interface. The command travels from the first responder up the responder chain until it is handled; it is ignored if no responder handles it. If a responder doesn't handle the command in the current context, it should pass it to the next responder.

See Also
- `copy:` (page 886)
- `cut:` (page 886)

Availability
Available in iOS 3.0 and later.

Declared In
`UIResponder.h`

select:

Select the next object the user taps. (required)

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

Parameters
sender
The object calling this method.

Discussion
This method is invoked when the user taps the Select command of the editing menu. This command is used for targeted selection of items in the receiving view that can be broken up into chunks. This could be, for example, words in a text view. Another example might be a view that puts lists of visible objects in multiple groups; the `select:` command could be implemented to select all the items in the same group as the currently selected item.

A subclass of `UIResponder` typically implements this method. The command travels from the first responder up the responder chain until it is handled; it is ignored if no responder handles it. If a responder doesn't handle the command in the current context, it should pass it to the next responder.

Availability
Available in iOS 3.0 and later.

See Also
- `selectAll:` (page 888)

Declared In
`UIResponder.h`

selectAll:

Select all objects in the current view. (required)

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

Parameters
sender
The object calling this method.

Discussion
This method is invoked when the user taps the Select All command of the editing menu. A subclass of `UIResponder` typically implements this method by selecting all objects in the current view. The command travels from the first responder up the responder chain until it is handled; it is ignored if no responder handles it. If a responder doesn't handle the command in the current context, it should pass it to the next responder.

See Also
`- select:` (page 888)

Availability
Available in iOS 3.0 and later.

Declared In
`UIResponder.h`

UIScrollViewDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UIScrollView.h
Related sample code	ScrollViewSuite

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 894)
  Tells the delegate when the user scrolls the content view within the receiver.
- `scrollViewWillBeginDragging:` (page 896)
  Tells the delegate when the scroll view is about to start scrolling the content.
- `scrollViewDidEndDragging:willDecelerate:` (page 892)
  Tells the delegate when dragging ended in the scroll view.
- `scrollViewShouldScrollToTop:` (page 895)
  Asks the delegate if the scroll view should scroll to the top of the content.
- `scrollViewDidScrollToTop:` (page 894)
  Tells the delegate that the scroll view scrolled to the top of the content.
- `scrollViewWillBeginDecelerating:` (page 896)
  Tells the delegate that the scroll view is starting to decelerate the scrolling movement.
- `scrollViewDidEndDecelerating:` (page 892)
  Tells the delegate that the scroll view has ended decelerating the scrolling movement.

Managing Zooming

- `viewForZoomingInScrollView:` (page 897)
  Asks the delegate for the view to scale when zooming is about to occur in the scroll view.
- `scrollViewWillBeginZooming:withView:` (page 897)
  Tells the delegate that zooming of the content in the scroll view is about to commence.
- `scrollViewDidEndZooming:withView:atScale:` (page 893)
  Tells the delegate when zooming of the content in the scroll view completed.
- `scrollViewDidZoom:` (page 895)
  Tells the delegate that the scroll view's zoom factor changed.

Responding to Scrolling Animations

- `scrollViewDidEndScrollingAnimation:` (page 893)
  Tells the delegate when a scrolling animation in the scroll view concludes.

Instance Methods

scrollViewDidEndDecelerating:

Tells the delegate that the scroll view has ended decelerating the scrolling movement.

```
- (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 487) property of `UIScrollView` controls deceleration.

Availability
Available in iOS 2.0 and later.

See Also
- `scrollViewWillBeginDecelerating:` (page 896)

Declared In
`UIScrollView.h`

scrollViewDidEndDragging:willDecelerate:

Tells the delegate when dragging ended in the scroll view.

```
- (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 487) property of UIScrollView controls deceleration.

Availability

Available in iOS 2.0 and later.

See Also

- scrollViewDidScroll: (page 894)
- scrollViewWillBeginDragging: (page 896)
- scrollViewWillBeginDecelerating: (page 896)

Declared In

UIScrollView.h

scrollViewDidEndScrollingAnimation:

Tells the delegate when a scrolling animation in the scroll view concludes.

```
- (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 495) and scrollRectToVisible:animated: (page 494) methods, but only if animations are requested.

Availability

Available in iOS 2.0 and later.

Declared In

UIScrollView.h

scrollViewDidEndZooming:withView:atScale:

Tells the delegate when zooming of the content in the scroll view completed.

```
- (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 490) and `minimumZoomScale` (page 490).

Discussion

The scroll view also calls this method after any "bounce" animations. It also calls this method after animated changes to the zoom level and after a zoom-related gesture ends (regardless of whether the gesture resulted in a change to the zoom level).

Availability

Available in iOS 2.0 and later.

See Also

`- viewForZoomingInScrollView:` (page 897)

Declared In

`UIScrollView.h`

scrollViewDidScroll:

Tells the delegate when the user scrolls the content view within the receiver.

```
- (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 iOS 2.0 and later.

See Also

`- scrollViewWillBeginDragging:` (page 896)

`- scrollViewDidEndDragging:willDecelerate:` (page 892)

Declared In

`UIScrollView.h`

scrollViewDidScrollToTop:

Tells the delegate that the scroll view scrolled to the top of the content.

```
- (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 491) property of the `UIScrollView` must be set to YES.

Availability

Available in iOS 2.0 and later.

See Also

- `scrollViewShouldScrollToTop:` (page 895)
- `scrollViewDidScroll:` (page 894)

Declared In

`UIScrollView.h`

scrollViewDidZoom:

Tells the delegate that the scroll view's zoom factor changed.

```
- (void)scrollViewDidZoom:(UIScrollView *)scrollView
```

Parameters

scrollView

The scroll-view object whose zoom factor changed.

Availability

Available in iOS 3.2 and later.

Declared In

`UIScrollView.h`

scrollViewShouldScrollToTop:

Asks the delegate if the scroll view should scroll to the top of the content.

```
- (BOOL)scrollViewShouldScrollToTop:(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 491) property of the `UIScrollView` must be set to YES.

Availability

Available in iOS 2.0 and later.

See Also
- `scrollViewDidScrollToTop:` (page 894)
- `scrollViewDidScroll:` (page 894)

Declared In
`UIScrollView.h`

scrollViewWillBeginDecelerating:

Tells the delegate that the scroll view is starting to decelerate the scrolling movement.

```
- (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 487) property of `UIScrollView` controls deceleration.

Availability
Available in iOS 2.0 and later.

See Also
- `scrollViewDidEndDecelerating:` (page 892)

Declared In
`UIScrollView.h`

scrollViewWillBeginDragging:

Tells the delegate when the scroll view is about to start scrolling the content.

```
- (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 iOS 2.0 and later.

See Also
- `scrollViewDidScroll:` (page 894)
- `scrollViewDidEndDragging:willDecelerate:` (page 892)

Declared In
`UIScrollView.h`

scrollViewWillBeginZooming:withView:

Tells the delegate that zooming of the content in the scroll view is about to commence.

```
- (void)scrollViewWillBeginZooming:(UIScrollView *)scrollView withView:(UIView
    *)view
```

Parameters

scrollView

The scroll-view object displaying the content view.

view

The view object whose content is about to be zoomed.

Discussion

This method is called at the beginning of zoom gestures and in cases where a change in zoom level is to be animated. You can use this method to store state information or perform any additional actions prior to zooming the view's content.

Availability

Available in iOS 3.2 and later.

Declared In

`UIScrollView.h`

viewForZoomingInScrollView:

Asks the delegate for the view to scale when zooming is about to occur in the scroll view.

```
- (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 iOS 2.0 and later.

See Also

`- scrollViewDidEndZooming:withView:atScale:` (page 893)

Declared In

`UIScrollView.h`

UISearchBarDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UISearchBar.h
Related sample code	ToolbarSearch

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 901)
  Tells the delegate that the user changed the search text.
- `searchBar:shouldChangeTextInRange:replacementText:` (page 900)
  Ask the delegate if text in a specified range should be replaced with given text.
- `searchBarShouldBeginEditing:` (page 903)
  Asks the delegate if editing should begin in the specified search bar.
- `searchBarTextDidBeginEditing:` (page 904)
  Tells the delegate when the user begins editing the search text.
- `searchBarShouldEndEditing:` (page 903)
  Asks the delegate if editing should stop in the specified search bar.
- `searchBarTextDidEndEditing:` (page 904)
  Tells the delegate that the user finished editing the search text.

Clicking Buttons

- `searchBarBookmarkButtonClicked:` (page 901)
  Tells the delegate that the bookmark button was tapped.
- `searchBarCancelButtonClicked:` (page 902)
  Tells the delegate that the cancel button was tapped.
- `searchBarSearchButtonClicked:` (page 902)
  Tells the delegate that the search button was tapped.
- `searchBarResultsListButtonClicked:` (page 902)
  Tells the delegate that the search results list button was tapped.

Scope Button

- `searchBar:selectedScopeButtonIndexDidChange:` (page 900)
  Tells the delegate that the scope button selection changed.

Instance Methods

searchBar:selectedScopeButtonIndexDidChange:

Tells the delegate that the scope button selection changed.

```
- (void)searchBar:(UISearchBar *)searchBar
    selectedScopeButtonIndexDidChange:(NSInteger)selectedScope
```

Parameters

searchBar
  The search bar that was tapped.

selectedScope
  The index of the selected scope button (see `selectedScopeButtonIndex`).

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchBar.h`

searchBar:shouldChangeTextInRange:replacementText:

Ask the delegate if text in a specified range should be replaced with given text.

```
- (BOOL)searchBar:(UISearchBar *)searchBar
    shouldChangeTextInRange:(NSRange)range
    replacementText:(NSString *)text
```

Parameters

searchBar

The search bar that is being edited.

range

The range of the text to be changed.

text

The text to replace existing text in *range*.

Return Value

YES if text in *range* should be replaced by *text*, otherwise, NO.

Availability

Available in iOS 3.0 and later.

Declared In

UISearchBar.h

searchBar:textDidChange:

Tells the delegate that 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 iOS 2.0 and later.

Declared In

UISearchBar.h

searchBarBookmarkButtonClicked:

Tells the delegate that the bookmark button was 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 iOS 2.0 and later.

See Also
`showsBookmarkButton` (page 504)

Declared In
`UISearchBar.h`

searchBarCancelButtonClicked:

Tells the delegate that the cancel button was 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 iOS 2.0 and later.

See Also
`showsCancelButton` (page 504)

Declared In
`UISearchBar.h`

searchBarResultsListButtonClicked:

Tells the delegate that the search results list button was tapped.

```
- (void)searchBarResultsListButtonClicked:(UISearchBar *)searchBar
```

Parameters

searchBar
The search bar that was tapped.

Availability
Available in iOS 3.2 and later.

Declared In
`UISearchBar.h`

searchBarSearchButtonClicked:

Tells the delegate that the search button was 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 505) property of the search bar to get the text. You can also send `becomeFirstResponder` (page 460) to the search bar to begin editing programmatically.

Availability

Available in iOS 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`.

Availability

Available in iOS 2.0 and later.

See Also

- `searchBarTextDidBeginEditing:` (page 904)
- `searchBarShouldEndEditing:` (page 903)
- `searchBarTextDidEndEditing:` (page 904)

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`.

Availability

Available in iOS 2.0 and later.

See Also
- `searchBarShouldBeginEditing:` (page 903)
- `searchBarTextDidBeginEditing:` (page 904)
- `searchBarTextDidEndEditing:` (page 904)

Declared In
`UISearchBar.h`

searchBarTextDidBeginEditing:

Tells 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 iOS 2.0 and later.

See Also
- `searchBarShouldBeginEditing:` (page 903)
- `searchBarShouldEndEditing:` (page 903)
- `searchBarTextDidEndEditing:` (page 904)

Declared In
`UISearchBar.h`

searchBarTextDidEndEditing:

Tells the delegate that the user finished 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 iOS 2.0 and later.

See Also
- `searchBarShouldBeginEditing:` (page 903)
- `searchBarTextDidBeginEditing:` (page 904)
- `searchBarShouldEndEditing:` (page 903)

Declared In
`UISearchBar.h`

UISearchDisplayDelegate Protocol Reference

Conforms to	NSObject
Availability	Available in iOS 3.0 and later.
Declared in	UIKit/UISearchDisplayDelegate.h

Overview

This protocol defines delegate methods for `UISearchDisplayController` objects.

Tasks

Search State Change

- `searchDisplayControllerWillBeginSearch:` (page 910)
  Tells the delegate that the controller is about to begin searching.
- `searchDisplayControllerDidBeginSearch:` (page 909)
  Tells the delegate that the controller has started searching.
- `searchDisplayControllerWillEndSearch:` (page 910)
  Tells the delegate that the controller is about to end searching.
- `searchDisplayControllerDidEndSearch:` (page 910)
  Tells the delegate that the controller has finished searching.

Loading and Unloading the Table View

- `searchDisplayController:didLoadSearchResultsTableView:` (page 906)
  Tells the delegate that the controller has loaded its table view.
- `searchDisplayController:willUnloadSearchResultsTableView:` (page 909)
  Tells the delegate that the controller is about to unload its table view.

Showing and Hiding the Table View

- `searchDisplayController:willShowSearchResultsTableView:` (page 909)
  Tells the delegate that the controller is about to display its table view.
- `searchDisplayController:didShowSearchResultsTableView:` (page 907)
  Tells the delegate that the controller just displayed its table view.
- `searchDisplayController:willHideSearchResultsTableView:` (page 908)
  Tells the delegate that the controller is about to hide its table view.
- `searchDisplayController:didHideSearchResultsTableView:` (page 906)
  Tells the delegate that the controller just hid its table view.

Responding to Changes in Search Criteria

- `searchDisplayController:shouldReloadTableForSearchString:` (page 908)
  Asks the delegate if the table view should be reloaded for a given search string.
- `searchDisplayController:shouldReloadTableForSearchScope:` (page 907)
  Asks the delegate if the table view should be reloaded for a given scope.

Instance Methods

searchDisplayController:didHideSearchResultsTableView:

Tells the delegate that the controller just hid its table view.

```
- (void)searchDisplayController:(UISearchDisplayController *)controller
    didHideSearchResultsTableView:(UITableView *)tableView
```

Parameters

controller
The search display controller for which the receiver is the delegate.

tableView
The search display controller's table view.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchDisplayController:didLoadSearchResultsTableView:

Tells the delegate that the controller has loaded its table view.

```
- (void)searchDisplayController:(UISearchDisplayController *)controller
    didLoadSearchResultsTableView:(UITableView *)tableView
```

Parameters

controller

The search display controller for which the receiver is the delegate.

tableView

The search display controller's table view.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchDisplayController:didShowSearchResultsTableView:

Tells the delegate that the controller just displayed its table view.

```
- (void)searchDisplayController:(UISearchDisplayController *)controller
    didShowSearchResultsTableView:(UITableView *)tableView
```

Parameters

controller

The search display controller for which the receiver is the delegate.

tableView

The search display controller's table view.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchDisplayController:shouldReloadTableForSearchScope:

Asks the delegate if the table view should be reloaded for a given scope.

```
- (BOOL)searchDisplayController:(UISearchDisplayController *)controller
    shouldReloadTableForSearchScope:(NSInteger)searchOption
```

Parameters

controller

The search display controller for which the receiver is the delegate.

searchOption

The index of the selected scope button in the search bar.

Return Value
YES if the display controller should reload the data in its table view, otherwise NO.

Discussion
If you don't implement this method, then the results table is reloaded as soon as the scope button selection changes.

You might implement this method if you want to perform an asynchronous search: you would initiate the search in this method, then return NO, and reload the table when you have results.

Availability
Available in iOS 3.0 and later.

See Also
- `searchDisplayController:shouldReloadTableForSearchString:` (page 908)
  `@property selectedScopeButtonIndex` (`UISearchBar`)

Declared In
`UISearchDisplayController.h`

searchDisplayController:shouldReloadTableForSearchString:

Asks the delegate if the table view should be reloaded for a given search string.

```
- (BOOL)searchDisplayController:(UISearchDisplayController *)controller
    shouldReloadTableForSearchString:(NSString *)searchString
```

Parameters

controller
The search display controller for which the receiver is the delegate.

searchString
The string in the search bar.

Return Value
`YES` if the display controller should reload the data in its table view, otherwise `NO`.

Discussion
If you don't implement this method, then the results table is reloaded as soon as the search string changes.

You might implement this method if you want to perform an asynchronous search. You would initiate the search in this method, then return `NO`. You would reload the table when you have results.

Availability
Available in iOS 3.0 and later.

See Also
- `searchDisplayController:shouldReloadTableForSearchScope:` (page 907)

Declared In
`UISearchDisplayController.h`

searchDisplayController:willHideSearchResultsTableView:

Tells the delegate that the controller is about to hide its table view.

```
- (void)searchDisplayController:(UISearchDisplayController *)controller
    willHideSearchResultsTableView:(UITableView *)tableView
```

Parameters

controller
The search display controller for which the receiver is the delegate.

tableView
The search display controller's table view.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchDisplayController:willShowSearchResultsTableView:

Tells the delegate that the controller is about to display its table view.

```
- (void)searchDisplayController:(UISearchDisplayController *)controller
    willShowSearchResultsTableView:(UITableView *)tableView
```

Parameters
controller
The search display controller for which the receiver is the delegate.

tableView
The search display controller's table view.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchDisplayController:willUnloadSearchResultsTableView:

Tells the delegate that the controller is about to unload its table view.

```
- (void)searchDisplayController:(UISearchDisplayController *)controller
    willUnloadSearchResultsTableView:(UITableView *)tableView
```

Parameters
controller
The search display controller for which the receiver is the delegate.

tableView
The search display controller's table view.

Availability
Available in iOS 3.0 and later.

Declared In
UISearchDisplayController.h

searchDisplayControllerDidBeginSearch:

Tells the delegate that the controller has started searching.

```
- (void)searchDisplayControllerDidBeginSearch:(UISearchDisplayController *)controller
```

Parameters

controller

The search display controller for which the receiver is the delegate.

Availability

Available in iOS 3.0 and later.

Declared In

UISearchDisplayController.h

searchDisplayControllerDidEndSearch:

Tells the delegate that the controller has finished searching.

```
- (void)searchDisplayControllerDidEndSearch:(UISearchDisplayController *)controller
```

Parameters

controller

The search display controller for which the receiver is the delegate.

Availability

Available in iOS 3.0 and later.

Declared In

UISearchDisplayController.h

searchDisplayControllerWillBeginSearch:

Tells the delegate that the controller is about to begin searching.

```
- (void)searchDisplayControllerWillBeginSearch:(UISearchDisplayController
    *)controller
```

Parameters

controller

The search display controller for which the receiver is the delegate.

Availability

Available in iOS 3.0 and later.

Declared In

UISearchDisplayController.h

searchDisplayControllerWillEndSearch:

Tells the delegate that the controller is about to end searching.

```
- (void)searchDisplayControllerWillEndSearch:(UISearchDisplayController *)controller
```

Parameters

controller

The search display controller for which the receiver is the delegate.

Availability
Available in iOS 3.0 and later.

Declared In
`UISearchDisplayController.h`

UISplitViewControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UISplitViewController.h
Related sample code	MultipleDetailViews

Overview

The `UISplitViewControllerDelegate` protocol defines methods that allow you to manage changes to the visible view controllers in a split view controller. When the split view controller rotates between portrait and landscape orientations, it hides or shows the first view controller in its array of view controllers. When the view controller is hidden, it is standard practice to add a button to the toolbar of the remaining view controller that, when tapped, displays the hidden view controller in a popover. The methods of this protocol provide you with the information you need to add and remove this button at the appropriate times.

For more information about the `UISplitViewController` class, see *UISplitViewController Class Reference*.

Tasks

Showing and Hiding View Controllers

`- splitViewController:willHideViewController:withBarButtonItem:forPopoverController:` (page 914)
Tells the delegate that the specified view controller is about to be hidden.

`- splitViewController:willShowViewController:invalidatingBarButtonItem:` (page 915)
Tells the delegate that the specified view controller is about to be shown again.

`- splitViewController:popoverController:willPresentViewController:` (page 914)
Tells the delegate that the hidden view controller is about to be displayed in a popover.

Instance Methods

splitViewController:popoverController:willPresentViewController:

Tells the delegate that the hidden view controller is about to be displayed in a popover.

```
- (void)splitViewController:(UISplitViewController*)svc
    popoverController:(UIPopoverController*)pc
    willPresentViewController:(UIViewController *)aViewController
```

Parameters

svc

The split view controller that owns the hidden view controller.

pc

The popover controller that is about to display the view controller.

aViewController

The view controller to be displayed in the popover.

Discussion

The toolbar button you add to your user interface facilitates the display of the hidden view controller in response to user taps. When the user taps that button, the split view controller calls this method. You can use this method to perform any additional steps prior to displaying the currently hidden view controller.

Availability

Available in iOS 3.2 and later.

Declared In

`UISplitViewController.h`

splitViewController:willHideViewController:withBarButtonItem:forPopoverController:

Tells the delegate that the specified view controller is about to be hidden.

```
- (void)splitViewController:(UISplitViewController*)svc
    willHideViewController:(UIViewController *)aViewController
    withBarButtonItem:(UIBarButtonItem*)barButtonItem
    forPopoverController:(UIPopoverController*)pc
```

Parameters

svc

The split view controller that owns the specified view controller.

aViewController

The view controller being hidden.

barButtonItem

A button you can add to your toolbar.

pc

The popover controller that uses taps in *barButtonItem* to display the specified view controller.

Discussion
When the split view controller rotates from a landscape to portrait orientation, it normally hides one of its view controllers. When that happens, it calls this method to coordinate the addition of a button to the toolbar (or navigation bar) of the remaining custom view controller. If you want the soon-to-be hidden view controller to be displayed in a popover, you must implement this method and use it to add the specified button to your interface.

Availability
Available in iOS 3.2 and later.

Declared In
`UISplitViewController.h`

splitViewController:willShowViewController:invalidatingBarButtonItem:

Tells the delegate that the specified view controller is about to be shown again.

```
- (void)splitViewController:(UISplitViewController*)svc
    willShowViewController:(UIViewController *)aViewController
    invalidatingBarButtonItem:(UIBarButtonItem *)button
```

Parameters

svc
The split view controller that owns the specified view controller.

aViewController
The view controller being hidden.

button
The button used to display the view controller while it was hidden.

Discussion
When the view controller rotates from a portrait to landscape orientation, it shows its hidden view controller once more. If you added the specified button to your toolbar to facilitate the display of the hidden view controller in a popover, you must implement this method and use it to remove that button.

Availability
Available in iOS 3.2 and later.

Declared In
`UISplitViewController.h`

UITabBarControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITabBarController.h
Companion guide	View Controller Programming Guide for iOS
Related sample code	MoviePlayer

Overview

You use the `UITabBarControllerDelegate` protocol when you want to augment the behavior of a tab bar. In particular, you can use it to determine whether specific tabs should be selected, to perform actions after a tab is selected, or to perform actions before or after the user customizes the order of the tabs. After implementing these methods in your custom object, you should then assign that object to the `delegate` (page 563) property of the corresponding `UITabBarController` object.

All of the methods in this protocol are optional. For more information on how to use and configure tab bar controllers and their delegates, see *View Controller Programming Guide for iOS*.

Tasks

Managing Tab Bar Selections

- `tabBarController:shouldSelectViewController:` (page 919)
  Asks the delegate whether the specified view controller should be made active.
- `tabBarController:didSelectViewController:` (page 918)
  Tells the delegate that the user selected an item in the tab bar.

Managing Tab Bar Customizations

- `tabBarController:willBeginCustomizingViewControllers:` (page 919)
  Tells the delegate that the tab bar customization sheet is about to be displayed.

- `tabBarController:willEndCustomizingViewControllers:changed:` (page 920)
  Tells the delegate that the tab bar customization sheet is about to be dismissed.
- `tabBarController:didEndCustomizingViewControllers:changed:` (page 918)
  Tells the delegate that the tab bar customization sheet was dismissed.

Instance Methods

tabBarController:didEndCustomizingViewControllers:changed:

Tells the delegate that the tab bar customization sheet was 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 of the tab bar controller. The arrangement of the controllers in the array represents the new display order within the tab bar.

changed
  A Boolean value indicating whether items changed on the tab bar. `YES` if items changed or `NO` if they did not.

Discussion
You can use this method to respond to changes to the order of tabs in the tab bar.

Availability
Available in iOS 2.0 and later.

Declared In
`UITabBarController.h`

tabBarController:didSelectViewController:

Tells the delegate that the user selected an item in the tab bar.

```
- (void)tabBarController:(UITabBarController *)tabBarController
    didSelectViewController:(UIViewController *)viewController
```

Parameters

tabBarController
  The tab bar controller containing *viewController*.

viewController
  The view controller that the user selected. In iOS v3.0 and later, this could be the same view controller that was already selected.

Discussion
In iOS v3.0 and later, the tab bar controller calls this method regardless of whether the selected view controller changed. In addition, it is called only in response to user taps in the tab bar and is not called when your code changes the tab bar contents programmatically.

In versions of iOS prior to version 3.0, this method is called only when the selected view controller actually changes. In other words, it is not called when the same view controller is selected. In addition, the method was called for both programmatic and user-initiated changes to the selected view controller.

Availability
Available in iOS 2.0 and later.

Declared In
`UITabBarController.h`

tabBarController:shouldSelectViewController:

Asks the delegate whether the specified view controller should be made active.

```
- (BOOL)tabBarController:(UITabBarController *)tabBarController
    shouldSelectViewController:(UIViewController *)viewController
```

Parameters

tabBarController
The tab bar controller containing *viewController*.

viewController
The view controller belonging to the tab that was tapped by the user.

Return Value
`YES` if the view controller's tab should be selected or `NO` if the current tab should remain active.

Discussion
The tab bar controller calls this method in response to the user tapping a tab bar item. You can use this method to dynamically decide whether a given tab should be made the active tab.

Availability
Available in iOS 3.0 and later.

Declared In
`UITabBarController.h`

tabBarController:willBeginCustomizingViewControllers:

Tells the delegate that the tab bar customization sheet is about to be displayed.

```
- (void)tabBarController:(UITabBarController *)tabBarController
    willBeginCustomizingViewControllers:(NSArray *)viewControllers
```

Parameters

tabBarController
The tab bar controller that is being customized.

viewControllers

The view controllers to be displayed in the customization sheet. This list typically contains all custom view controllers you added but does not include some standard controllers, such as the one that manages the More tab.

Availability
Available in iOS 3.0 and later.

Declared In
UITabBarController.h

tabBarController:willEndCustomizingViewControllers:changed:

Tells the delegate that the tab bar customization sheet is about to be dismissed.

```
- (void)tabBarController:(UITabBarController *)tabBarController
    willEndCustomizingViewControllers:(NSArray *)viewControllers
    changed:(BOOL)changed
```

Parameters

tabBarController

The tab bar controller that is being customized.

viewControllers

The view controllers of the tab bar controller. The arrangement of the controllers in the array represents the new display order within the tab bar.

changed

A Boolean value indicating whether items changed on the tab bar. YES if items changed or NO if they did not.

Discussion
This method is called in response to the user tapping the Done button on the sheet but before the sheet is dismissed.

Availability
Available in iOS 3.0 and later.

Declared In
UITabBarController.h

UITabBarDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITabBar.h

Overview

The `UITabBarDelegate` protocol defines optional methods for a delegate of a `UITabBar` object. The `UITabBar` class provides the ability for the user to reorder, remove, and add items to the tab bar; this process is referred to as customizing the tab bar. The tab bar delegate receives messages when customizing occurs.

Send `beginCustomizingItems:` (page 555) to a `UITabBar` object to begin customizing. Implement the methods in "Customizing Tab Bars" (page 921) to intervene while a user is customizing a tab bar. The customizing modal view is dismissed when the user taps the Done button on the modal view.

Tasks

Customizing Tab Bars

- `tabBar:willBeginCustomizingItems:` (page 923)
  Sent to the delegate before the customizing modal view is displayed.
- `tabBar:didBeginCustomizingItems:` (page 922)
  Sent to the delegate after the customizing modal view is displayed.
- `tabBar:willEndCustomizingItems:changed:` (page 923)
  Sent to the delegate before the customizing modal view is dismissed.
- `tabBar:didEndCustomizingItems:changed:` (page 922)
  Sent to the delegate after the customizing modal view is dismissed.
- `tabBar:didSelectItem:` (page 923) *required method*
  Sent to the delegate when the user selects a tab bar item. (required)

Instance Methods

tabBar:didBeginCustomizingItems:

Sent to the delegate after the customizing modal view is displayed.

```
- (void)tabBar:(UITabBar *)tabBar didBeginCustomizingItems:(NSArray *)items
```

Parameters

tabBar

The tab bar that is being customized.

items

The items on the customizing modal view.

Availability
Available in iOS 2.0 and later.

See Also
- `tabBar:willBeginCustomizingItems:` (page 923)
- `tabBar:willEndCustomizingItems:changed:` (page 923)
- `tabBar:didEndCustomizingItems:changed:` (page 922)

Declared In
`UITabBar.h`

tabBar:didEndCustomizingItems:changed:

Sent to the delegate after the customizing modal view is dismissed.

```
- (void)tabBar:(UITabBar *)tabBar didEndCustomizingItems:(NSArray *)items
    changed:(BOOL)changed
```

Parameters

tabBar

The tab bar that is being customized.

items

The items on the customizing modal view.

changed

`YES` if the visible set of items on the tab bar changed; otherwise, `NO`.

Availability
Available in iOS 2.0 and later.

See Also
- `tabBar:willBeginCustomizingItems:` (page 923)
- `tabBar:didBeginCustomizingItems:` (page 922)
- `tabBar:willEndCustomizingItems:changed:` (page 923)

Declared In
`UITabBar.h`

tabBar:didSelectItem:

Sent to the delegate when the user selects a tab bar item. (required)

```
- (void)tabBar:(UITabBar *)tabBar didSelectItem:(UITabBarItem *)item
```

Parameters

tabBar

The tab bar that is being customized.

item

The tab bar item that was selected.

Availability
Available in iOS 2.0 and later.

Declared In
UITabBar.h

tabBar:willBeginCustomizingItems:

Sent to the delegate before the customizing modal view is displayed.

```
- (void)tabBar:(UITabBar *)tabBar willBeginCustomizingItems:(NSArray *)items
```

Parameters

tabBar

The tab bar that is being customized.

items

The items on the customizing modal view.

Discussion
Use the `beginCustomizingItems:` (page 555) method of `UITabBar` to display the customizing modal view and begin the customizing mode.

Availability
Available in iOS 2.0 and later.

See Also
- `tabBar:didBeginCustomizingItems:` (page 922)
- `tabBar:willEndCustomizingItems:changed:` (page 923)
- `tabBar:didEndCustomizingItems:changed:` (page 922)

Declared In
UITabBar.h

tabBar:willEndCustomizingItems:changed:

Sent to the delegate before the customizing modal view is dismissed.

```
- (void)tabBar:(UITabBar *)tabBar willEndCustomizingItems:(NSArray *)items
    changed:(BOOL)changed
```

Parameters

tabBar

The tab bar that is being customized.

items

The items on the customizing modal view.

changed

`YES` if the visible set of items on the tab bar changed; otherwise, `NO`.

Availability

Available in iOS 2.0 and later.

See Also

- `tabBar:willBeginCustomizingItems:` (page 923)
- `tabBar:didBeginCustomizingItems:` (page 922)
- `tabBar:didEndCustomizingItems:changed:` (page 922)

Declared In

`UITabBar.h`

UITableViewDataSource Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITableView.h
Companion guide	Table View Programming Guide for iOS
Related sample code	GKRocket

Overview

The `UITableViewDataSource` protocol is adopted by an 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 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.

Note: To enable the swipe-to-delete feature of table views (wherein a user swipes horizontally across a row to display a Delete button), you must implement the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) method.

Many methods take `NSIndexPath` objects as parameters. `UITableView` declares a category on `NSIndexPath` that enables you to get the represented row index (`row` (page 42) property) and section index (`section` (page 42) property), and to construct an index path from a given row index and section index (`indexPathForRow:inSection:` (page 42) 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 928) *required method*
  Asks the data source for a cell to insert in a particular location of the table view. (required)
- `numberOfSectionsInTableView:` (page 926)
  Asks the data source to return the number of sections in the table view.
- `tableView:numberOfRowsInSection:` (page 930) *required method*
  Tells the data source to return the number of rows in a given section of a table view. (required)
- `sectionIndexTitlesForTableView:` (page 927)
  Asks the data source to return the titles for the sections for a table view.
- `tableView:sectionForSectionIndexTitle:atIndex:` (page 931)
  Asks the data source to return the index of the section having the given title and section title index.
- `tableView:titleForHeaderInSection:` (page 932)
  Asks the data source for the title of the header of the specified section of the table view.
- `tableView:titleForFooterInSection:` (page 931)
  Asks the data source for the title of the footer of the specified section of the table view.

Inserting or Deleting Table Rows

- `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929)
  Asks the data source to commit the insertion or deletion of a specified row in the receiver.
- `tableView:canEditRowAtIndexPath:` (page 927)
  Asks the data source to verify that the given row is editable.

Reordering Table Rows

- `tableView:canMoveRowAtIndexPath:` (page 928)
  Asks the data source whether a given row can be moved to another location in the table view.
- `tableView:moveRowAtIndexPath:toIndexPath:` (page 930)
  Tells the data source to move a row at a specific location in the table view to another location.

Instance Methods

numberOfSectionsInTableView:

Asks the data source to return the number of sections in the table view.

```
- (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.

Availability

Available in iOS 2.0 and later.

See Also

- tableView:numberOfRowsInSection: (page 930)

Declared In

UITableView.h

sectionIndexTitlesForTableView:

Asks the data source to return the titles for the sections for a table view.

```
- (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 iOS 2.0 and later.

See Also

- tableView:sectionForSectionIndexTitle:atIndex: (page 931)

Declared In

UITableView.h

tableView:canEditRowAtIndexPath:

Asks the data source to verify that the given row is editable.

```
- (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 615) 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 631) from the `tableView:editingStyleForRowAtIndexPath:` (page 939) delegate method.

Availability
Available in iOS 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.

```
- (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 930) method.

Availability
Available in iOS 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. (required)

```
- (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 587) message to *tableView*. The identifier for a reusable cell object is assigned when the delegate initializes the cell object by calling the `initWithStyle:reuseIdentifier:` (page 625) 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 iOS 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.

```
- (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 631) or `UITableViewCellEditingStyleDelete` (page 631).

indexPath

An index path locating the row in *tableView*.

Discussion

When users tap the insertion (green plus) control or Delete button associated with a `UITableViewCell` object in the table view, the table view sends this message to the data source, asking it to commit the change. (If the user taps the deletion (red minus) control, the table view then displays the Delete button to get confirmation.) The data source commits the insertion or deletion by invoking the `UITableView` methods `insertRowsAtIndexPaths:withRowAnimation:` (page 591) or `deleteRowsAtIndexPaths:withRowAnimation:` (page 586), as appropriate.

To enable the swipe-to-delete feature of table views (wherein a user swipes horizontally across a row to display a Delete button), you must implement this method.

You should not call `setEditing:animated:` within an implementation of this method. If for some reason you must, invoke it after a delay by using the `performSelector:withObject:afterDelay:` method.

Availability
Available in iOS 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.

```
- (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 iOS 2.0 and later.

See Also
- tableView:commitEditingStyle:forRowAtIndexPath: (page 929)

Declared In
`UITableView.h`

tableView:numberOfRowsInSection:

Tells the data source to return the number of rows in a given section of a table view. (required)

```
- (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 iOS 2.0 and later.

See Also
- `numberOfSectionsInTableView:` (page 926)

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.

```
- (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 927).

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 600)). 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 iOS 2.0 and later.

See Also
- `numberOfSectionsInTableView:` (page 926)

Declared In
`UITableView.h`

tableView:titleForFooterInSection:

Asks the data source for the title of the footer of the specified section of the table view.

```
- (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 944) instead.

Availability

Available in iOS 2.0 and later.

See Also

`- tableView:titleForHeaderInSection:` (page 932)

Declared In

`UITableView.h`

tableView:titleForHeaderInSection:

Asks the data source for the title of the header of the specified section of the table view.

```
- (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 944) instead.

Availability

Available in iOS 2.0 and later.

See Also

`- tableView:titleForFooterInSection:` (page 931)

Declared In
`UITableView.h`

UITableViewDelegate Protocol Reference

Conforms to	NSObject UIScrollViewDelegate
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITableView.h
Companion guide	Table View Programming Guide for iOS
Related sample code	AddMusic GKRocket

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 42) property) and section index (`section` (page 42) property), and to construct an index path from a given row index and section index (`indexPathForRow:inSection:` (page 42) 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 941)
  Asks the delegate for the height to use for a row in a specified location.
- `tableView:indentationLevelForRowAtIndexPath:` (page 942)
  Asks the delegate to return the level of indentation for a row in a given section.
- `tableView:willDisplayCell:forRowAtIndexPath:` (page 946)
  Tells the delegate the table view is about to draw a cell for a particular row.

Managing Accessory Views

- `tableView:accessoryTypeForRowWithIndexPath:` (page 937)
  Asks the delegate for the type of standard accessory view to use as a disclosure control for the specified row. (Deprecated. Use the accessory-view and accessory-type properties (for both normal and editing modes) of the `UITableViewCell` class when configuring table-view cells.)
- `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937)
  Tells the delegate that the user tapped the accessory (disclosure) view associated with a given row.

Managing Selections

- `tableView:willSelectRowAtIndexPath:` (page 947)
  Tells the delegate that a specified row is about to be selected.
- `tableView:didSelectRowAtIndexPath:` (page 939)
  Tells the delegate that the specified row is now selected.
- `tableView:willDeselectRowAtIndexPath:` (page 946)
  Tells the delegate that a specified row is about to be deselected.
- `tableView:didDeselectRowAtIndexPath:` (page 938)
  Tells the delegate that the specified row is now deselected.

Modifying the Header and Footer of Sections

- `tableView:viewForHeaderInSection:` (page 944)
  Asks the delegate for a view object to display in the header of the specified section of the table view.
- `tableView:viewForFooterInSection:` (page 944)
  Asks the delegate for a view object to display in the footer of the specified section of the table view.
- `tableView:heightForHeaderInSection:` (page 941)
  Asks the delegate for the height to use for the header of a particular section.
- `tableView:heightForFooterInSection:` (page 940)
  Asks the delegate for the height to use for the footer of a particular section.

Editing Table Rows

- `tableView:willBeginEditingRowAtIndexPath:` (page 945)
  Tells the delegate that the table view is about to go into editing mode.
- `tableView:didEndEditingRowAtIndexPath:` (page 938)
  Tells the delegate that the table view has left editing mode.
- `tableView:editingStyleForRowAtIndexPath:` (page 939)
  Asks the delegate for the editing style of a row at a particular location in a table view.
- `tableView:titleForDeleteConfirmationButtonForRowAtIndexPath:` (page 943)
  Changes the default title of the delete-confirmation button.
- `tableView:shouldIndentWhileEditingRowAtIndexPath:` (page 942)
  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 943)
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.

```
- (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 tap on the disclosure button (the accessory view) by displaying a new view related to the selected row. This method is not called when an accessory view is set for the row at *indexPath*.

Availability
Available in iOS 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. (Deprecated in iOS 3.0. Use the accessory-view and accessory-type properties (for both normal and editing modes) of the `UITableViewCell` class when configuring table-view cells.)

```
- (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 *UITableViewCell Class Reference*.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 3.0.

See Also
- `tableView:accessoryButtonTappedForRowWithIndexPath:` (page 937)

Declared In
`UITableView.h`

tableView:didDeselectRowAtIndexPath:

Tells the delegate that the specified row is now deselected.

```
- (void)tableView:(UITableView *)tableView didDeselectRowAtIndexPath:(NSIndexPath
    *)indexPath
```

Parameters

tableView
A table-view object informing the delegate about the row deselection.

indexPath
An index path locating the deselected row in *tableView*.

Discussion
The delegate handles row deselections in this method. It could, for example, remove the check-mark image (`UITableViewCellAccessoryCheckmark` (page 632)) associated with the row.

Availability
Available in iOS 3.0 and later.

See Also
- `tableView:willDeselectRowAtIndexPath:` (page 946)
- `tableView:didSelectRowAtIndexPath:` (page 939)

Declared In
`UITableView.h`

tableView:didEndEditingRowAtIndexPath:

Tells the delegate that the table view has left editing mode.

```
- (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; however, in this "swipe-to-delete" mode the table view does not display any insertion, deletion, and reordering controls. When entering this "swipe-to-delete" editing mode, the table view sends a `tableView:willBeginEditingRowAtIndexPath:` (page 945) message to the delegate to allow it to adjust its user interface.

Availability
Available in iOS 2.0 and later.

Declared In
`UITableView.h`

tableView:didSelectRowAtIndexPath:

Tells the delegate that the specified row is now selected.

```
- (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 632)) to one row in a section (radio-list style). This method isn't called when the `editing` (page 581) property of the table is set to `YES` (that is, the table view is in editing mode). See "Managing Selections" in *Table View Programming Guide for iOS* for further information (and code examples) related to this method.

Availability
Available in iOS 2.0 and later.

See Also
- `tableView:willSelectRowAtIndexPath:` (page 947)
- `tableView:didDeselectRowAtIndexPath:` (page 938)

Declared In
`UITableView.h`

tableView:editingStyleForRowAtIndexPath:

Asks the delegate for the editing style of a row at a particular location in a table view.

```
- (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 613) property set to `YES`), the cell has the `UITableViewCellEditingStyleDelete` (page 631) style set for it.

Availability

Available in iOS 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.

```
- (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 600)).

Availability

Available in iOS 2.0 and later.

See Also

`- tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:` (page 943)

`- tableView:viewForFooterInSection:` (page 944)

Declared In

`UITableView.h`

tableView:heightForHeaderInSection:

Asks the delegate for the height to use for the header of a particular section.

```
- (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.

Availability

Available in iOS 2.0 and later.

See Also

- tableView:willDisplayCell:forRowAtIndexPath: (page 946)
- tableView:viewForHeaderInSection: (page 944)

Declared In

UITableView.h

tableView:heightForRowAtIndexPath:

Asks the delegate for the height to use for a row in a specified location.

```
- (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. If this method is implemented, the value it returns overrides the value specified for the rowHeight (page 581) property of UITableView for the given row.

There are performance implications to using tableView:heightForRowAtIndexPath: instead of the rowHeight property. Every time a table view is displayed, it calls tableView:heightForRowAtIndexPath: on the delegate for each of its rows, which can result in a significant performance problem with table views having a large number of rows (approximately 1000 or more).

Important: Due to an underlying implementation detail, you should not return values greater than 2009.

Availability
Available in iOS 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.

```
- (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 iOS 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 942).

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
UITableView.h

tableView:titleForDeleteConfirmationButtonForRowAtIndexPath:

Changes the default title of the delete-confirmation button.

```
- (NSString *)tableView:(UITableView *)tableView
    titleForDeleteConfirmationButtonForRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView

The table-view object requesting this information.

indexPath

An index-path object locating the row in its section.

Return Value
A localized string to used as the title of the delete-confirmation button.

Discussion
By default, the delete-confirmation button, which appears on the right side of the cell, has the title of "Delete". The table view displays this button when the user attempts to delete a row, either by swiping the row or tapping the red minus icon in editing mode. You can implement this method to return an alternative title, which should be localized.

Availability
Available in iOS 3.0 and later.

See Also
- `tableView:editingStyleForRowAtIndexPath:` (page 939)

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.

```
- (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 600)).

Availability
Available in iOS 2.0 and later.

See Also
- `tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:` (page 943)
- `tableView:heightForFooterInSection:` (page 940)

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.

```
- (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 calls this method even if it was created in a plain style (`UITableViewStylePlain` (page 600)). This method only works correctly when `tableView:heightForHeaderInSection:` (page 941) is also implemented.

Availability

Available in iOS 2.0 and later.

See Also

`- tableView:willDisplayCell:forRowAtIndexPath:` (page 946)

Declared In

`UITableView.h`

tableView:willBeginEditingRowAtIndexPath:

Tells the delegate that the table view is about to go into editing mode.

```
- (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 581) property to `YES` (thereby entering editing mode) and displays a Delete button in the row identified by `indexPath`. In this "swipe to delete" mode the table view does not display any insertion, deletion, and reordering controls. 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 938).

Note: A swipe motion across a cell does not cause the display of a Delete button unless the table view's data source implements the `tableView:commitEditingStyle:forRowAtIndexPath:` (page 929) method.

Availability

Available in iOS 2.0 and later.

Declared In
UITableView.h

tableView:willDeselectRowAtIndexPath:

Tells the delegate that a specified row is about to be deselected.

```
- (NSIndexPath *)tableView:(UITableView *)tableView
    willDeselectRowAtIndexPath:(NSIndexPath *)indexPath
```

Parameters

tableView
A table-view object informing the delegate about the impending deselection.

indexPath
An index path locating the row in *tableView* to be deselected.

Return Value
An index-path object that confirms or alters the deselected row. Return an NSIndexPath object other than *indexPath* if you want another cell to be deselected. Return nil if you don't want the row deselected.

Discussion
This method is only called if there is an existing selection when the user tries to select a different row. The delegate is sent this method for the previously selected row. You can use UITableViewCellSelectionStyleNone to disable the appearance of the cell highlight on touch-down.

Availability
Available in iOS 3.0 and later.

See Also
- tableView:didDeselectRowAtIndexPath: (page 938)
- tableView:willSelectRowAtIndexPath: (page 947)

Declared In
UITableView.h

tableView:willDisplayCell:forRowAtIndexPath:

Tells the delegate the table view is about to draw a cell for a particular row.

```
- (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 iOS 2.0 and later.

See Also
- `tableView:cellForRowAtIndexPath:` (page 928) (`UITableViewDataSource`)
- `prepareForReuse` (page 626) (`UITableViewCell`)

Declared In
`UITableView.h`

tableView:willSelectRowAtIndexPath:

Tells the delegate that a specified row is about to be selected.

```
- (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. This method isn't called when the `editing` (page 581) property of the table is set to `YES` (that is, the table view is in editing mode).

Availability
Available in iOS 2.0 and later.

See Also
- `tableView:didSelectRowAtIndexPath:` (page 939)
- `tableView:shouldIndentWhileEditingRowAtIndexPath:` (page 942)
- `tableView:willDeselectRowAtIndexPath:` (page 946)

Declared In
`UITableView.h`

UITextFieldDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITextField.h
Related sample code	BonjourWeb MoviePlayer

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 951)
  Asks the delegate if editing should begin in the specified text field.
- `textFieldDidBeginEditing:` (page 950)
  Tells the delegate that editing began for the specified text field.
- `textFieldShouldEndEditing:` (page 952)
  Asks the delegate if editing should stop in the specified text field.
- `textFieldDidEndEditing:` (page 951)
  Tells the delegate that editing stopped for the specified text field.

Editing the Text Field's Text

- `textField:shouldChangeCharactersInRange:replacementString:` (page 950)
  Asks the delegate if the specified text should be changed.
- `textFieldShouldClear:` (page 952)
  Asks the delegate if the text field's current contents should be removed.

`- textFieldShouldReturn:` (page 953)
Asks the delegate if the text field should process the pressing of the return button.

Instance Methods

textField:shouldChangeCharactersInRange:replacementString:

Asks the delegate if the specified text should be changed.

```
- (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 iOS 2.0 and later.

Declared In
`UITextField.h`

textFieldDidBeginEditing:

Tells the delegate that editing began for the specified text field.

```
- (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 iOS 2.0 and later.

Declared In
`UITextField.h`

textFieldDidEndEditing:

Tells the delegate that editing stopped for the specified text field.

```
- (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 iOS 2.0 and later.

Declared In
`UITextField.h`

textFieldShouldBeginEditing:

Asks the delegate if editing should begin in the specified text field.

```
- (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 iOS 2.0 and later.

Declared In
UITextField.h

textFieldShouldClear:

Asks the delegate if the text field's current contents should be removed.

```
- (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 646) property of the text field.) This method is also called when editing begins and the clearsOnBeginEditing (page 646) 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 iOS 2.0 and later.

Declared In
UITextField.h

textFieldShouldEndEditing:

Asks the delegate if editing should stop in the specified text field.

```
- (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 iOS 2.0 and later.

Declared In
`UITextField.h`

textFieldShouldReturn:

Asks the delegate if the text field should process the pressing of the return button.

```
- (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 iOS 2.0 and later.

Declared In
`UITextField.h`

UITextInput Protocol Reference

Conforms to	UIKeyInput
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h
Companion guide	Text and Web Programming Guide for iOS

Overview

Classes that adopt the `UITextInput` protocol (and conform with inherited protocols) interact with the text input system and thus acquire features such as autocorrection and multistage text input for their documents. (Multistage text input is required when the language is ideographic and the keyboard is phonetic.)

Note: Here, a document is assumed to be a view capable of displaying and manipulating text.

Objects of classes that adopt the `UITextInput` protocol provide the text input system with text positions and text ranges on demand, answer questions about layout and writing direction, perform hit-testing (returning text positions and ranges for a given point), and provide the system with rectangles that can be used for highlighting ranges of text and drawing the caret. In addition, a `UITextInput` object maintains ranges for selected text and marked text.

Marked text, which is part of multistage text input, represents provisionally inserted text that the user has yet to confirm. It is styled in a distinctive way. The range of marked text always contains within it a range of selected text, which might be a range of characters or the caret.

The `UITextInput` protocol is the center of a constellation of classes and protocols for integrating text-processing applications with the text input system. The other parts of this constellation are the following:

- `UITextPosition` and `UITextRange` classes—All `UITextInput`-conforming document classes must create custom subclasses of these classes. A UITextPosition object represents a position in a text container. A `UITextRange` object, which encapsulates beginning and ending `UITextPosition` objects, represents a range of characters in the text container.
- `UITextInputTokenizer` protocol and `UITextInputStringTokenizer` class—The protocol defines an interface for a tokenizer object that enables the text input system to evaluate text units of different granularities. The class is a default implementation of this protocol.

- `UITextInputDelegate` protocol—The text input system automatically assigns its own text input delegate (which conforms to this protocol) to the UITextInput-conforming document object. Through this text input delegate, a document object informs the text input system of changes in text and selection.
- `UIKeyInput` protocol—Implemented to acquire the capabilities of text entry and deletion at an insertion point.

The `UITextInput` protocol also inherits the `UITextInputTraits` protocol, and thus the ability to customize the keyboard and its behaviors.

Tasks

Replacing and Returning Text

`- textInRange:` (page 970) *required method*
Return the text in the specified range. (required)

`- replaceRange:withText:` (page 968) *required method*
Replace the text in a document that is in the specified range. (required)

Working with Marked and Selected Text

`selectedTextRange` (page 960) *required property*
The range of selected text in a document. (required)

`markedTextRange` (page 959) *required property*
The range of text that is currently marked in a document. (required) (read-only)

`markedTextStyle` (page 959) *required property*
A dictionary of attributes that describes how marked text should be drawn. (required)

`- setMarkedText:selectedRange:` (page 969) *required method*
Insert the provided text and marks it to indicate that it is part of an active input session. (required)

`- unmarkText` (page 971) *required method*
Unmark the currently marked text. (required)

`selectionAffinity` (page 960) *property*
The desired location for the insertion point.

Computing Text Ranges and Text Positions

`- textRangeFromPosition:toPosition:` (page 970) *required method*
Return the range between two text positions. (required)

`- positionFromPosition:offset:` (page 967) *required method*
Returns the text position at a given offset from another text position. (required)

`- positionFromPosition:inDirection:offset:` (page 966) *required method*
Returns the text position at a given offset in a specified direction from another text position. (required)

`beginningOfDocument` (page 958) *required property*
The text position for the beginning of a document. (required) (read-only)

`endOfDocument` (page 958) *required property*
The text position for the end of a document. (required) (read-only)

Evaluating Text Positions

- `comparePosition:toPosition:` (page 965) *required method*
Return how one text position compares to another text position. (required)
- `offsetFromPosition:toPosition:` (page 966) *required method*
Return the number of visible characters between one text position and another text position. (required)

Determining Layout and Writing Direction

- `positionWithinRange:farthestInDirection:` (page 968)
Return the text position that is at the farthest extent in a given layout direction within a range of text.
- `characterRangeByExtendingPosition:inDirection:` (page 963) *required method*
Return a text range from a given text position to its farthest extent in a certain direction of layout. (required)
- `baseWritingDirectionForPosition:inDirection:` (page 961) *required method*
Return the base writing direction for a position in the text going in a certain direction. (required)
- `setBaseWritingDirection:forRange:` (page 969) *required method*
Set the base writing direction for a given range of text in a document. (required)

Geometry and Hit-Testing Methods

- `firstRectForRange:` (page 965) *required method*
Return the first rectangle that encloses a range of text in a document. (required)
- `caretRectForPosition:` (page 962) *required method*
Return a rectangle used to draw the caret at a given insertion point. (required)
- `closestPositionToPoint:` (page 964) *required method*
Return the position in a document that is closest to a specified point. (required)
- `closestPositionToPoint:withinRange:` (page 964) *required method*
Return the position in a document that is closest to a specified point in a given range. (required)
- `characterRangeAtPoint:` (page 963) *required method*
Return the character or range of characters that is at a given point in a document. (required)

Text Input Delegate and Text Input Tokenizer

`inputDelegate` (page 959) *required property*
An input delegate that is notified when text changes or when the selection changes. (required)

`tokenizer` (page 961) *required property*
An input tokenizer that provides information about the granularity of text units. (required) (read-only)

Returning Text Styling Information

`- textStylingAtPosition:inDirection:` (page 971)
Return a dictionary with properties that specify how text is to be style at a certain location in a document.

Reconciling Text Position and Character Offset

`- positionWithinRange:atCharacterOffset:` (page 967)
Return the position within a range of a document's text that corresponds to the character offset from the start of that range.

`- characterOffsetOfPosition:withinRange:` (page 962)
Return the character offset of a position in a document's text that falls within a given range.

Returning the Text Input View

`textInputView` (page 961) *property*
An affiliated view that provides a coordinate system for all geometric values in this protocol. (read-only)

Properties

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

beginningOfDocument

The text position for the beginning of a document. (required) (read-only)

```
@property(nonatomic, readonly) UITextPosition *beginningOfDocument
```

Availability
Available in iOS 3.2 and later.

See Also
`@property endOfDocument` (page 958)

Declared In
`UITextInput.h`

endOfDocument

The text position for the end of a document. (required) (read-only)

```
@property(nonatomic, readonly) UITextPosition *endOfDocument
```

Availability
Available in iOS 3.2 and later.

See Also
`@property beginningOfDocument` (page 958)

Declared In
`UITextInput.h`

inputDelegate

An input delegate that is notified when text changes or when the selection changes. (required)

```
@property(nonatomic, assign) id<UITextInputDelegate> inputDelegate
```

Discussion
The text input system automatically assigns a delegate to this property at runtime. It is the responsibility of the view that adopts the `UITextInput` protocol to notify the input delegate at the appropriate junctures.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

markedTextRange

The range of text that is currently marked in a document. (required) (read-only)

```
@property(nonatomic, readonly) UITextRange *markedTextRange
```

Discussion
If there is no marked text, the value of the property is `nil`. Marked text is provisionally inserted text that requires user confirmation; it occurs in multistage text input. The current selection, which can be a caret or an extended range, always occurs within the marked text.

Availability
Available in iOS 3.2 and later.

See Also
`@property markedTextStyle` (page 959)
`- setMarkedText:selectedRange:` (page 969)
`- unmarkText` (page 971)

Declared In
`UITextInput.h`

markedTextStyle

A dictionary of attributes that describes how marked text should be drawn. (required)

```
@property (nonatomic, copy) NSDictionary *markedTextStyle
```

Discussion
Marked text requires a unique visual treatment when displayed to users. See "Style Dictionary Keys" (page 974) for descriptions of the valid keys and values for this dictionary.

Availability
Available in iOS 3.2 and later.

See Also
`@property markedTextRange` (page 959)
`- setMarkedText:selectedRange:` (page 969)
`- unmarkText` (page 971)

Declared In
`UITextInput.h`

selectedTextRange

The range of selected text in a document. (required)

```
@property(readwrite, copy) UITextRange *selectedTextRange
```

Discussion
If the text range has a length, it indicates the currently selected text. If it has zero length, it indicates the caret (insertion point). If the text-range object is `nil`, it indicates that there is no current selection.

Availability
Available in iOS 3.2 and later.

See Also
`@property markedTextRange` (page 959)
`empty` (page 664) (`UITextRange`)

Declared In
`UITextInput.h`

selectionAffinity

The desired location for the insertion point.

```
@property (nonatomic) UITextStorageDirection selectionAffinity
```

Discussion
For text selections that wrap across line boundaries, this property determines whether the insertion point appears after the last character on the line or before the first character on the following line. The selection affinity is set in response to the user navigating via the keyboard (for example, command-right-arrow). The text input system checks this property when it moves the insertion point around in a document.

In the default implementation, if the selection is not at the end of the line, or if the selection is at the start of a paragraph for an empty line, a forward direction is assumed (`UITextStorageDirectionForward` (page 972)); otherwise, a backward direction `UITextStorageDirectionBackward` (page 972) is assumed.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

textInputView

An affiliated view that provides a coordinate system for all geometric values in this protocol. (read-only)

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

Discussion
The view that both draws the text and provides a coordinate system for all geometric values in this protocol. (This is typically an instance of the `UITextInput`-adopting class.) If this property is unimplemented, the first view in the responder chain is selected.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

tokenizer

An input tokenizer that provides information about the granularity of text units. (required) (read-only)

```
@property(nonatomic, readonly) id<UITextInputTokenizer> tokenizer
```

Discussion
Standard units of granularity include characters, words, lines, and paragraphs. In most cases, you may lazily create and assign an instance of a subclass of `UITextInputStringTokenizer` for this purpose. If you require different behavior than this system-provided tokenizer, you can create a custom tokenizer that adopts the `UITextInputTokenizer` protocol.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

Instance Methods

baseWritingDirectionForPosition:inDirection:

Return the base writing direction for a position in the text going in a certain direction. (required)

```
- (UITextWritingDirection)baseWritingDirectionForPosition:(UITextPosition *)position
    inDirection:(UITextStorageDirection)direction
```

Parameters

position

An object that identifies a location in a document.

direction

A constant that indicates a direction of storage (forward or backward).

Return Value

A constant that represents a writing direction (for example, left-to-right or right-to-left)

Discussion

The base writing direction is set previously when the text input system sends a `setBaseWritingDirection:forRange:` (page 969) message to the conforming document object.

Availability

Available in iOS 3.2 and later.

Declared In

`UITextInput.h`

caretRectForPosition:

Return a rectangle used to draw the caret at a given insertion point. (required)

```
- (CGRect)caretRectForPosition:(UITextPosition *)position
```

Parameters

position

An object that identifies a location in a document.

Return Value

A rectangle that defines the area for drawing the caret.

Availability

Available in iOS 3.2 and later.

See Also

`- firstRectForRange:` (page 965)

Declared In

`UITextInput.h`

characterOffsetOfPosition:withinRange:

Return the character offset of a position in a document's text that falls within a given range.

```
- (NSInteger)characterOffsetOfPosition:(UITextPosition *)position
    withinRange:(UITextRange *)range
```

Parameters

position

An object that identifies a location in a document's text.

range

An object that specifies a range of text in a document.

Return Value
The number of characters in a document's text that occur between *position* and the beginning of *range*.

Discussion
You should implement this method if you don't have a one-to-one correspondence between `UITextPosition` objects within the given range and character offsets into a document string.

Availability
Available in iOS 3.2 and later.

See Also
`- positionWithinRange:atCharacterOffset:` (page 967)

Declared In
`UITextInput.h`

characterRangeAtPoint:

Return the character or range of characters that is at a given point in a document. (required)

```
- (UITextRange *)characterRangeAtPoint:(CGPoint)point
```

Parameters

point
A point in the view that is drawing a document's text.

Return Value
An object representing a range that encloses a character (or characters) at *point*.

Availability
Available in iOS 3.2 and later.

See Also
`- closestPositionToPoint:` (page 964)
`- closestPositionToPoint:withinRange:` (page 964)

Declared In
`UITextInput.h`

characterRangeByExtendingPosition:inDirection:

Return a text range from a given text position to its farthest extent in a certain direction of layout. (required)

```
- (UITextRange *)characterRangeByExtendingPosition:(UITextPosition *)position
    inDirection:(UITextLayoutDirection)direction
```

Parameters

position
A text-position object that identifies a location in a document.

direction
A constant that indicates a direction of layout (right, left, up, down).

Return Value
A text-range object that represents the distance from *position* to the farthest extent in *direction*.

Availability
Available in iOS 3.2 and later.

See Also
- positionWithinRange:farthestInDirection: (page 968)

Declared In
UITextInput.h

closestPositionToPoint:

Return the position in a document that is closest to a specified point. (required)

```
- (UITextPosition *)closestPositionToPoint:(CGPoint)point
```

Parameters
point
A point in the view that is drawing a document's text.

Return Value
An object locating a position in a document that is closest to *point*.

Availability
Available in iOS 3.2 and later.

See Also
- closestPositionToPoint:withinRange: (page 964)
- characterRangeAtPoint: (page 963)

Declared In
UITextInput.h

closestPositionToPoint:withinRange:

Return the position in a document that is closest to a specified point in a given range. (required)

```
- (UITextPosition *)closestPositionToPoint:(CGPoint)point withinRange:(UITextRange
    *)range
```

Parameters
point
A point in the view that is drawing a document's text.

range
An object representing a range in a document's text.

Return Value
An object representing the character position in *range* that is closest to *point*.

Availability
Available in iOS 3.2 and later.

See Also
- `closestPositionToPoint:` (page 964)
- `characterRangeAtPoint:` (page 963)

Declared In
`UITextInput.h`

comparePosition:toPosition:

Return how one text position compares to another text position. (required)

```
- (NSComparisonResult)comparePosition:(UITextPosition *)position
    toPosition:(UITextPosition *)other
```

Parameters

position
A custom object that represents a location within a document.

other
A custom object that represents another location within a document.

Return Value
A value that indicates whether the two text positions are identical or whether one is before the other.

Availability
Available in iOS 3.2 and later.

See Also
- `offsetFromPosition:toPosition:` (page 966)

Declared In
`UITextInput.h`

firstRectForRange:

Return the first rectangle that encloses a range of text in a document. (required)

```
- (CGRect)firstRectForRange:(UITextRange *)range
```

Parameters

range
An object that represents a range of text in a document.

Return Value
The first rectangle in a *range* of text. You might use this rectangle to draw a correction rectangle. The "first" in the name refers the rectangle enclosing the first line when the range encompasses multiple lines of text.

Availability
Available in iOS 3.2 and later.

See Also
- `caretRectForPosition:` (page 962)

Declared In
`UITextInput.h`

offsetFromPosition:toPosition:

Return the number of visible characters between one text position and another text position. (required)

```
- (NSInteger)offsetFromPosition:(UITextPosition *)fromPosition
    toPosition:(UITextPosition *)toPosition
```

Parameters

fromPosition
A custom object that represents a location within a document.

toPosition
A custom object that represents another location within document.

Return Value
The number of visible characters between *fromPosition* and *toPosition*.

Availability
Available in iOS 3.2 and later.

See Also
- `comparePosition:toPosition:` (page 965)

Declared In
`UITextInput.h`

positionFromPosition:inDirection:offset:

Returns the text position at a given offset in a specified direction from another text position. (required)

```
- (UITextPosition *)positionFromPosition:(UITextPosition *)position
    inDirection:(UITextLayoutDirection)direction offset:(NSInteger)offset
```

Parameters

position
A custom `UITextPosition` object that represents a location in a document.

direction
A `UITextLayoutDirection` (page 972) constant that represents the direction of the offset from *position*. Return `nil` if the computed text position is less than 0 or greater than the length of the backing string.

offset
A character offset from *position*.

Discussion
For an example of an implementation of the related method, `positionFromPosition:offset:` (page 967), see "Drawing and Managing Text"" in *Text and Web Programming Guide for iOS*.

Availability
Available in iOS 3.2 and later.

See Also
- `textRangeFromPosition:toPosition:` (page 970)
- `positionFromPosition:offset:` (page 967)

Declared In
`UITextInput.h`

positionFromPosition:offset:

Returns the text position at a given offset from another text position. (required)

```
- (UITextPosition *)positionFromPosition:(UITextPosition *)position
    offset:(NSInteger)offset;
```

Parameters

position

A custom `UITextPosition` object that represents a location in a document.

offset

A character offset from *position*. It can be a positive or negative value.

Return Value
A custom `UITextPosition` object that represents the location in a document that is at the specified offset from *position*. Return `nil` if the computed text position is less than 0 or greater than the length of the backing string.

Discussion
For an example of an implementation of this method, see "Drawing and Managing Text" in *Text and Web Programming Guide for iOS*.

Availability
Available in iOS 3.2 and later.

See Also
- `positionFromPosition:inDirection:offset:` (page 966)
- `textRangeFromPosition:toPosition:` (page 970)

Declared In
`UITextInput.h`

positionWithinRange:atCharacterOffset:

Return the position within a range of a document's text that corresponds to the character offset from the start of that range.

```
- (UITextPosition *)positionWithinRange:(UITextRange *)range
    atCharacterOffset:(NSInteger)offset
```

Parameters

range

An object that specifies a range of text in a document.

offset

A character offset from the start of *range*.

Return Value
An object that represents a position in a document's visible text.

Discussion
You should implement this method if you don't have a one-to-one correspondence between `UITextPosition` objects within the given range and character offsets into a document string.

Availability
Available in iOS 3.2 and later.

See Also
`- characterOffsetOfPosition:withinRange:` (page 962)

Declared In
`UITextInput.h`

positionWithinRange:farthestInDirection:

Return the text position that is at the farthest extent in a given layout direction within a range of text.

```
- (UITextPosition *)positionWithinRange:(UITextRange *)range
    farthestInDirection:(UITextLayoutDirection)direction
```

Parameters

range
A text-range object that demarcates a range of text in a document.

direction
A constant that indicates a direction of layout (right, left, up, down).

Return Value
A text-position object that identifies a location in the visible text.

Availability
Available in iOS 3.2 and later.

See Also
`- characterRangeByExtendingPosition:inDirection:` (page 963)

Declared In
`UITextInput.h`

replaceRange:withText:

Replace the text in a document that is in the specified range. (required)

```
- (void)replaceRange:(UITextRange *)range withText:(NSString *)text
```

Parameters

range
A range of text in a document.

text
A string to replace the text in *range*.

Availability
Available in iOS 3.2 and later.

See Also
- `textInRange:` (page 970)

Declared In
`UITextInput.h`

setBaseWritingDirection:forRange:

Set the base writing direction for a given range of text in a document. (required)

```
- (void)setBaseWritingDirection:(UITextWritingDirection)writingDirection
    forRange:(UITextRange *)range
```

Parameters

writingDirection
A constant that represents a writing direction (for example, left-to-right or right-to-left)

range
An object that represents a range of text in a document.

Availability
Available in iOS 3.2 and later.

See Also
- `baseWritingDirectionForPosition:inDirection:` (page 961)

Declared In
`UITextInput.h`

setMarkedText:selectedRange:

Insert the provided text and marks it to indicate that it is part of an active input session. (required)

```
- (void)setMarkedText:(NSString *)markedText selectedRange:(NSRange)selectedRange
```

Parameters

markedText
The text to be marked.

selectedRange
A range within *markedText* that indicates the current selection. This range is always relative to *markedText*.

Discussion
Setting marked text either replaces the existing marked text or, if none is present, inserts it in place of the current selection.

Availability
Available in iOS 3.2 and later.

See Also
- `@property markedTextRange` (page 959)

`@property markedTextStyle` (page 959)
`- unmarkText` (page 971)

Declared In
`UITextInput.h`

textInRange:

Return the text in the specified range. (required)

```
- (NSString *)textInRange:(UITextRange *)range
```

Parameters

range
A range of text in a document.

Return Value
A substring of a document that falls within the specified range.

Availability
Available in iOS 3.2 and later.

See Also
`- replaceRange:withText:` (page 968)

Declared In
`UITextInput.h`

textRangeFromPosition:toPosition:

Return the range between two text positions. (required)

```
- (UITextRange *)textRangeFromPosition:(UITextPosition *)fromPosition
    toPosition:(UITextPosition *)toPosition
```

Parameters

fromPosition
An object that represents a location in a document.

toPosition
An object that represents another location in a document.

Return Value
An object that represents the range between *fromPosition* and *toPosition*.

Availability
Available in iOS 3.2 and later.

See Also
`- positionFromPosition:offset:` (page 967)
`- positionFromPosition:inDirection:offset:` (page 966)

Declared In
`UITextInput.h`

textStylingAtPosition:inDirection:

Return a dictionary with properties that specify how text is to be style at a certain location in a document.

```
- (NSDictionary *)textStylingAtPosition:(UITextPosition *)position
    inDirection:(UITextStorageDirection)direction
```

Parameters

position

An object that indicates a location in the text of a document.

direction

The direction of the styling attributes in text storage.

Return Value

A dictionary whose elements are one or more of the key-value pairs defining text color, font, and background color. See "Style Dictionary Keys" (page 974) for descriptions of these key-value pairs.

Discussion

Text styling information can affect, for example, the appearance of a correction rectangle.

Availability

Available in iOS 3.2 and later.

Declared In

`UITextInput.h`

unmarkText

Unmark the currently marked text. (required)

```
- (void)unmarkText
```

Discussion

After this method is called, the value of `markedTextRange` (page 959) is `nil`.

Availability

Available in iOS 3.2 and later.

See Also

`@property markedTextRange` (page 959)

`@property markedTextRange` (page 959)

`- setMarkedText:selectedRange:` (page 969)

Declared In

`UITextInput.h`

Constants

UITextStorageDirection

The direction of text storage.

```
typedef enum {
    UITextStorageDirectionForward = 0,
    UITextStorageDirectionBackward
} UITextStorageDirection;
```

Constants

`UITextStorageDirectionForward`
Storage of the text in a forward direction.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextStorageDirectionBackward`
Storage of the text in a backward direction.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

Discussion

Constants of this type are used as arguments to the `baseWritingDirectionForPosition:inDirection:` (page 961) and `textStylingAtPosition:inDirection:` (page 971) methods.

Availability

Available in iOS 3.2 and later.

Declared In

`UITextInput.h`

UITextLayoutDirection

The direction of text layout.

```
typedef enum {
    UITextLayoutDirectionRight = 2,
    UITextLayoutDirectionLeft,
    UITextLayoutDirectionUp,
    UITextLayoutDirectionDown
} UITextLayoutDirection;
```

Constants

`UITextLayoutDirectionRight`
Layout of the text to the right.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextLayoutDirectionLeft`
Layout of the text to the left.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextLayoutDirectionUp`
Layout of the text in an upward direction.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextLayoutDirectionDown`
Layout of the text in a downward direction.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

Discussion
Constants of this type are used as arguments in the `positionFromPosition:inDirection:offset:` (page 966), `positionWithinRange:farthestInDirection:` (page 968), and `characterRangeByExtendingPosition:inDirection:` (page 963) methods.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

UITextWritingDirection

The writing direction of the text, based on language.

```
typedef enum {
    UITextWritingDirectionNatural = -1,
    UITextWritingDirectionLeftToRight = 0,
    UITextWritingDirectionRightToLeft,
} UITextWritingDirection;
```

Constants
`UITextWritingDirectionNatural`
The natural writing direction as defined by the Bidi algorithm.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

`UITextWritingDirectionLeftToRight`
Writing that goes from left to right.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

`UITextWritingDirectionRightToLeft`
Writing that goes from right to left.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

Discussion
Constants of this type are returned from the `baseWritingDirectionForPosition:inDirection:` (page 961) method and are used as arguments of the `setBaseWritingDirection:forRange:` (page 969) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

Style Dictionary Keys

A dictionary containing properties that define text style characteristics.

```
NSString *const UITextInputTextBackgroundColorKey;
NSString *const UITextInputTextColorKey;
NSString *const UITextInputTextFontKey;
```

Constants

`UITextInputTextBackgroundColorKey`
The background color of the text. The value of this key is a `UIColor` object.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

`UITextInputTextColorKey`
The color of the text. The value of this key is a `UIColor` object.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

`UITextInputTextFontKey`
The font of the text. The value of this key is a `UIFont` object.

Available in iOS 3.2 and later.

Declared in `UITextInput.h`.

Discussion

The style `NSDictionary` object is used for providing styling information for marked text (`markedTextStyle` (page 959) property) and for providing text-styling information at a certain position (`textStylingAtPosition:inDirection:` (page 971) method).

UITextInputDelegate Protocol Reference

Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h

Overview

The text input delegate acts as an intermediary between a document and the text input system, conveying notifications of pending or transpired changes in text and selection in the document.

The UIKit provides a private text input delegate, which it assigns at runtime to the inputDelegate property of the object whose class adopts the UITextInput protocol.

Tasks

Notifying the Delegate of Textual Changes

- `textWillChange:` (page 977) *required method*
  Tells the input delegate when text is about to change in the document. (required)
- `textDidChange:` (page 976) *required method*
  Tells the input delegate when text has changed in the document. (required)

Notifying the Delegate of Selection Changes

- `selectionWillChange:` (page 976) *required method*
  Tells the input delegate when the selection is about to change in the document. (required)
- `selectionDidChange:` (page 976) *required method*
  Tells the input delegate when the selection has changed in the document. (required)

Instance Methods

selectionDidChange:

Tells the input delegate when the selection has changed in the document. (required)

```
- (void)selectionDidChange:(id <UITextInput>)textInput
```

Parameters

textInput

The document instance whose class adopts the UITextInput protocol.

Availability

Available in iOS 3.2 and later.

See Also

- textDidChange: (page 976)

Declared In

UITextInput.h

selectionWillChange:

Tells the input delegate when the selection is about to change in the document. (required)

```
- (void)selectionWillChange:(id <UITextInput>)textInput
```

Parameters

textInput

The document instance whose class adopts the UITextInput protocol.

Availability

Available in iOS 3.2 and later.

See Also

- textWillChange: (page 977)

Declared In

UITextInput.h

textDidChange:

Tells the input delegate when text has changed in the document. (required)

```
- (void)textDidChange:(id <UITextInput>)textInput
```

Parameters

textInput

The document instance whose class adopts the UITextInput protocol.

Availability

Available in iOS 3.2 and later.

See Also
- selectionDidChange: (page 976)

Declared In
UITextInput.h

textWillChange:

Tells the input delegate when text is about to change in the document. (required)

```
- (void)textWillChange:(id <UITextInput>)textInput
```

Parameters

textInput
The document instance whose class adopts the UITextInput protocol.

Availability
Available in iOS 3.2 and later.

See Also
- selectionWillChange: (page 976)

Declared In
UITextInput.h

UITextInputTokenizer Protocol Reference

Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Declared in	UITextInput.h

Overview

An instance of a class that adopts the `UITextInputTokenizer` protocol is a tokenizer; a tokenizer allows the text input system to evaluate text units of different granularities. Granularities of text units are always evaluated with reference to a storage or reference direction.

Text-processing objects that conform to the UITextInput protocol must hold a reference to a tokenizer (via the tokenizer property). The UITextInputStringTokenizer class of the UIKit framework provides a default base implementation of the `UITextInputTokenizer` protocol. Tokenizers of this class are suitable for most western-language keyboards. Applications with different requirements may adopt the `UITextInputTokenizer` protocol and create their own tokenizers.

Tasks

Determining Text Positions Relative to Unit Boundaries

`- isPosition:atBoundary:inDirection:` (page 980) *required method*

Return whether a text position is at a boundary of a text unit of a specified granularity in a specified direction. (required)

`- isPosition:withinTextUnit:inDirection:` (page 980) *required method*

Return whether a text position is within a text unit of a specified granularity in a specified direction. (required)

Computing Text Position by Unit Boundaries

`- positionFromPosition:toBoundary:inDirection:` (page 981) *required method*

Return the next text position at a boundary of a text unit of the given granularity in a given direction. (required)

Getting Ranges of Specific Text Units

`- rangeEnclosingPosition:withGranularity:inDirection:` (page 981) *required method*
Return the range for the text enclosing a text position in a text unit of a given granularity in a given direction. (required)

Instance Methods

isPosition:atBoundary:inDirection:

Return whether a text position is at a boundary of a text unit of a specified granularity in a specified direction. (required)

```
- (BOOL)isPosition:(UITextPosition *)position
    atBoundary:(UITextGranularity)granularity inDirection:(UITextDirection)direction
```

Parameters

position
A text-position object that represents a location in a document.

granularity
A constant that indicates a certain granularity of text unit.

direction
A constant that indicates a direction relative to *position*. The constant can be of type UITextStorageDirection or UITextLayoutDirection.

Return Value
`YES` if the text position is at the given text-unit boundary in the given direction; `NO` if it is not at the boundary.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

isPosition:withinTextUnit:inDirection:

Return whether a text position is within a text unit of a specified granularity in a specified direction. (required)

```
- (BOOL)isPosition:(UITextPosition *)position
    withinTextUnit:(UITextGranularity)granularity
    inDirection:(UITextDirection)direction
```

Parameters

position
A text-position object that represents a location in a document.

granularity
A constant that indicates a certain granularity of text unit.

direction

A constant that indicates a direction relative to *position*. The constant can be of type UITextStorageDirection or UITextLayoutDirection.

Return Value
YES if the text position is within a text unit of the specified granularity in the specified direction; otherwise, return NO. If the text position is *at* a boundary, return YES only if the boundary is part of the text unit in the given direction.

Availability
Available in iOS 3.2 and later.

Declared In
UITextInput.h

positionFromPosition:toBoundary:inDirection:

Return the next text position at a boundary of a text unit of the given granularity in a given direction. (required)

```
- (UITextPosition *)positionFromPosition:(UITextPosition *)position
    toBoundary:(UITextGranularity)granularity inDirection:(UITextDirection)direction
```

Parameters

position

A text-position object that represents a location in a document.

granularity

A constant that indicates a certain granularity of text unit.

direction

A constant that indicates a direction relative to *position*. The constant can be of type UITextStorageDirection or UITextLayoutDirection.

Return Value
The next boundary position of a text unit of the given granularity in the given direction, or nil if there is no such position.

Availability
Available in iOS 3.2 and later.

Declared In
UITextInput.h

rangeEnclosingPosition:withGranularity:inDirection:

Return the range for the text enclosing a text position in a text unit of a given granularity in a given direction. (required)

```
- (UITextRange *)rangeEnclosingPosition:(UITextPosition
    *)positionwithGranularity:(UITextGranularity)granularityinDirection:(UITextDirection)direction
```

Parameters

position

A text-position object that represents a location in a document.

granularity
A constant that indicates a certain granularity of text unit.

direction
A constant that indicates a direction relative to *position*. The constant can be of type UITextStorageDirection or UITextLayoutDirection.

Return Value
A text-range representing a text unit of the given granularity in the given direction, or `nil` if there is no such enclosing unit. Whether a boundary position is enclosed depends on the given direction, using the same rule as the `isPosition:withinTextUnit:inDirection:` (page 980) method.

Discussion
In this method, return the range for the text enclosing a text position in a text unit of the given granularity, or `nil` if there is no such enclosing unit. If the text position is entirely enclosed within a text unit of the given granularity, it is considered enclosed. If the text position is at a text-unit boundary, it is considered enclosed only if the next position in the given direction is entirely enclosed.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

Constants

UITextDirection

A direction of the text.

```
typedef int UITextDirection;
```

Discussion
This parameter is used in methods declared by the `UITextInputTokenizer` protocol. This general direction type subsumes constants of the UITextStorageDirection and UITextLayoutDirection types.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

UITextGranularity

The granularity of a unit of text.

```
typedef enum {
    UITextGranularityCharacter,
    UITextGranularityWord,
    UITextGranularitySentence,
    UITextGranularityParagraph,
    UITextGranularityLine,
    UITextGranularityDocument
} UITextGranularity;
```

Constants

`UITextGranularityCharacter`
The unit of text is a character.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextGranularityWord`
The unit of text is a word.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextGranularitySentence`
The unit of text is a sentence.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextGranularityParagraph`
The unit of text is a paragraph.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextGranularityLine`
The unit of text is a line.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

`UITextGranularityDocument`
The unit of text is a document.
Available in iOS 3.2 and later.
Declared in `UITextInput.h`.

Discussion
Constants of this type are used as parameters in all methods of the `UITextInputTokenizer` protocol.

Availability
Available in iOS 3.2 and later.

Declared In
`UITextInput.h`

UITextInputTraits Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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 986) *required property*
The auto-capitalization style for the text object. (required)

`autocorrectionType` (page 986) *required property*
The auto-correction style for the text object. (required)

`enablesReturnKeyAutomatically` (page 986) *required property*
A Boolean value indicating whether the return key is automatically enabled when text is entered by the user. (required)

`keyboardAppearance` (page 987) *required property*
The appearance style of the keyboard that is associated with the text object (required)

`keyboardType` (page 987) *required property*
The keyboard style associated with the text object. (required)

`returnKeyType` (page 987) *required property*
The contents of the “return” key. (required)

`secureTextEntry` (page 988) *required property*
Identifies whether the text object should hide the text being entered. (required)

Properties

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

autocapitalizationType

The auto-capitalization style for the text object. (required)

```
@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 `UITextAutocapitalizationTypeSentences` (page 989).

Some keyboard types do not support auto-capitalization. Specifically, this option is ignored if the value in the `keyboardType` (page 987) property is set to `UIKeyboardTypeNumberPad` (page 990), `UIKeyboardTypePhonePad` (page 990), or `UIKeyboardTypeNamePhonePad` (page 990).

Availability
Available in iOS 2.0 and later.

Declared In
`UITextInputTraits.h`

autocorrectionType

The auto-correction style for the text object. (required)

```
@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 iOS 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. (required)

```
@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 iOS 2.0 and later.

Declared In
`UITextInputTraits.h`

keyboardAppearance

The appearance style of the keyboard that is associated with the text object (required)

```
@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 iOS 2.0 and later.

Declared In
`UITextInputTraits.h`

keyboardType

The keyboard style associated with the text object. (required)

```
@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 iOS 2.0 and later.

Declared In
`UITextInputTraits.h`

returnKeyType

The contents of the “return” key. (required)

```
@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 iOS 2.0 and later.

Declared In
`UITextInputTraits.h`

secureTextEntry

Identifies whether the text object should hide the text being entered. (required)

```
@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 iOS 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 iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UITextAutocapitalizationTypeWords`
Capitalize the first letter of each word automatically.
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UITextAutocapitalizationTypeSentences`
Capitalize the first letter of each sentence automatically.

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UITextAutocapitalizationTypeAllCharacters`
Capitalize all characters automatically.

Available in iOS 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 iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UITextAutocorrectionTypeNo`
Disable auto-correction behavior.

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UITextAutocorrectionTypeYes`
Enable auto-correction behavior.

Available in iOS 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,
    UIKeyboardTypeAlphabet = UIKeyboardTypeASCIICapable
} UIKeyboardType;
```

Constants

`UIKeyboardTypeDefault`

Use the default keyboard for the current input method.

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIKeyboardTypeASCIICapable`

Use a keyboard that displays standard ASCII characters.

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIKeyboardTypeNumbersAndPunctuation`

Use the numbers and punctuation keyboard.

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIKeyboardTypeURL`

Use a keyboard optimized for URL entry. This type features ".", "/", and ".com" prominently.

Available in iOS 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. This keyboard type does not support auto-capitalization.

Available in iOS 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. This keyboard type does not support auto-capitalization.

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIKeyboardTypeNamePhonePad`

Use a keypad designed for entering a person's name or phone number. This keyboard type does not support auto-capitalization.

Available in iOS 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 iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIKeyboardTypeAlphabet`
Deprecated.

Use `UIKeyboardTypeASCIICapable` (page 990) instead.

Available in iOS 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 iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIKeyboardAppearanceAlert`
Use a keyboard that is suitable for an alert panel.

Available in iOS 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 iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeyGo`
Set the text of the return key to "Go".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeyGoogle`
Set the text of the return key to "Google".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeyJoin`
Set the text of the return key to "Join".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeyNext`
Set the text of the return key to "Next".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeyRoute`
Set the text of the return key to "Route".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeySearch`
Set the text of the return key to "Search".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeySend`
Set the text of the return key to "Send".
Available in iOS 2.0 and later.
Declared in `UITextInputTraits.h`.

`UIReturnKeyYahoo`

Set the text of the return key to "Yahoo".

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIReturnKeyDone`

Set the text of the return key to "Done".

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

`UIReturnKeyEmergencyCall`

Set the text of the return key to "Emergency Call".

Available in iOS 2.0 and later.

Declared in `UITextInputTraits.h`.

UITextViewDelegate Protocol Reference

Conforms to	NSObject UIScrollViewDelegate
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UITextView.h
Related sample code	KeyboardAccessory

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 998)
  Asks the delegate if editing should begin in the specified text view.
- `textViewDidBeginEditing:` (page 996)
  Tells the delegate that editing of the specified text view has begun.
- `textViewShouldEndEditing:` (page 998)
  Asks the delegate if editing should stop in the specified text view.
- `textViewDidEndEditing:` (page 998)
  Tells the delegate that editing of the specified text view has ended.

Responding to Text Changes

- `textView:shouldChangeTextInRange:replacementText:` (page 996)
  Asks the delegate whether the specified text should be replaced in the text view.

- `textViewDidChange:` (page 997)
  Tells the delegate that the text or attributes in the specified text view changed.

Responding to Selection Changes

- `textViewDidChangeSelection:` (page 997)
  Tells the delegate that the text selection changed in the specified text view.

Instance Methods

textView:shouldChangeTextInRange:replacementText:

Asks the delegate whether the specified text should be replaced in the text view.

```
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range
    replacementText:(NSString *)text
```

Parameters

textView
  The text view containing the changes.

range
  The current selection range. If the length of the range is 0, *range* reflects the current insertion point. If the user presses the Delete key, the length of the range is 1 and an empty string object replaces that single character.

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 iOS 2.0 and later.

Declared In
`UITextView.h`

textViewDidBeginEditing:

Tells the delegate that editing of the specified text view has begun.

```
- (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 iOS 2.0 and later.

Declared In

UITextView.h

textViewDidChange:

Tells the delegate that the text or attributes in the specified text view changed.

```
- (void)textViewDidChange:(UITextView *)textView
```

Parameters

textView

The text view containing the changes.

Discussion

Implementation of this method is optional.

Availability

Available in iOS 2.0 and later.

Declared In

UITextView.h

textViewDidChangeSelection:

Tells the delegate that the text selection changed in the specified text view.

```
- (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 669) property of the text view to get the new selection.

Availability

Available in iOS 2.0 and later.

Declared In

UITextView.h

textViewDidEndEditing:

Tells the delegate that editing of the specified text view has ended.

```
- (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 iOS 2.0 and later.

Declared In

UITextView.h

textViewShouldBeginEditing:

Asks the delegate if editing should begin in the specified text view.

```
- (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 iOS 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 iOS 2.0 and later.

Declared In

UITextView.h

UIVideoEditorControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.1 and later.
Declared in	

Overview

The `UIVideoEditorControllerDelegate` protocol defines methods that your delegate object must implement to respond to the video editor. The methods of this protocol notify your delegate when the system has saved an edited movie or the user has cancelled editing to discard any changes. There is also a method for responding to errors encountered by the video editor.

The delegate methods are responsible for dismissing the video editor when the operation completes. To dismiss the editor, call the `dismissModalViewControllerAnimated:` method of the parent controller responsible for displaying the video editor. The video editor is described in *UIVideoEditorController Class Reference*.

Tasks

Closing the Video Editor

- `videoEditorController:didSaveEditedVideoToPath:` (page 1002) *required method*
  Called when the system has finished saving an edited movie. (required)
- `videoEditorControllerDidCancel:` (page 1002) *required method*
  Called when the user has cancelled a movie editing operation. (required)

Handling Errors

- `videoEditorController:didFailWithError:` (page 1002) *required method*
  Called when the video editor is unable to load or save a movie. (required)

Instance Methods

videoEditorController:didFailWithError:

Called when the video editor is unable to load or save a movie. (required)

```
- (void)videoEditorController:(UIVideoEditorController *)editor
    didFailWithError:(NSError *)error
```

Parameters

editor

The video editor that was unable to load or save a movie.

error

The loading or saving error.

Discussion

Loading a movie into the video editor could fail because of an invalid filesystem path or an invalid media format. Saving could fail because of a lack of disk space or other reasons.

Availability

Available in iOS 3.1 and later.

Declared In

`UIVideoEditorController.h`

videoEditorController:didSaveEditedVideoToPath:

Called when the system has finished saving an edited movie. (required)

```
- (void)videoEditorController:(UIVideoEditorController *)editor
    didSaveEditedVideoToPath:(NSString *)editedVideoPath
```

Parameters

editor

The video editor that has finished editing and saving a movie.

editedVideoPath

The filesystem path to the edited movie.

Availability

Available in iOS 3.1 and later.

Declared In

`UIVideoEditorController.h`

videoEditorControllerDidCancel:

Called when the user has cancelled a movie editing operation. (required)

```
- (void)videoEditorControllerDidCancel:(UIVideoEditorController *)editor
```

Parameters

editor

The video editor that the user cancelled, not wanting to save changes.

Availability

Available in iOS 3.1 and later.

Declared In

`UIVideoEditorController.h`

UIWebViewDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 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.

Important: Before releasing an instance of `UIWebView` for which you have set a delegate, you must first set the `UIWebView` delegate property to `nil` before disposing of the `UIWebView` instance. This can be done, for example, in the dealloc method where you dispose of the `UIWebView`.

Tasks

Loading Content

- `webView:shouldStartLoadWithRequest:navigationType:` (page 1006)
  Sent before a web view begins loading content.
- `webViewDidStartLoad:` (page 1007)
  Sent after a web view starts loading content.
- `webViewDidFinishLoad:` (page 1007)
  Sent after a web view finishes loading content.
- `webView:didFailLoadWithError:` (page 1006)
  Sent if a web view failed to load content.

Instance Methods

webView:didFailLoadWithError:

Sent if a web view failed to load content.

```
- (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 iOS 2.0 and later.

See Also
- `webView:shouldStartLoadWithRequest:navigationType:` (page 1006)
- `webViewDidStartLoad:` (page 1007)
- `webViewDidFinishLoad:` (page 1007)

Declared In
`UIWebView.h`

webView:shouldStartLoadWithRequest:navigationType:

Sent before a web view begins loading content.

```
- (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 iOS 2.0 and later.

See Also
- `webViewDidStartLoad:` (page 1007)
- `webViewDidFinishLoad:` (page 1007)
- `webView:didFailLoadWithError:` (page 1006)

Declared In
UIWebView.h

webViewDidFinishLoad:

Sent after a web view finishes loading content.

```
- (void)webViewDidFinishLoad:(UIWebView *)webView
```

Parameters

webView
The web view has finished loading.

Availability
Available in iOS 2.0 and later.

See Also
- webView:shouldStartLoadWithRequest:navigationType: (page 1006)
- webViewDidStartLoad: (page 1007)
- webView:didFailLoadWithError: (page 1006)

Declared In
UIWebView.h

webViewDidStartLoad:

Sent after a web view starts loading content.

```
- (void)webViewDidStartLoad:(UIWebView *)webView
```

Parameters

webView
The web view that has begun loading content.

Availability
Available in iOS 2.0 and later.

See Also
- webView:shouldStartLoadWithRequest:navigationType: (page 1006)
- webViewDidFinishLoad: (page 1007)
- webView:didFailLoadWithError: (page 1006)

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

UIBarStyle

Defines the stylistic appearance of different types of views.

```
typedef enum {
    UIBarStyleDefault          = 0,
    UIBarStyleBlack            = 1,

    UIBarStyleBlackOpaque      = 1, // Deprecated
    UIBarStyleBlackTranslucent = 2, // Deprecated
} 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 iOS 2.0 and later.

Declared in `UIInterface.h`.

`UIBarStyleBlack`

Use an opaque black style.

Available in iOS 3.0 and later.

Declared in `UIInterface.h`.

`UIBarStyleBlackOpaque`

Deprecated. Use `UIBarStyleBlack` instead.

Available in iOS 2.0 and later.

Declared in `UIInterface.h`.

`UIBarStyleBlackTranslucent`
Deprecated. Use `UIBarStyleBlack` and set the `translucent` (page 374) property to `YES` instead.

Available in iOS 2.0 and later.

Declared in `UIInterface.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIInterface.h`

UIDataDetectorTypes

Defines the types of information that can be detected in text-based content.

```
enum {
    UIDataDetectorTypePhoneNumber   = 1 << 0,
    UIDataDetectorTypeLink          = 1 << 1,
    UIDataDetectorTypeAddress       = 1 << 2,
    UIDataDetectorTypeCalendarEvent = 1 << 3,
    UIDataDetectorTypeNone          = 0,
    UIDataDetectorTypeAll           = NSUIntegerMax
};
typedef NSUInteger UIDataDetectorTypes;
```

Constants
`UIDataDetectorTypePhoneNumber`
Detect strings formatted as phone numbers.

Available in iOS 3.0 and later.

Declared in `UIDataDetectors.h`.

`UIDataDetectorTypeLink`
Detect strings formatted as URLs.

Available in iOS 3.0 and later.

Declared in `UIDataDetectors.h`.

`UIDataDetectorTypeAddress`
Detect strings formatted as addresses.

Available in iOS 4.0 and later.

Declared in `UIDataDetectors.h`.

`UIDataDetectorTypeCalendarEvent`
Detect strings formatted as calendar events.

Available in iOS 4.0 and later.

Declared in `UIDataDetectors.h`.

`UIDataDetectorTypeNone`
Do no data detection.

Available in iOS 3.0 and later.

Declared in `UIDataDetectors.h`.

`UIDataDetectorTypeAll`
Detect all available types of data.

Available in iOS 3.0 and later.

Declared in `UIDataDetectors.h`.

Availability
Available in iOS 3.0 and later.

Declared In
`UIDataDetectors.h`

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 iOS 2.0 and later.

Declared In
`UIGeometry.h`

Constants

PART IV

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 iOS 2.0 and later.

Declared in `UIGeometry.h`.

Interface Builder Constants

Type qualifiers used by Interface Builder to synchronize with Xcode.

```
#define IBAction void
#define IBOutlet
#define IBOutletCollection(ClassName)
```

Constants

`IBAction`

Type qualifier used by Interface Builder to synchronize actions. Use this type as the return type of any action methods defined in your project. For examples of how to use this identifier, see "Xcode Integration".

Available in iOS 2.0 and later.

Declared in `UINibDeclarations.h`.

`IBOutlet`

Identifier used to qualify an instance-variable declaration so that Interface Builder can synchronize the display and connection of outlets with Xcode. Insert this identifier immediately before the variable type in any variable declarations. For examples, including how to use it with the `@property` syntax, see "Xcode Integration".

Available in iOS 2.0 and later.

Declared in `UINibDeclarations.h`.

`IBOutletCollection`

Identifier used to qualify a one-to-many instance-variable declaration so that Interface Builder can synchronize the display and connection of outlets with Xcode. You can insert this macro only in front of variables typed as `NSArray` or `NSMutableArray`.

This macro takes an optional *`ClassName`* parameter. If specified, Interface Builder requires all objects added to the array to be instances of that class. For example, to define a property that stores only `UIView` objects, you could use a declaration similar to the following:

```
@property (nonatomic, retain) IBOutletCollection(UIView) NSArray *views;
```

For additional examples of how to declare outlets, including how to create outlets with the `@property` syntax, see "Xcode Integration".

Available in iOS 4.0 and later.

Declared in `UINibDeclarations.h`.

Discussion

For more information about how to use these constants, see "Communicating With Objects". For information about defining and using actions and outlets in Interface Builder, see *Interface Builder User Guide*.

Other References

UIKit Function Reference

Framework:	UIKit/UIKit.h
Declared in	UIApplication.h UIGeometry.h UIGraphics.h UIImagePickerController.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 1030)
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 1044)
Returns the data for the specified image in JPEG format.

`UIImageWriteToSavedPhotosAlbum` (page 1045)
Adds the specified image to the user's Saved Photos album.

`UIImagePNGRepresentation` (page 1045) Deprecated in iOS 3.0
Returns the data for the specified image in PNG format

Movie Saving

`UISaveVideoAtPathToSavedPhotosAlbum` (page 1049)
Adds the movie at the specified path to the user's Saved Photos album.

`UIVideoAtPathIsCompatibleWithSavedPhotosAlbum` (page 1050)
Returns a Boolean value indicating whether the specified video can be saved to user's Saved Photos album.

Graphics

`UIGraphicsPushContext` (page 1042)
Makes the specified graphics context the current context.

`UIGraphicsPopContext` (page 1042)
Removes the current graphics context from the top of the stack, restoring the previous context.

`UIGraphicsBeginImageContext` (page 1035)
Creates a bitmap-based graphics context and makes it the current context.

`UIGraphicsGetImageFromCurrentImageContext` (page 1041)
Returns an image based on the contents of the current bitmap-based graphics context.

`UIRectClip` (page 1047)
Modifies the current clipping path by intersecting it with the specified rectangle.

`UIRectFrame` (page 1048)
Draws a frame around the inside of the specified rectangle.

`UIRectFrameUsingBlendMode` (page 1049)
Draws a frame around the inside of a rectangle using the specified blend mode.

`UIGraphicsGetCurrentContext` (page 1040) Deprecated in iOS 3.0
Returns the current graphics context.

`UIRectFillUsingBlendMode` (page 1048) Deprecated in iOS 3.0
Fills a rectangle with the current fill color using the specified blend mode.

`UIGraphicsBeginImageContextWithOptions` (page 1035) Deprecated in iOS 3.0
Creates a bitmap-based graphics context with the specified options.

`UIGraphicsEndImageContext` (page 1040) Deprecated in iOS 3.0
Removes the current bitmap-based graphics context from the top of the stack.

`UIRectFill` (page 1047) Deprecated in iOS 3.0 Deprecated in iOS 3.0
Fills the specified rectangle with the current color.

PDF Creation

`UIGraphicsBeginPDFContextToData` (page 1037)
Creates a PDF-based graphics context that targets the specified mutable data object.

`UIGraphicsBeginPDFContextToFile` (page 1037)
Creates a PDF-based graphics context that targets a file at the specified path.

`UIGraphicsEndPDFContext` (page 1040)
Closes a PDF graphics context and pops it from the current context stack.

`UIGraphicsBeginPDFPage` (page 1038)
Marks the beginning of a new page in a PDF context and configures it using default values.

`UIGraphicsBeginPDFPageWithInfo` (page 1039)
Marks the beginning of a new page in a PDF context and configures it using the specified values.

`UIGraphicsGetPDFContextBounds` (page 1041)
Returns the current page bounds.

`UIGraphicsAddPDFContextDestinationAtPoint` (page 1034)
Creates a jump destination in the current page.

`UIGraphicsSetPDFContextDestinationForRect` (page 1043)
Links a rectangle on the current page to the specified jump destination.

`UIGraphicsSetPDFContextURLForRect` (page 1043) Deprecated in iOS 3.0
Links a rectangle on the current page to the specified URL.

String Conversions

`CGPointFromString` (page 1025)
Returns a Core Graphics point structure corresponding to the data in a given string.

`CGRectFromString` (page 1025)
Returns a Core Graphics rectangle structure corresponding to the data in a given string.

`CGSizeFromString` (page 1026)
Returns a Core Graphics size structure corresponding to the data in a given string.

`CGAffineTransformFromString` (page 1024)
Returns a Core Graphics affine transform structure corresponding to the data in a given string.

`UIEdgeInsetsFromString` (page 1032)
Returns a UIKit edge insets structure corresponding to the data in a given string.

`NSStringFromCGPoint` (page 1027)
Returns a string object formatted to contain the data from a point.

`NSStringFromCGSize` (page 1028)
Returns a string object formatted to contain the data from a size data structure.

`NSStringFromCGAffineTransform` (page 1027) Deprecated in iOS 3.0 Deprecated in iOS 3.2
Returns a string object formatted to contain the data from an affine transform.

`NSStringFromUIEdgeInsets` (page 1028) Deprecated in iOS 3.0 Deprecated in iOS 3.0
Returns a string object formatted to contain the data from an edge insets structure.

`NSStringFromCGRect` (page 1028) Deprecated in iOS 4.0
Returns a string object formatted to contain the data from a rectangle.

Setting Edge Insets

`UIEdgeInsetsEqualToEdgeInsets` (page 1032)
Compares two edge insets to determine if they are the same.

`UIEdgeInsetsInsetRect` (page 1033)
Adjusts a rectangle by the given edge insets.

`UIEdgeInsetsMake` (page 1034) Deprecated in iOS 3.0
Creates an edge inset for a button or view.

Interface Orientation Macros

`UIInterfaceOrientationIsPortrait` (page 1046)
Returns a Boolean value indicating whether the user interface is currently presented in a portrait orientation.

`UIInterfaceOrientationIsLandscape` (page 1046)
Returns a Boolean value indicating whether the user interface is currently presented in a landscape orientation.

Device Orientation Macros

`UIDeviceOrientationIsValidInterfaceOrientation` (page 1032)
Returns a Boolean value indicating whether the specified orientation constant is valid.

`UIDeviceOrientationIsPortrait` (page 1031)
Returns a Boolean value indicating whether the device is in a portrait orientation.

`UIDeviceOrientationIsLandscape` (page 1031)
Returns a Boolean value indicating whether the device is in a landscape orientation.

Interface Idiom Macro

`UI_USER_INTERFACE_IDIOM` (page 1051) Deprecated in iOS 3.0
Returns the interface idiom supported by the current device.

Accessibility

`UIAccessibilityPostNotification` (page 1029)
Posts a notification to assistive applications.

`UIAccessibilityIsVoiceOverRunning` (page 1029) Deprecated in iOS 3.0
Returns a Boolean value indicating whether VoiceOver is running.

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 separated 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 iOS 2.0 and later.

See Also
`NSStringFromCGAffineTransform` (page 1027)

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 separated 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 iOS 2.0 and later.

See Also
`NSStringFromCGPoint` (page 1027)

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*},{*w*, *h*}}", where *x* is the x coordinate, *y* is the y coordinate, *w* is the width, and *h* is the height. 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 separated 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 iOS 2.0 and later.

See Also
`NSStringFromCGRect` (page 1028)

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 "{*w*, *h*}", where *w* is the width and *h* is the height. The *w* and *h* 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 separated 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 iOS 2.0 and later.

See Also
`NSStringFromCGSize` (page 1028)

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 1024) for a discussion of the string format.

Availability
Available in iOS 2.0 and later.

See Also
CGAffineTransformFromString (page 1024)

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 1025) for a discussion of the string format.

Availability
Available in iOS 2.0 and later.

See Also
CGPointFromString (page 1025)

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 1025) for a discussion of the string format.

Availability

Available in iOS 2.0 and later.

See Also

CGRectFromString (page 1025)

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 1026) for a discussion of the string format.

Availability

Available in iOS 2.0 and later.

See Also

CGSizeFromString (page 1026)

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 1032) for a discussion of the string format.

Availability

Available in iOS 2.0 and later.

See Also

`UIEdgeInsetsFromString` (page 1032)

Declared In

`UIGeometry.h`

UIAccessibilityIsVoiceOverRunning

Returns a Boolean value indicating whether VoiceOver is running.

```
BOOL UIAccessibilityIsVoiceOverRunning();
```

Return Value

`YES` if VoiceOver is currently running; otherwise, `NO`.

Discussion

You can use this function to customize your application's UI specifically for VoiceOver users. For example, you might want UI elements that usually disappear quickly to persist onscreen for VoiceOver users. Note that you can also listen for the `UIAccessibilityVoiceOverStatusChanged` notification to find out when VoiceOver starts and stops.

Availability

Available in iOS 4.0 and later.

Declared In

`UIAccessibility.h`

UIAccessibilityPostNotification

Posts a notification to assistive applications.

```
void UIAccessibilityPostNotification (
   UIAccessibilityNotifications notification,
   id argument
);
```

Parameters

notification

The notification to post (see "Notifications" in *UIAccessibility Protocol Reference* for a list of notifications).

argument
The argument specified by the notification. Pass `nil` unless a notification specifies otherwise.

Discussion
Your application might need to post accessibility notifications if you have user interface components that change very frequently or that appear and disappear.

Availability
Available in iOS 3.0 and later.

Declared In
`UIAccessibility.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. Specify `nil` if you load the delegate object from your application's main nib file.

Return Value
Even though an integer return type is specified, this function never returns. When users terminate an iPhone application by pressing the Home button, the application immediately exits by calling the `exit` system function with an argument of zero.

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. If the application's `Info.plist` file specifies a main nib file to be loaded, by including the `NSMainNibFile` key and a valid nib file name for the value, this function loads that nib file.

Despite the declared return type, this function never returns. For more information on how this function behaves, see ""Build-Time Configuration Details"" in *iOS Application Programming Guide*.

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKRocket
GKTank
ScrollViewSuite
SpeakHere
WiTap

Declared In
`UIApplication.h`

UIDeviceOrientationIsLandscape

Returns a Boolean value indicating whether the device is in a landscape orientation.

```
#define UIDeviceOrientationIsLandscape(orientation) \
   ((orientation) == UIDeviceOrientationLandscapeLeft || \
   (orientation) == UIDeviceOrientationLandscapeRight)
```

Parameters

`orientation`

Specify the value of the `orientation` property of the `UIDevice` class.

Return Value
Returns `YES` if the device orientation is landscape, otherwise returns `NO`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIDevice.h`

UIDeviceOrientationIsPortrait

Returns a Boolean value indicating whether the device is in a portrait orientation.

```
#define UIDeviceOrientationIsPortrait(orientation)  \
   ((orientation) == UIDeviceOrientationPortrait || \
   (orientation) == UIDeviceOrientationPortraitUpsideDown)
```

Parameters

`orientation`

Specify the value of the `orientation` property of the `UIDevice` class.

Return Value
Returns `YES` if the device orientation is portrait, otherwise returns `NO`.

Availability
Available in iOS 2.0 and later.

Declared In
`UIDevice.h`

UIDeviceOrientationIsValidInterfaceOrientation

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

```
#define UIDeviceOrientationIsValidInterfaceOrientation(orientation) \
   ((orientation) == UIDeviceOrientationPortrait || \
   (orientation) == UIDeviceOrientationPortraitUpsideDown || \
   (orientation) == UIDeviceOrientationLandscapeLeft || \
   (orientation) == UIDeviceOrientationLandscapeRight)
```

Parameters

orientation

Specify the orientation constant to check.

Return Value

Returns `YES` if the specified orientation constant is valid or `NO` if it is not valid.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`UIEdgeInsetsMake` (page 1034)

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` (page 1013) 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` (page 1017).

Discussion

In general, you should use this function only to convert strings that were previously created using the `NSStringFromUIEdgeInsets` function.

Availability

Available in iOS 2.0 and later.

See Also

`NSStringFromUIEdgeInsets` (page 1028)

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 iOS 2.0 and later.

See Also

`UIEdgeInsetsMake` (page 1034)

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 iOS 2.0 and later.

See Also
UIEdgeInsetsEqualToEdgeInsets (page 1032)

Declared In
UIGeometry.h

UIGraphicsAddPDFContextDestinationAtPoint

Creates a jump destination in the current page.

```
void UIGraphicsAddPDFContextDestinationAtPoint (
   NSString *name,
   CGPoint point
);
```

Parameters

name

The name of the destination point. The name you assign is local to the PDF document and is what you use when creating links to this destination.

point

A point on the current page of the PDF context.

Discussion

This function marks the specified point in the current page as the destination of a jump. When the user taps a link that takes them to this jump destination, the PDF document scrolls until the specified point is visible.

If the current graphics context is not a PDF context, this function does nothing.

For information on how to create links to this destination, see the `UIGraphicsSetPDFContextDestinationForRect` (page 1043) function.

Availability

Available in iOS 3.2 and later.

Declared In

`UIGraphics.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

This function is equivalent to calling the `UIGraphicsBeginImageContextWithOptions` (page 1035) function with the opaque parameter set to `NO` and a scale factor of `1.0`.

Availability

Available in iOS 2.0 and later.

Declared In

`UIGraphics.h`

UIGraphicsBeginImageContextWithOptions

Creates a bitmap-based graphics context with the specified options.

```
void UIGraphicsBeginImageContextWithOptions(
   CGSize size,
   BOOL opaque,
   CGFloat scale
);
```

Parameters

size

The size (measured in points) of the new bitmap context. This represents the size of the image returned by the `UIGraphicsGetImageFromCurrentImageContext` function. To get the size of the bitmap in pixels, you must multiply the width and height values by the value in the *scale* parameter.

opaque

A Boolean flag indicating whether the bitmap is opaque. If you know the bitmap is fully opaque, you can specify NO for this parameter to optimize the bitmap storage.

scale

The scale factor to apply to the bitmap. If you specify a value of `0.0`, the scale factor is set to the scale factor of the device's main screen.

Discussion

You use this function to configure the drawing environment for rendering into a bitmap. The format for the bitmap is as follows:

- For bitmaps created in iOS 3.2 and later, the drawing environment uses the premultiplied ARGB format to store the bitmap data. If the opaque parameter is `YES`, the bitmap's alpha channel is ignored.
- For bitmaps created in iOS 3.1.x and earlier, the drawing environment uses the premultiplied RGBA format to store the 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 supplied scale factor is also applied to the coordinate system and resulting images. 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 iOS 4.0 and later.

See Also

`UIGraphicsGetCurrentContext` (page 1040)
`UIGraphicsGetImageFromCurrentImageContext` (page 1041)
`UIGraphicsEndImageContext` (page 1040)

Declared In

`UIGraphics.h`

UIGraphicsBeginPDFContextToData

Creates a PDF-based graphics context that targets the specified mutable data object.

```
void UIGraphicsBeginPDFContextToData (
   NSMutableData *data,
   CGRect bounds,
   NSDictionary *documentInfo
);
```

Parameters

data

The data object to receive the PDF output data.

bounds

A rectangle that specifies the default size and location of PDF pages. (This value is used as the default media box for each new page.) The origin of the rectangle should typically be (0, 0). Specifying an empty rectangle (`CGRectZero`) sets the default page size to 8.5 by 11 inches (612 by 792 points).

documentInfo

A dictionary that specifies additional information to be associated with the PDF file. You can use these keys to specify additional metadata and security information for the PDF, such as the author of the PDF or the password for accessing it. The keys in this dictionary are the same keys you pass to the `CGPDFContextCreate` function and are described in the Auxiliary Dictionary Keys section of *CGPDFContext Reference*. The dictionary is retained by the new context, so on return you may safely release it.

Specify `nil` if you do not want to associate any additional information with the PDF document.

Discussion

After creating the graphics context, this function makes it the current drawing context. Any subsequent drawing commands are therefore captured and turned into PDF data. When you are done drawing, you must call the `UIGraphicsEndPDFContext` function to close the PDF graphics context.

You can use all of the same drawing routines that you would normally use to draw the contents of your application. The graphics context converts all drawing commands into PDF drawing commands automatically. However, before you issue any drawing commands to a PDF context, you must start a new page by calling the `UIGraphicsBeginPDFPage` (page 1038) or `UIGraphicsBeginPDFPageWithInfo` (page 1039) function. You can also use these functions to define additional pages later.

After creating it, you can get the PDF context using the `UIGraphicsGetCurrentContext` (page 1040) function.

Availability

Available in iOS 3.2 and later.

See Also

`UIGraphicsEndPDFContext` (page 1040)

Declared In

`UIGraphics.h`

UIGraphicsBeginPDFContextToFile

Creates a PDF-based graphics context that targets a file at the specified path.

```
BOOL UIGraphicsBeginPDFContextToFile (
   NSString *path,
   CGRect bounds,
   NSDictionary *documentInfo
);
```

Parameters

path

A POSIX-style path string identifying the location of the resulting PDF file. The specified path may be relative or a full path name. If a file does not exist at the specified path, one is created; otherwise, the contents of any existing file are deleted. The directories in the path must exist.

bounds

A rectangle that specifies the default size and location of PDF pages. (This value is used as the default media box for each new page.) The origin of the rectangle should typically be (0, 0). Specifying an empty rectangle (`CGRectZero`) sets the default page size to 8.5 by 11 inches (612 by 792 points).

documentInfo

A dictionary that specifies additional information to be associated with the PDF file. You can use these keys to specify additional metadata and security information for the PDF, such as the author of the PDF or the password for accessing it. The keys in this dictionary are the same keys you pass to the `CGPDFContextCreate` function and are described in the Auxiliary Dictionary Keys section of *CGPDFContext Reference*. The dictionary is retained by the new context, so on return you may safely release it.

Specify `nil` if you do not want to associate any additional information with the PDF document.

Return Value
`YES` if the PDF context was created successfully or `NO` if it was not.

Discussion
After creating the graphics context, this function makes it the current drawing context. Any subsequent drawing commands are therefore captured and turned into PDF data. When you are done drawing, you must call the `UIGraphicsEndPDFContext` function to close the PDF graphics context.

You can use all of the same drawing routines that you would normally use to draw the contents of your application. However, before you issue any drawing commands to a PDF context, you must start a new page by calling the `UIGraphicsBeginPDFPage` (page 1038) or `UIGraphicsBeginPDFPageWithInfo` (page 1039) function. You can also use these functions to define additional pages later.

After creating it, you can get the PDF context using the `UIGraphicsGetCurrentContext` (page 1040) function.

Availability
Available in iOS 3.2 and later.

Declared In
`UIGraphics.h`

UIGraphicsBeginPDFPage

Marks the beginning of a new page in a PDF context and configures it using default values.

```
void UIGraphicsBeginPDFPage (
   void
);
```

Discussion
This function ends any previous page before beginning a new one. It sets the media box of the new page to the rectangle you specified when you created the PDF context.

If the current graphics context is not a PDF context, this function does nothing.

You must call this function or the `UIGraphicsBeginPDFPageWithInfo` (page 1039) function before you issue any drawing commands.

Availability
Available in iOS 3.2 and later.

Declared In
`UIGraphics.h`

UIGraphicsBeginPDFPageWithInfo

Marks the beginning of a new page in a PDF context and configures it using the specified values.

```
void UIGraphicsBeginPDFPageWithInfo (
   CGRect bounds,
   NSDictionary *pageInfo
);
```

Parameters

bounds

A rectangle that specifies the size and location of the new PDF page. This rectangle corresponds to the media box rectangle for the page.

pageInfo

A dictionary that specifies additional page-related information, such as the boxes that define different parts of the page. For a list of keys you can include in this dictionary, see Box Dictionary Keys in *CGPDFContext Reference*. The dictionary is retained by the new page, so you may release it after this function returns.

Specify `nil` if you do not want to associate any additional information with the page.

Discussion
This function ends any previous page before beginning a new one. It sets the media box of the new page to the value in the `kCGPDFContextMediaBox` key of the *pageInfo* dictionary, or to the value in the *bounds* parameter if the dictionary does not contain the key.

If the current graphics context is not a PDF context, this function does nothing.

You must call this function or the `UIGraphicsBeginPDFPageWithInfo` (page 1039) function before you issue any drawing commands.

Availability
Available in iOS 3.2 and later.

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 function does nothing.

Availability
Available in iOS 2.0 and later.

See Also
`UIGraphicsBeginImageContext` (page 1035)

Declared In
`UIGraphics.h`

UIGraphicsEndPDFContext

Closes a PDF graphics context and pops it from the current context stack.

```
void UIGraphicsEndPDFContext (
   void
);
```

Discussion
You must call this function after you finish drawing to a PDF graphics context. This function closes the current open page and removes the PDF context from the graphics context stack. It also releases the `CGContextRef` associated with the PDF context. If the current graphics context is not a PDF context, this function does nothing.

Availability
Available in iOS 3.2 and later.

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 1042) function.

You should call this function from the main thread of your application only.

Availability
Available in iOS 2.0 and later.

See Also
`UIGraphicsPushContext` (page 1042)
`UIGraphicsPopContext` (page 1042)

Related Sample Code
GKRocket
SpeakHere

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 iOS 2.0 and later.

See Also
`UIGraphicsBeginImageContext` (page 1035)

Declared In
`UIGraphics.h`

UIGraphicsGetPDFContextBounds

Returns the current page bounds.

```
CGRect UIGraphicsGetPDFContextBounds (
   void
);
```

Return Value
The current page bounds associated with the PDF context or `CGRectZero` if the current context is not a PDF context.

Discussion
If a page has not yet been started, this function returns the default media box you specified when you created the PDF context; otherwise, it returns the page bounds for the current page.

Availability
Available in iOS 3.2 and later.

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 1042) function.

You should call this function from the main thread of your application only.

Availability
Available in iOS 2.0 and later.

See Also
`UIGraphicsPushContext` (page 1042)

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 1042) function.

You should call this function from the main thread of your application only.

Availability
Available in iOS 2.0 and later.

See Also
`UIGraphicsPopContext` (page 1042)

Declared In
`UIGraphics.h`

UIGraphicsSetPDFContextDestinationForRect

Links a rectangle on the current page to the specified jump destination.

```
void UIGraphicsSetPDFContextDestinationForRect (
   NSString *name,
   CGRect rect
);
```

Parameters

name
A named destination in the PDF document. This is the same name you used when creating the jump destination using the `UIGraphicsAddPDFContextDestinationAtPoint` (page 1034) function.

rect
A rectangle on the current page of the PDF context.

Discussion
You use this function to create live links within a PDF document. Tapping the specified rectangle in the PDF document causes the document to display the contents at the associated jump destination.

If the current graphics context is not a PDF context, this function does nothing.

Availability
Available in iOS 3.2 and later.

Declared In
`UIGraphics.h`

UIGraphicsSetPDFContextURLForRect

Links a rectangle on the current page to the specified URL.

```
void UIGraphicsSetPDFContextURLForRect (
   NSURL *url,
   CGRect rect
);
```

Parameters

url

The URL to open.

rect

A rectangle on the current page of the PDF context.

Discussion
You use this function to create external links within a PDF document. If the URL you specify is a type handled by a different application, tapping the rectangle opens that application.

If the current graphics context is not a PDF context, this function does nothing.

Availability
Available in iOS 3.2 and later.

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
An autoreleased 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 function forces that data to be reloaded into memory.

Availability
Available in iOS 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

An autoreleased 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 function forces that data to be reloaded into memory.

Availability

Available in iOS 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.

completionSelector

The method selector, of the target object, to call. This optional 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 iOS 2.0 and later.

Declared In
`UIImagePickerController.h`

UIInterfaceOrientationIsLandscape

Returns a Boolean value indicating whether the user interface is currently presented in a landscape orientation.

```
#define UIInterfaceOrientationIsLandscape(orientation) \
    ((orientation) == UIInterfaceOrientationLandscapeLeft || \
    (orientation) == UIInterfaceOrientationLandscapeRight)
```

Parameters
orientation
Specify the orientation constant to check.

Return Value
Returns `YES` if the interface orientation is landscape, otherwise returns `NO`.

Discussion
The interface orientation can be different than the device orientation. You typically use this macro in your view controller code to check the current orientation.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.h`

UIInterfaceOrientationIsPortrait

Returns a Boolean value indicating whether the user interface is currently presented in a portrait orientation.

```
#define UIInterfaceOrientationIsPortrait(orientation) \
    ((orientation) == UIInterfaceOrientationPortrait || \
    (orientation) == UIInterfaceOrientationPortraitUpsideDown)
```

Parameters
orientation
Specify the orientation constant to check.

Return Value
Returns `YES` if the interface orientation is portrait, otherwise returns `NO`.

Discussion
The interface orientation can be different than the device orientation. You typically use this macro in your view controller code to check the current orientation.

Availability
Available in iOS 2.0 and later.

Declared In
`UIApplication.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 function 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 1042) function. When you are ready to restore the original clipping region, you can then use the `UIGraphicsPopContext` (page 1042) function to remove the current context and restore the previous graphics state.

You should call this function from the main thread of your application only.

Availability
Available in iOS 2.0 and later.

See Also
`UIGraphicsPushContext` (page 1042)
`UIGraphicsPopContext` (page 1042)

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 `kCGBlendModeCopy` blend mode.

You should call this function from the main thread of your application only.

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKRocket

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 function does nothing.

You should call this function from the main thread of your application only.

Availability
Available in iOS 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 `kCGBlendModeCopy` 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 function 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.

You should call this function from the main thread of your application only.

Availability
Available in iOS 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 function 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.

You should call this function from the main thread of your application only.

Availability
Available in iOS 2.0 and later.

Declared In
`UIGraphics.h`

UISaveVideoAtPathToSavedPhotosAlbum

Adds the movie at the specified path to the user's Saved Photos album.

```
void UISaveVideoAtPathToSavedPhotosAlbum (
   NSString *videoPath,
   id completionTarget,
   SEL completionSelector,
   void *contextInfo
);
```

Parameters

videoPath

The filesystem path to the movie file you want to save to the user's Saved Photos album.

completionTarget

The object whose selector should be called after the movie has been written to the user's device.

completionSelector

The method selector, of the target object, to call. This optional method should be of the form:

```
- (void) video: (NSString *) videoPath
    didFinishSavingWithError: (NSError *) error
    contextInfo: (void *) contextInfo;
```

contextInfo

An optional pointer to any context-specific data that you want passed to the completion selector.

Discussion

Before calling this function, you should call the `UIVideoAtPathIsCompatibleWithSavedPhotosAlbum` function to determine if it is possible to save videos to the Saved Photos album.

Availability

Available in iOS 3.1 and later.

See Also

`UIVideoAtPathIsCompatibleWithSavedPhotosAlbum` (page 1050)

Declared In

`UIImagePickerController.h`

UIVideoAtPathIsCompatibleWithSavedPhotosAlbum

Returns a Boolean value indicating whether the specified video can be saved to user's Saved Photos album.

```
BOOL UIVideoAtPathIsCompatibleWithSavedPhotosAlbum (
   NSString *videoPath
);
```

Parameters

videoPath

The filesystem path to the movie file you want to save.

Return Value

`YES` if the video can be saved to the photo album or `NO` if it cannot.

Discussion

Not all devices are able to play video files placed in the user's Saved Photos album. So, before attempting to save videos to the user's camera roll, you should call this function to ensure that saving the video is supported for the current device.

Availability
Available in iOS 3.1 and later.

Declared In
UIImagePickerController.h

UI_USER_INTERFACE_IDIOM

Returns the interface idiom supported by the current device.

```
#define UI_USER_INTERFACE_IDIOM() \
   ([[UIDevice currentDevice] respondsToSelector:@selector(userInterfaceIdiom)] ?
 \
      [[UIDevice currentDevice] userInterfaceIdiom] : \
      UIUserInterfaceIdiomPhone)
```

Return Value
UIUserInterfaceIdiomPhone (page 248) if the device is an iPhone or iPod touch or UIUserInterfaceIdiomPad (page 248) if the device is an iPad.

Availability
Available in iOS 3.2 and later.

Declared In
UIDevice.h

Document Revision History

This table describes the changes to *UIKit Framework Reference*.

Date	Notes
2010-04-22	Added accessibility protocol references.
2010-03-03	Updated for iOS 3.2.
2009-08-20	Updated for iOS 3.1.
	Updated class hierarchy illustration in Figure I-1 (page 25). Added links to *UIVideoEditorController Class Reference* and *UIVideoEditorControllerDelegate Protocol Reference*.
2009-05-26	Added missing protocol references and updated for iOS 3.0.
2008-05-18	New document that describes the programming interface for constructing and managing the user interface of iPhone applications.

Document Revision History