UIViewController Class Reference

Developer

Contents

UIViewController Class Reference

Inherits from	UIResponder : NSObject
Conforms to	NSCoding UIAppearanceContainer NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 2.0 and later.
Declared in	UINavigationController.h UIPopoverController.h UISplitViewController.h UITabBarController.h UIViewController.h UIViewControllerTransitionCoordinator.h
Companion guides	View Controller Programming Guide for iOS iOS 7 UI Transition Guide View Controller Catalog for iOS
Related sample code	AdvancedURLConnections GLAirplay iAdSuite UIKit Dynamics Catalog UnwindSegue

Overview

The `UIViewController` class provides the fundamental view-management model for all iOS apps. You rarely instantiate `UIViewController` objects directly. Instead, you instantiate subclasses of the `UIViewController` class based on the specific task each subclass performs.

A view controller manages a set of views that make up a portion of your app's user interface. As part of the controller layer of your app, a view controller coordinates its efforts with model objects and other controller objects—including other view controllers—so your app presents a single coherent user interface.

Where necessary, a view controller:

- resizes and lays out its views
- adjusts the contents of the views
- acts on behalf of the views when the user interacts with them

A view controller is tightly bound to the views it manages and takes part in the responder chain used to handle events. View controllers descend from the `UIResponder` class and are inserted into the responder chain between the managed root view and its superview, which typically belongs to a different view controller. If the view controller's view does not handle an event, the view controller has the option of handling the event or it can pass the event to the superview.

View controllers are rarely used in isolation. Instead, you use multiple view controllers, each of which owns a portion of your app's user interface. For example, one view controller might manage a table of items while a different view controller manages the display of a selected item from that table. Each view controller displays its own views to show the content it is responsible for.

Subclassing Notes

An app contains at least one custom subclass of `UIViewController` and more often there are several. These custom subclasses define behaviors specific to your app, such as how it handles user interactions with its views. The following sections provide a brief overview of some of the tasks your custom subclass performs. For more information about implementing your custom subclasses, see *View Controller Programming Guide for iOS*.

View Management

When you define a new subclass of `UIViewController`, you must specify the views to be managed by the controller. A typical view hierarchy consists of a view with flexible bounds—a reference to which is available in the `view` (page 40) property of this class—and one or more subviews that provide the actual content. The size and position of the root view is usually determined by another object that owns the view controller and displays its contents. The view controller determines the positions of any subviews it owns based on the size given to its root view by the owning object. A view controller is usually owned by a window or another view controller. If a view controller is owned by a window object, it acts as the window's root view controller. The view controller's root view is added as a subview of the window and resized to fill the window. If the view controller is owned by another view controller, then the parent view controller determines when and how the child view controller's contents are displayed.

The `UIViewController` class provides built-in support for loading a view controller's views whenever they are needed. Specifically, views are automatically loaded when the `view` property is accessed. There are a few ways you can implement your app to load these views:

- You can specify the views for a view controller using a storyboard created in Interface Builder. A storyboard contains preconfigured view controllers and their associated views and is the preferred way to develop your app's user interface. A key advantage of storyboards is that they can express relationships between different view controllers in your app. For example, you can state that one view controller's contents are contained inside another view controller or that a view controller is displayed as a transition (known as a segue) from another view controller . By allowing you to see the relationships between view controllers, storyboards make it easier to understand your app's behavior at a glance.

  At runtime, the view controller uses the storyboard to automatically instantiate and configure its views. Often, the view controller itself is automatically created by segues defined in the storyboard. When you define a view controller's contents using a storyboard, you never directly allocate and initialize the view controller object. Instead, when you need to programmatically instantiate the view controller, you do so by calling the `instantiateViewControllerWithIdentifier:` method on a `UIStoryboard` object.

- You can specify the views for a view controller using a nib file, also created in Interface Builder. Like a storyboard, a nib file allows you to create and configure a set of views. However, you cannot easily create or see the relationships between view controllers using nib files as you can when using storyboards.

  To initialize your view controller object using a nib, you use the `initWithNibName:bundle:` method to specify the nib file used by the view controller. Then, when the view controller needs to load its views, it automatically creates and configures the views using the information stored in the nib file.

- If you cannot define your views in a storyboard or a nib file, override the `loadView` (page 56) method to manually instantiate a view hierarchy and assign it to the `view` (page 40) property.

All of these techniques have the same end result, which is to create the appropriate set of views and expose them through the `view` (page 40) property.

Important: A view controller is the sole owner of its view and any subviews it creates. It is responsible for creating those views and for relinquishing ownership of them at the appropriate times such as when the view controller itself is released. If you use a storyboard or a nib file to store your view objects, each view controller object automatically gets its own copy of these views when the view controller asks for them. 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` and `autoresizingMask` 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. If your view controller supports auto layout and is a child of another view controller, you should call the view's `setTranslatesAutoresizingMaskIntoConstraints:` method to disable these constraints.

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 47) method, which releases unneeded memory.

Prior to iOS 6, when a low-memory warning occurred, the `UIViewController` class purged its views if it knew it could reload or recreate them again later. If this happens, it also calls the `viewWillUnload` (page 91) and `viewDidUnload` (page 90) methods to give your code a chance to relinquish ownership of any objects that are associated with your view hierarchy, including objects loaded from the nib file, objects created in your `viewDidLoad` (page 74) method, and objects created lazily at runtime and added to the view hierarchy. On iOS 6, views are never purged and these methods are never called. If your view controller needs to perform specific tasks when memory is low, it should override the `didReceiveMemoryWarning` method.

Handling View Rotations

In iOS 6 and later, your app supports the interface orientations defined in your app's `Info.plist` file. A view controller can override the `supportedInterfaceOrientations` (page 69) method to limit the list of supported orientations. Typically, the system calls this method only on the root view controller of the window or a view controller presented to fill the entire screen; child view controllers use the portion of the window provided for them by their parent view controller and no longer participate directly in decisions about what rotations are supported. The intersection of the app's orientation mask and the view controller's orientation mask is used to determine which orientations a view controller can be rotated into.

You can override the `preferredInterfaceOrientationForPresentation` (page 58) for a view controller that is intended to be presented full screen in a specific orientation.

In iOS 5 and earlier, the `UIViewController` class displays views in portrait mode only. To support additional orientations, you must override the `shouldAutorotateToInterfaceOrientation:` (page 89) 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. Generally, if your view controller is intended to be used as a child view controller, it should support all interface orientations.

When a rotation occurs for a visible view controller, the `willRotateToInterfaceOrientation:duration:` (page 79), `willAnimateRotationToInterfaceOrientation:duration:` (page 77), and `didRotateFromInterfaceOrientation:` (page 48) methods are called during the rotation. The `viewWillLayoutSubviews` (page 77) method is also called after the view is resized and positioned by its parent. If a view controller is not visible when an orientation change occurs, then the rotation methods are never called. However, the `viewWillLayoutSubviews` (page 77) method is called when the view becomes visible. Your implementation of this method can call the `statusBarOrientation` method to determine the device orientation.

Note: At launch time, apps should always set up their interface in a portrait orientation. After the `application:didFinishLaunchingWithOptions:` method returns, the app uses the view controller rotation mechanism described above to rotate the views to the appropriate orientation prior to showing the window.

View Event Notification

The `UIViewController` class automatically responds to many notifications, such as when the view controller's view is added to or removed from a window's view hierarchy or when it is resized. The `UIViewController` class provides specific methods that are called when these events occur. Subclasses can override these methods to implement specific behaviors.

Implementing a Container View Controller

A custom `UIViewController` subclass can also act as a container view controller. A container view controller manages the presentation of content of other view controllers it owns, also known as its child view controllers. A child's view can be presented as-is or in conjunction with views owned by the container view controller.

Your container view controller subclass should declare a public interface to associate its children. The nature of these methods is up to you and depends on the semantics of the container you are creating. You need to decide how many children can be displayed by your view controller at once, when those children are displayed, and where they appear in your view controller's view hierarchy. Your view controller class defines what relationships, if any, are shared by the children. By establishing a clean public interface for your container, you ensure that children use its capabilities logically, without accessing too many private details about how your container implements the behavior.

Your container view controller must associate a child view controller with itself before adding the child's root view to the view hierarchy. This allows iOS to properly route events to child view controllers and the views those controllers manage. Likewise, after it removes a child's root view from its view hierarchy, it should disconnect that child view controller from itself. To make or break these associations, your container calls specific methods defined by the base class. These methods are not intended to be called by clients of your container class; they are to be used only by your container's implementation to provide the expected containment behavior.

Here are the essential methods you might need to call:

`addChildViewController:` (page 42)

`removeFromParentViewController` (page 62)

`willMoveToParentViewController:` (page 78)

`didMoveToParentViewController:` (page 46)

Note: You are not required to override any methods when creating a container view controller.

By default, rotation and appearance callbacks are automatically forwarded to children. You may optionally override the `shouldAutomaticallyForwardRotationMethods` (page 67) and `shouldAutomaticallyForwardAppearanceMethods` (page 66) methods to take control of this behavior yourself.

State Preservation and Restoration

If you assign a value to the view controller's `restorationIdentifier` (page 34) property, the system may ask the view controller to encode itself when the app transitions to the background. When preserved, a view controller preserves the state of any views in its view hierarchy that also have restoration identifiers. View controllers do not automatically save any other state. If you are implementing a custom container view controller, you must encode any child view controllers yourself. Each child you encode must have a unique restoration identifier.

For more information about how the system determines which view controllers to preserve and restore, see *iOS App Programming Guide* .

Tasks

Creating a View Controller Using Nib Files

– `initWithNibName:bundle:` (page 52)
Returns a newly initialized view controller with the nib file in the specified bundle.

`nibName` (page 30) *property*
Return the name of the receiver's nib file, if one was specified. (read-only)

`nibBundle` (page 30) *property*
Return the name of the receiver's nib bundle if it exists. (read-only)

Using a Storyboard

– `shouldPerformSegueWithIdentifier:sender:` (page 68)
Determines whether the segue with the specified identifier should be triggered.

– `performSegueWithIdentifier:sender:` (page 57)
Initiates the segue with the specified identifier from the view controller's storyboard file.

– `prepareForSegue:sender:` (page 60)
Notifies the view controller that a segue is about to be performed.

`storyboard` (page 35) *property*
The storyboard from which the view controller originated. (read-only)

– `canPerformUnwindSegueAction:fromViewController:withSender:` (page 44)
Called on a view controller to determine whether it wants to respond to an unwind action.

– `transitionCoordinator` (page 70)
Returns a transition coordinator.

Managing the View

`view` (page 40) *property*
The view that the controller manages.

– `isViewLoaded` (page 56)
Returns a Boolean value indicating whether the view is currently loaded into memory.

– `loadView` (page 56)
Creates the view that the controller manages.

– `viewDidLoad` (page 74)
Called after the controller's view is loaded into memory.

`title` (page 37) *property*
A localized string that represents the view this controller manages.

– `viewDidUnload` (page 90) Deprecated in iOS 6.0
Called when the controller's view is released from memory. (Deprecated. Views are no longer purged under low-memory conditions and so this method is never called.)

– `viewWillUnload` (page 91) Deprecated in iOS 6.0
  Called just before releasing the controller's view from memory. (Deprecated. Views are no longer purged under low-memory conditions and so this method is never called.)

Handling Memory Warnings

– `didReceiveMemoryWarning` (page 47)
  Sent to the view controller when the app receives a memory warning.

Responding to View Events

– `viewWillAppear:` (page 75)
  Notifies the view controller that its view is about to be added to a view hierarchy.

– `viewDidAppear:` (page 73)
  Notifies the view controller that its view was added to a view hierarchy.

– `viewWillDisappear:` (page 76)
  Notifies the view controller that its view is about to be removed from a view hierarchy.

– `viewDidDisappear:` (page 73)
  Notifies the view controller that its view was removed from a view hierarchy.

– `viewWillLayoutSubviews` (page 77)
  Called to notify the view controller that its view is about to layout its subviews.

– `viewDidLayoutSubviews` (page 74)
  Called to notify the view controller that its view has just laid out its subviews.

Testing for Specific Kinds of View Transitions

– `isMovingFromParentViewController` (page 55)
  Returns a Boolean value that indicates that the view controller is in the process of being removed from its parent.

– `isMovingToParentViewController` (page 55)
  Returns a Boolean value that indicates that the view controller is in the process of being added to a parent.

– `isBeingPresented` (page 54)
: Returns a Boolean value that indicates whether the view controller is in the process of being presented by one of its ancestors.

– `isBeingDismissed` (page 54)
: Returns a Boolean value that indicates whether the view controller is in the process of being dismissed by one of its ancestors.

Configuring the View's Layout Behavior

– `updateViewConstraints` (page 71)
: Called when the view controller's view needs to update its constraints.

`automaticallyAdjustsScrollViewInsets` (page 20) *property*
: Specifies whether or not the view controller should automatically adjust its scroll view insets.

`bottomLayoutGuide` (page 21) *property*
: Indicates the lowest vertical extent for your onscreen content, for use with Auto Layout constraints. (read-only)

`topLayoutGuide` (page 38) *property*
: Indicates the highest vertical extent for your onscreen content, for use with Auto Layout constraints. (read-only)

`edgesForExtendedLayout` (page 24) *property*
: Indicates the extended edges to use for the layout.

`preferredContentSize` (page 31) *property*
: The preferred content size for any container view that is laying out a child view controller.

`extendedLayoutIncludesOpaqueBars` (page 25) *property*
: Indicates whether or not the extended layout includes opaque bars.

– `childViewControllerForStatusBarHidden` (page 45)
: Called when the system needs the view controller to use for determining status bar hidden/unhidden state.

– `childViewControllerForStatusBarStyle` (page 45)
: Called when the system needs the view controller to use for determining status bar style.

– `preferredStatusBarStyle` (page 59)
: The preferred status bar style for the view controller.

– `prefersStatusBarHidden` (page 60)
: Specifies whether the view controller prefers the status bar to be hidden or shown.

`modalPresentationCapturesStatusBarAppearance` (page 27) *property*
: Specifies whether a view controller, presented non–fullscreen, takes over control of status bar appearance from the presenting view controller.

– `preferredStatusBarUpdateAnimation` (page 59)
: Specifies the animation style to use for hiding and showing the status bar for the view controller.

– `setNeedsStatusBarAppearanceUpdate` (page 65)
: Indicates to the system that the view controller status bar attributes have changed.

`wantsFullScreenLayout` (page 92) *property* Deprecated in iOS 7.0
: A Boolean value indicating whether the view should underlap the status bar.

Configuring the View Rotation Settings

– `shouldAutorotate` (page 68)
: Returns whether the view controller's contents should auto rotate.

– `supportedInterfaceOrientations` (page 69)
: Returns all of the interface orientations that the view controller supports.

– `preferredInterfaceOrientationForPresentation` (page 58)
: Returns the interface orientation to use when presenting the view controller.

`interfaceOrientation` (page 25) *property*
: Convenience property that provides the current orientation of the interface, meaningful only if the view controller is taking up the full screen. (read-only)

+ `attemptRotationToDeviceOrientation` (page 41)
: Attempts to rotate all windows to the orientation of the device.

– `rotatingHeaderView` (page 63)
: Returns the header view to transition during an interface orientation change.

– `rotatingFooterView` (page 62)
: Returns the footer view to transition during an interface orientation change.

– `shouldAutorotateToInterfaceOrientation:` (page 89) Deprecated in iOS 6.0
: Returns a Boolean value indicating whether the view controller supports the specified orientation. (Deprecated. Override the `supportedInterfaceOrientations` (page 69) and `preferredInterfaceOrientationForPresentation` (page 58) methods instead.)

Responding to View Rotation Events

– `willRotateToInterfaceOrientation:duration:` (page 79)
: Sent to the view controller just before the user interface begins rotating.

– `willAnimateRotationToInterfaceOrientation:duration:` (page 77)
: Sent to the view controller before performing a one-step user interface rotation.

– `didRotateFromInterfaceOrientation:` (page 48)
: Sent to the view controller after the user interface rotates.

– `didAnimateFirstHalfOfRotationToInterfaceOrientation:` (page 83) Deprecated in iOS 5.0
: Sent to the view controller after the completion of the first half of the user interface rotation. (Deprecated. Use the one-step rotation technique instead. See the `willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.)

– `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 84) Deprecated in iOS 5.0
: Sent to the view controller before performing the first half of a user interface rotation. (Deprecated. Use the one-step rotation technique instead. See the `willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.)

– `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 85) Deprecated in iOS 5.0
: Sent to the view controller before the second half of the user interface rotates. (Deprecated. Use the one-step rotation technique instead. See the `willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.)

Responding to Containment Events

– `willMoveToParentViewController:` (page 78)
: Called just before the view controller is added or removed from a container view controller.

– `didMoveToParentViewController:` (page 46)
: Called after the view controller is added or removed from a container view controller.

Adding Editing Behaviors to Your View Controller

`editing` (page 24) *property*
: A Boolean value indicating whether the view controller currently allows the user to edit the view contents.

– `setEditing:animated:` (page 65)
: Sets whether the view controller shows an editable view.

Managing State Restoration

`restorationIdentifier` (page 34) *property*
The identifier that determines whether the view controller supports state restoration.

`restorationClass` (page 33) *property*
The class responsible for recreating this view controller when restoring the app's state.

– `encodeRestorableStateWithCoder:` (page 51)
Encodes state-related information for the view controller.

– `decodeRestorableStateWithCoder:` (page 46)
Decodes and restores state-related information for the view controller.

– `applicationFinishedRestoringState` (page 43)
Called on restored view controllers after other object decoding is complete.

Presenting Another View Controller's Content

– `presentViewController:animated:completion:` (page 61)
Presents a view controller.

– `dismissViewControllerAnimated:completion:` (page 49)
Dismisses the view controller that was presented by the receiver.

`modalTransitionStyle` (page 28) *property*
The transition style to use when presenting the receiver.

`modalPresentationStyle` (page 27) *property*
The presentation style for modally presented view controllers.

`definesPresentationContext` (page 23) *property*
A Boolean value that indicates whether this view controller's view is covered when the view controller or one of its descendants presents a view controller.

`transitioningDelegate` (page 40) *property*
The delegate object that provides transition animator and interactive controller objects.

`providesPresentationContextTransitionStyle` (page 33) *property*
A Boolean value that indicates whether the view controller defines the transition style for view controllers it presents.

– `disablesAutomaticKeyboardDismissal` (page 49)
Returns a Boolean indicating whether the current input view is dismissed automatically when changing controls.

– `dismissModalViewControllerAnimated:` (page 87) Deprecated in iOS 6.0

Dismisses the view controller that was presented by the receiver. (Deprecated. Use `dismissViewControllerAnimated:completion:` (page 49) instead.)

– `presentModalViewController:animated:` (page 88) Deprecated in iOS 6.0

Presents a modal view managed by the given view controller to the user. (Deprecated. Use `presentViewController:animated:completion:` (page 61) instead.)

Getting Other Related View Controllers

`presentingViewController` (page 32) *property*

The view controller that presented this view controller. (read-only)

`presentedViewController` (page 32) *property*

The view controller that is presented by this view controller, or one of its ancestors in the view controller hierarchy. (read-only)

`parentViewController` (page 31) *property*

The parent view controller of the recipient. (read-only)

`navigationController` (page 28) *property*

The nearest ancestor in the view controller hierarchy that is a navigation controller. (read-only)

`splitViewController` (page 35) *property*

The nearest ancestor in the view controller hierarchy that is a split view controller. (read-only)

`tabBarController` (page 36) *property*

The nearest ancestor in the view controller hierarchy that is a tab bar controller. (read-only)

`searchDisplayController` (page 34) *property*

The search display controller associated with the view controller. (read-only)

`modalViewController` (page 86) *property* Deprecated in iOS 6.0

The controller for the active presented view—that is, the view that is temporarily displayed on top of the view managed by the receiver. (read-only) (Deprecated. Use `presentedViewController` (page 32) instead.)

Managing Child View Controllers in a Custom Container

`childViewControllers` (page 23) *property*

An array of the view controllers that are the children of the receiver in the view controller hierarchy. (read-only)

– `addChildViewController:` (page 42)
  Adds the given view controller as a child.

– `removeFromParentViewController` (page 62)
  Removes the receiver from its parent in the view controller hierarchy.

– `shouldAutomaticallyForwardRotationMethods` (page 67)
  Returns a Boolean value indicating whether rotation methods are forwarded to child view controllers.

– `shouldAutomaticallyForwardAppearanceMethods` (page 66)
  Returns a Boolean value indicating whether appearance methods are forwarded to child view controllers.

– `transitionFromViewController:toViewController:duration:options:animations:completion:` (page 70)
  Transitions between two of the view controller's child view controllers.

– `beginAppearanceTransition:animated:` (page 43)
  Tells a child controller its appearance is about to change.

– `endAppearanceTransition` (page 52)
  Tells a child controller its appearance has changed.

– `viewControllerForUnwindSegueAction:fromViewController:withSender:` (page 72)
  Called when an unwind segue action wants to search a container's children for a view controller to handle the unwind action.

– `segueForUnwindingToViewController:fromViewController:identifier:` (page 64)
  Called when an unwind segue action needs to transition between two view controllers.

– `automaticallyForwardAppearanceAndRotationMethodsToChildViewControllers` (page 86) Deprecated in iOS 6.0
  Returns a Boolean value that indicates whether appearance and rotation methods are forwarded. (Deprecated. Use `shouldAutomaticallyForwardRotationMethods` (page 67) and `shouldAutomaticallyForwardAppearanceMethods` (page 66) instead.)

Configuring a Navigation Interface

`navigationItem` (page 29) *property*
  The navigation item used to represent the view controller in a parent's navigation bar. (read-only)

– `editButtonItem` (page 50)
  Returns a bar button item that toggles its title and associated state between Edit and Done.

`hidesBottomBarWhenPushed` (page 25) *property*
  A Boolean value indicating whether the toolbar at the bottom of the screen is hidden when the view controller is pushed on to a navigation controller.

– `setToolbarItems:animated:` (page 66)
Sets the toolbar items to be displayed along with the view controller.

`toolbarItems` (page 37) *property*
The toolbar items associated with the view controller.

Configuring Tab Bar Items

`tabBarItem` (page 36) *property*
The tab bar item that represents the view controller when added to a tab bar controller.

Configuring Display in a Popover Controller

`modalInPopover` (page 26) *property*
A Boolean value indicating whether the view controller should be presented modally by a popover.

`contentSizeForViewInPopover` (page 91) *property* Deprecated in iOS 7.0
The size of the view controller's view while displayed in a popover.

Properties

automaticallyAdjustsScrollViewInsets

Specifies whether or not the view controller should automatically adjust its scroll view insets.

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

Discussion
Default value is `YES`, which allows the view controller to adjust its scroll view insets in response to the screen areas consumed by the status bar, navigation bar, and toolbar or tab bar. Set to `NO` if you want to manage scroll view inset adjustments yourself, such as when there is more than one scroll view in the view hierarchy.

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewController.h`

bottomLayoutGuide

Indicates the lowest vertical extent for your onscreen content, for use with Auto Layout constraints. (read-only)

```
@property(nonatomic, readonly, retain) id<UILayoutSupport> bottomLayoutGuide
```

Discussion

The `bottomLayoutGuide` property comes into play when a view controller is frontmost onscreen. It indicates the lowest vertical extent for content that you don't want to appear behind a translucent or transparent UIKit bar (such as a tab bar or toolbar). This property implements the `UILayoutSupport` protocol and you can employ it as a constraint item when using the `NSLayoutConstraint` class.

The value of this property is, specifically, the value of the `length` property of the object returned when you query this property. This value is constrained by either the view controller or by its enclosing container view controller (such as a navigation or tab bar controller), as follows:

- A view controller **not within** a container view controller constrains this property to indicate the top of the tab bar or toolbar, if one of these is visible, or else to indicate the bottom edge of the view controller's view.
- A view controller **within** a container view controller does not set this property's value. Instead, the container view controller constrains the value to indicate the top of the tab bar or toolbar, if one of these is visible, or else to indicate the bottom edge of the view controller's view.

If a container view controller's toolbar or tab bar is visible and opaque, the container lays out the frontmost view controller's view so its bottom edge abuts the top of the bar. In this case, the value of this property is 0.

Query this property within your implementation of the `viewDidLayoutSubviews` (page 74) method.

When laying out a storyboard scene, the Bottom Layout Guide object is available in the Interface Builder outline view as a child of the View Controller object. Adding a bottom layout guide using Interface Builder provides backward layout compatibility to iOS 6.

As an example of how to programmatically use this property with Auto Layout, say you want to position a control such that its bottom edge is 20 points above the bottom layout guide. This scenario applies to any of the scenarios listed above. Use code similar to the following:

```
[button setTranslatesAutoresizingMaskIntoConstraints: NO];
id bottomGuide = myViewController.bottomLayoutGuide;
NSDictionary *viewsDictionary = NSDictionaryOfVariableBindings (button, bottomGuide);
[myViewController.view addConstraints:
    [NSLayoutConstraint constraintsWithVisualFormat: @"V: [button]-20-[bottomGuide]"
                                                    options: 0
```

```
                                                metrics: nil
                                                  views: viewsDictionary]
self.view layoutSubviews; // You must call this method here or the system raises
an exception
];
```

Important: If you define Auto Layout constraints in a storyboard file as well as programmatically, it is your responsibility to ensure the constraints do not conflict. If they do conflict, the system may throw a runtime exception.

To use a bottom layout guide without using constraints, obtain the guide's position relative to the bottom bound of the containing view. In the case of using a view controller subclass, obtain the numbers you need as follows:

```
- (void) viewDidLayoutSubviews {
    CGRect viewBounds = self.view.bounds;
    CGFloat bottomBarOffset = self.bottomLayoutGuide.length;
}
```

In the case of using a view subclass, obtain the numbers you need as follows:

```
- (void) layoutSubviews {
    [super layoutSubviews]; // You must call super here or the system raises an
exception
    CGRect bounds = self.bounds;
    CGFloat bottomBarOffset = myVCReference.bottomLayoutGuide.length;
}
```

For more guidance on using view controllers in iOS 7, read "Using View Controllers" in *iOS 7 UI Transition Guide* .

Availability
Available in iOS 7.0 and later.

See Also
`@property topLayoutGuide` (page 38)

Declared in
`UIViewController.h`

childViewControllers

An array of the view controllers that are the children of the receiver in the view controller hierarchy. (read-only)

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

Discussion
This property does not include any presented view controllers.

This property is only intended to be read by an implementation of a custom container view controller.

Availability
Available in iOS 5.0 and later.

Declared in
`UIViewController.h`

definesPresentationContext

A Boolean value that indicates whether this view controller's view is covered when the view controller or one of its descendants presents a view controller.

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

Discussion
When a view controller is presented, iOS starts with the presenting view controller and asks it if it wants to provide the presentation context. If the presenting view controller does not provide a context, then iOS asks the presenting view controller's parent view controller. iOS searches up through the view controller hierarchy until a view controller provides a presentation context. If no view controller offers to provide a context, the window's root view controller provides the presentation context.

If a view controller returns `YES`, then it provides a presentation context. The portion of the window covered by the view controller's view determines the size of the presented view controller's view. The default value for this property is `NO`.

Availability
Available in iOS 5.0 and later.

Declared in
`UIViewController.h`

edgesForExtendedLayout

Indicates the extended edges to use for the layout.

```
@property(nonatomic, assign) UIRectEdge edgesForExtendedLayout
```

Discussion
Default value is `UIRectEdgeAll`.

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewController.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 50) method. Set the `editing` (page 24) property to the initial state of your view. Use the `setEditing:animated:` (page 65) 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 65)
– `editButtonItem` (page 50)

Related Sample Code
BonjourWeb
iPhoneCoreDataRecipes

Declared in
`UIViewController.h`

extendedLayoutIncludesOpaqueBars

Indicates whether or not the extended layout includes opaque bars.

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

Discussion

Default value is NO.

Note: Bars are translucent by default in iOS 7.0

Availability

Available in iOS 7.0 and later.

Declared in

`UIViewController.h`

hidesBottomBarWhenPushed

A Boolean value indicating whether the toolbar at the bottom of the screen is hidden when the view controller is pushed on to a navigation controller.

```
@property(nonatomic) BOOL hidesBottomBarWhenPushed
```

Discussion

A view controller added as a child of a navigation controller can display an optional toolbar at the bottom of the screen. The value of this property on the topmost view controller determines whether the toolbar is visible. If the value of this property is YES, the toolbar is hidden. If the value of this property is NO, the bar is visible.

Availability

Available in iOS 2.0 and later.

Related Sample Code
iPhoneCoreDataRecipes

Declared in

`UINavigationController.h`

interfaceOrientation

Convenience property that provides the current orientation of the interface, meaningful only if the view controller is taking up the full screen. (read-only)

```
@property(nonatomic, readonly) UIInterfaceOrientation interfaceOrientation
```

Discussion

Do not use this property for informing layout decisions. Instead, use the `statusBarOrientation` property, described in *UIApplication Class Reference* .

The possible values for the `interfaceOrientation` property are described in the `UIInterfaceOrientation` enum.

Availability

Available in iOS 2.0 and later.

See Also

– `willRotateToInterfaceOrientation:duration:` (page 79)
– `didRotateFromInterfaceOrientation:` (page 48)

Related Sample Code

Accessory
CryptoExercise
GenericKeychain
GeocoderDemo
iPhoneCoreDataRecipes

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`

modalPresentationCapturesStatusBarAppearance

Specifies whether a view controller, presented non—fullscreen, takes over control of status bar appearance from the presenting view controller.

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

Discussion

Default value is `NO`.

When you present a view controller by calling the `presentViewController:animated:completion:` (page 61) method, status bar appearance control is transferred from the presenting to the presented view controller only if the presented controller's `modalPresentationStyle` (page 27) value is `UIModalPresentationFullScreen` (page 81). By setting this property to `YES`, you specify the presented view controller controls status bar appearance, even though presented non–fullscreen.

The system ignores this property's value for a view controller presented fullscreen.

Availability

Available in iOS 7.0 and later.

Declared in

`UIViewController.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 onscreen. On iPhone and iPod touch, modal view controllers are always presented full-screen, but on iPad there are several different presentation options. For a list of possible presentation styles, and their compatibility with the available transition styles, see the “`Modal Presentation Styles`” (page 81) constant descriptions.

Availability

Available in iOS 3.2 and later.

Related Sample Code

AVLoupe

AVMovieExporter

AVPlayerDemo

Real-time Video Processing Using AVPlayerItemVideoOutput

Using UIImagePickerController to Select Pictures and Take Photos

Declared in
`UIViewController.h`

modalTransitionStyle

The transition style to use when presenting the receiver.

```
@property(nonatomic, assign) UIModalTransitionStyle modalTransitionStyle
```

Discussion
This property determines how the view controller's is animated onscreen when it is presented using the `presentViewController:animated:completion:` (page 61) method. To change the transition type, you must set this property before presenting the view controller. The default value for this property is `UIModalTransitionStyleCoverVertical` (page 80).

For a list of possible transition styles, and their compatibility with the available presentation styles, see the `"Presentation Transition Styles"` (page 80) constant descriptions.

Availability
Available in iOS 3.0 and later.

See Also
– `presentViewController:animated:completion:` (page 61)

Related Sample Code
AddMusic
PhotosByLocation

Declared in
`UIViewController.h`

navigationController

The nearest ancestor in the view controller hierarchy that is a navigation controller. (read-only)

```
@property(nonatomic, readonly, retain) UINavigationController *navigationController
```

Discussion
If the receiver or one of its ancestors is a child of a navigation controller, this property contains the owning navigation controller. This property is `nil` if the view controller is not embedded inside a navigation controller.

Availability
Available in iOS 2.0 and later.

See Also
`@property splitViewController` (page 35)
`@property tabBarController` (page 36)

Related Sample Code
AirDrop Examples

EADemo

MVCNetworking

NavBar

UICatalog

Declared in
`UINavigationController.h`

navigationItem

The navigation item used to represent the view controller in a parent's navigation bar. (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 controller. The first time the property is accessed, the `UINavigationItem` object is created. Therefore, you shouldn't access this property if you are not using a navigation controller to display the view controller. To ensure the navigation item is configured, you can either override this property and add code to create the bar button items when first accessed or create the items in your view controller's initialization code.

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.

The default behavior is to create a navigation item that displays the view controller's title.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb

iPhoneCoreDataRecipes

ListAdder

MVCNetworking
NavBar

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 52)
`@property nibName` (page 30)

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 you use a nib file to store your view controller's view, it is recommended that you specify that nib file explicitly when initializing your view controller. However, if you do not specify a nib name, and do not override the `loadView` method in your custom subclass, the view controller searches for a nib file using other means. Specifically, it looks for a nib file with an appropriate name (without the `.nib` extension) and loads that nib file whenever its view is requested. Specifically, it looks (in order) for a nib file with one of the following names:

1. If the view controller class name ends with the word "Controller", as in `MyViewController`, it looks for a nib file whose name matches the class name without the word "Controller", as in `MyView.nib`.
2. It looks for a nib file whose name matches the name of the view controller class. For example, if the class name is `MyViewController`, it looks for a `MyViewController.nib` file.

Note: Nib names that include a platform-specific identifier such as `~iphone` or `~ipad` are loaded only on a device of the corresponding type. For example, a nib name of `MyViewController~ipad.nib` is loaded only on iPad. If your app supports both platform types, you must provide versions of your nib files for each platform.

Availability
Available in iOS 2.0 and later.

See Also
– `initWithNibName:bundle:` (page 52)
`@property nibBundle` (page 30)

Declared in
`UIViewController.h`

parentViewController

The parent view controller of the recipient. (read-only)

```
@property(nonatomic, readonly) UIViewController *parentViewController
```

Discussion
If the recipient is a child of a container view controller, this property holds the view controller it is contained in. If the recipient has no parent, the value in this property is `nil`.

Prior to iOS 5.0, if a view did not have a parent view controller and was being presented, the presenting view controller would be returned. On iOS 5, this behavior no longer occurs. Instead, use the `presentingViewController` (page 32) property to access the presenting view controller.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AdvancedURLConnections

AVPlayerDemo

Declared in
`UIViewController.h`

preferredContentSize

The preferred content size for any container view that is laying out a child view controller.

```
@property(nonatomic) CGSize preferredContentSize
```

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewController.h`

presentedViewController

The view controller that is presented by this view controller, or one of its ancestors in the view controller hierarchy. (read-only)

```
@property(nonatomic, readonly) UIViewController *presentedViewController
```

Availability
Available in iOS 5.0 and later.

Related Sample Code
AlternateViews

Declared in
`UIViewController.h`

presentingViewController

The view controller that presented this view controller. (read-only)

```
@property(nonatomic, readonly) UIViewController *presentingViewController
```

Discussion
If the view controller that received this message is presented by another view controller, this property holds the view controller that is presenting it. If the view controller is not presented, but one of its ancestors is being presented, this property holds the view controller presenting the nearest ancestor. If neither the view controller nor any of its ancestors are being presented, this property holds `nil`.

Availability
Available in iOS 5.0 and later.

Related Sample Code
AVPlayerDemo

Declared in
`UIViewController.h`

providesPresentationContextTransitionStyle

A Boolean value that indicates whether the view controller defines the transition style for view controllers it presents.

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

Discussion
When a view controller provides a presentation context to a presented view controller, it can choose to override the transition style of the presented view controller and substitute its own. If the value of this property is `YES`, then its own modal transition style is used whenever it provides the context for a presented view controller. If the value of this property is `NO`, then the modal transition style of the presented view controller's modal transition style is used. The default value is `NO`.

Availability
Available in iOS 5.0 and later.

See Also
`@property definesPresentationContext` (page 23)

Declared in
`UIViewController.h`

restorationClass

The class responsible for recreating this view controller when restoring the app's state.

```
@property(nonatomic, readwrite, assign) Class<UIViewControllerRestoration> *restorationClass
```

Discussion
If a view controller has an associated restoration class, the `viewControllerWithRestorationIdentifierPath:coder:` method of that class is called during state restoration. That method is responsible for returning the view controller object that matches the indicated view controller. If you do not specify a restoration class for your view controller, the state restoration engine asks your app delegate to provide the view controller object instead.

The restoration class must conform to the `UIViewControllerRestoration` protocol.

Availability
Available in iOS 6.0 and later.

Declared in
`UIViewController.h`

restorationIdentifier

The identifier that determines whether the view controller supports state restoration.

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

Discussion

This property indicates whether the view controller and its contents should be preserved and is used to identify the view controller during the restoration process. The value of this property is `nil` by default, which indicates that the view controller should not be saved. Assigning a string object to the property lets the system know that the view controller should be saved. In addition, the contents of the string are your way to identify the purpose of the view controller.

During subsequent launches, UIKit asks your app for help in recreating the view controllers that were installed the last time your app ran. When it asks for a specific view controller, UIKit provides your app with this restoration identifier and the restoration identifiers of any parent view controllers in the view controller hierarchy. Your app must use this information to create or locate the appropriate view controller object.

Important: Simply setting the value of this property is not enough to ensure that the view controller is preserved and restored. All parent view controllers must also have a restoration identifier. For more information about the preservation and restoration process, see *View Controller Programming Guide for iOS* .

Availability

Available in iOS 6.0 and later.

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 nearest ancestor in the view controller hierarchy that is a split view controller. (read-only)

```
@property(nonatomic, readonly, retain) UISplitViewController *splitViewController
```

Discussion
If the receiver or one of its ancestors is a child of a split view controller, this property contains the owning split view controller. This property is `nil` if the view controller is not embedded inside a split view controller.

Availability
Available in iOS 3.2 and later.

See Also
`@property navigationController` (page 28)
`@property tabBarController` (page 36)

Related Sample Code
MultipleDetailViews

Declared in
`UISplitViewController.h`

storyboard

The storyboard from which the view controller originated. (read-only)

```
@property(nonatomic, readonly, retain) UIStoryboard *storyboard
```

Discussion
If the view controller was not instantiated from a storyboard, this property is `nil`.

Availability
Available in iOS 5.0 and later.

Declared in
`UIViewController.h`

tabBarController

The nearest ancestor in the view controller hierarchy that is a tab bar controller. (read-only)

```
@property(nonatomic, readonly, retain) UITabBarController *tabBarController
```

Discussion

If the receiver or one of its ancestors is a child of a tab bar controller, this property contains the owning tab bar controller. This property is `nil` if the view controller is not embedded inside a tab bar controller.

Availability

Available in iOS 2.0 and later.

See Also

`@property navigationController` (page 28)
`@property splitViewController` (page 35)

Related Sample Code
AVPlayerDemo
XMLPerformance

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

This is a unique instance of `UITabBarItem` created to represent the view controller when it is a child of a tab bar controller. The first time the property is accessed, the `UITabBarItem` is created. Therefore, you shouldn't access this property if you are not using a tab bar controller to display the view controller. To ensure the tab bar item is configured, you can either override this property and add code to create the bar button items when first accessed or create the items in your view controller's initialization code.

The default value is a tab bar item that displays the view controller's title.

Availability

Available in iOS 2.0 and later.

Related Sample Code
GeocoderDemo

Declared in
`UITabBarController.h`

title

A localized string that represents the view 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.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb
CoreBluetooth Temperature Sensor
GenericKeychain
GeocoderDemo
UICatalog

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 66)

Related Sample Code

AVPlayerDemo

StitchedStreamPlayer

UICatalog

Declared in

`UINavigationController.h`

topLayoutGuide

Indicates the highest vertical extent for your onscreen content, for use with Auto Layout constraints. (read-only)

```
@property(nonatomic, readonly, retain) id<UILayoutSupport> topLayoutGuide
```

Discussion

The `topLayoutGuide` property comes into play when a view controller is frontmost onscreen. It indicates the highest vertical extent for content that you don't want to appear behind a translucent or transparent UIKit bar (such as a status or navigation bar). This property implements the `UILayoutSupport` protocol and you can employ it as a constraint item when using the `NSLayoutConstraint` class.

The value of this property is, specifically, the value of the `length` property of the object returned when you query this property. This value is constrained by either the view controller or by its enclosing container view controller (such as a navigation or tab bar controller), as follows:

- A view controller **not within** a container view controller constrains this property to indicate the bottom of the status bar, if visible, or else to indicate the top edge of the view controller's view.
- A view controller **within** a container view controller does not set this property's value. Instead, the container view controller constrains the value to indicate:
  - The bottom of the navigation bar, if a navigation bar is visible
  - The bottom of the status bar, if only a status bar is visible
  - The top edge of the view controller’s view, if neither a status bar nor navigation bar is visible

If a container navigation controller’s navigation bar is visible and opaque, the navigation controller lays out the frontmost view controller’s view so its top edge abuts the bottom of the navigation bar. In this case, the value of this property is `0`.

Query this property within your implementation of the `viewDidLayoutSubviews` (page 74) method.

When laying out a storyboard scene, the Top Layout Guide object is available in the Interface Builder outline view as a child of the View Controller object. Adding a top layout guide using Interface Builder provides backward compatibility to iOS 6.

As an example of how to programmatically use this property with Auto Layout, say you want to position a control such that its top edge is 20 points below the top layout guide. This scenario applies to any of the scenarios listed above. Use code similar to the following:

```
[button setTranslatesAutoresizingMaskIntoConstraints: NO];
id topGuide = myViewController.topLayoutGuide;
NSDictionary *viewsDictionary = NSDictionaryOfVariableBindings (button, topGuide);
[myViewController.view addConstraints:
    [NSLayoutConstraint constraintsWithVisualFormat: @"V: [topGuide]-20-[button]"
                                            options: 0
                                            metrics: nil
                                              views: viewsDictionary]
self.view layoutSubviews; // You must call this method here or the system raises
an exception
];
```

Important: If you define Auto Layout constraints in a storyboard file as well as programmatically, it is your responsibility to ensure the constraints do not conflict. If they do conflict, the system may throw a runtime exception.

To use a top layout guide without using constraints, obtain the guide's position relative to the top bound of the containing view. In the case of using a view controller subclass, obtain the numbers you need as follows:

```
- (void) viewDidLayoutSubviews {
    CGRect viewBounds = self.view.bounds;
    CGFloat topBarOffset = self.topLayoutGuide.length;
}
```

In the case of using a view subclass, obtain the numbers you need as follows:

```
- (void) layoutSubviews {
    [super layoutSubviews]; // You must call super here or the system raises an
exception
    CGRect bounds = self.bounds;
    CGFloat topBarOffset = myVCReference.topLayoutGuide.length;
}
```

For more guidance on using view controllers in iOS 7, read "Using View Controllers" in *iOS 7 UI Transition Guide* .

Availability
Available in iOS 7.0 and later.

See Also
`@property bottomLayoutGuide` (page 21)

Declared in
`UIViewController.h`

transitioningDelegate

The delegate object that provides transition animator and interactive controller objects.

```
@property(nonatomic, assign) id<UIViewControllerTransitioningDelegate> transitioningDelegate
```

Discussion
To support custom or interactive transitions in a view controller, assign a delegate object to this property. Your delegate, which must conform to the `UIViewControllerTransitioningDelegate` protocol, is then responsible for providing the objects needed to manage transitions.

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewController.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. 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 56) method and returns the resulting view.

Each view controller object is the sole owner of its view. You must not associate the same view object with multiple view controller objects. The only exception to this rule is that a container view controller implementation may add this view as a subview in its own view hierarchy. Before adding the subview, the container must first call its `addChildViewController:` (page 42) method to create a parent-child relationship between the two view controller objects.

Because accessing this property can cause the view to be loaded automatically, you can use the `isViewLoaded` method 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 can automatically set this property to `nil` during low-memory conditions and also when the view controller itself is finally released.

For more information about how a view controller loads and unloads its view, see "Resource Management in View Controllers".

Availability
Available in iOS 2.0 and later.

See Also
– `loadView` (page 56)
– `isViewLoaded` (page 56)
– `viewDidLoad` (page 74)

Related Sample Code
iAdInterstitialSuite

MoviePlayer

SimpleNetworkStreams

UICatalog

UnwindSegue

Declared in
`UIViewController.h`

Class Methods

attemptRotationToDeviceOrientation

Attempts to rotate all windows to the orientation of the device.

`+ (void)attemptRotationToDeviceOrientation`

Discussion
Some view controllers may want to use app-specific conditions to determine what interface orientations are supported. If your view controller does this, when those conditions change, your app should call this class method. The system immediately attempts to rotate to the new orientation.

Availability
Available in iOS 5.0 and later.

See Also
`– supportedInterfaceOrientations` (page 69)

Declared in
`UIViewController.h`

Instance Methods

addChildViewController:

Adds the given view controller as a child.

```
– (void)addChildViewController:(UIViewController *)childController
```

Parameters
`childController`
The view controller to be added as a child.

Discussion
If the new child view controller is already the child of a container view controller, it is removed from that container before being added.

This method is only intended to be called by an implementation of a custom container view controller. If you override this method, you must call `super` in your implementation.

Availability
Available in iOS 5.0 and later.

Related Sample Code
State Restoration of Child View Controllers

Declared in
`UIViewController.h`

applicationFinishedRestoringState

Called on restored view controllers after other object decoding is complete.

```
- (void)applicationFinishedRestoringState
```

Discussion
After other object decoding has completed, the system calls this method. This allows a view controller to complete setup after other state restoration, relying on the system to ensure that the states of all objects from the restoration archive have been decoded.

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewController.h`

beginAppearanceTransition:animated:

Tells a child controller its appearance is about to change.

```
- (void)beginAppearanceTransition:(BOOL)isAppearing animated:(BOOL)animated
```

Parameters
`isAppearing`
: `YES` if the child view controller's view is about to be added to the view hierarchy, `NO` if it is being removed.

`animated`
: If `YES`, the transition is being animated.

Discussion
If you are implementing a custom container controller, use this method to tell the child that its views are about to appear or disappear. Do not invoke `viewWillAppear:` (page 75), `viewWillDisappear:` (page 76), `viewDidAppear:` (page 73), or `viewDidDisappear:` (page 73) directly.

Availability
Available in iOS 5.0 and later.

See Also
– `endAppearanceTransition` (page 52)

Declared in
`UIViewController.h`

canPerformUnwindSegueAction:fromViewController:withSender:

Called on a view controller to determine whether it wants to respond to an unwind action.

```
- (BOOL)canPerformUnwindSegueAction:(SEL)action fromViewController:(UIViewController
 *)fromViewController withSender:(id)sender
```

Parameters

`action`
    The action the unwind action wants to invoke on your view controller.

`fromViewController`
    The view controller that initiated the unwind action.

`sender`
    The object that triggered the action.

Return Value

`YES` if the view controller wants to handle the unwind action, otherwise `NO`.

Discussion

This method is called when the system is attempting to find a view controller to handle an unwind action. Your custom view controller should implement this method to tell the system that it wants to support an unwind action.

If your view controller returns `YES`, then a series of steps are performed:

1. The target view controller's container invokes its `segueForUnwindingToViewController:fromViewController:identifier:` (page 64) method to create a segue to unwind the view controller state.
2. The action method is called on the target view controller.
3. The segue is triggered to perform the segue.

new segue is created to transition from the initiating view controller to your view controller. After the segue is created, your action is called and then the segue is invoked,

Availability

Available in iOS 6.0 and later.

Declared in
`UIViewController.h`

childViewControllerForStatusBarHidden

Called when the system needs the view controller to use for determining status bar hidden/unhidden state.

```
- (UIViewController *)childViewControllerForStatusBarHidden
```

Return Value
The view controller whose status bar hidden/unhidden status should be used. Default return value is `nil`.

Discussion
Implement this method to specify which child view controller you want to control the status bar hidden/unhidden state. If you return `nil` or do not implement this method, the status bar hidden/unhidden state for `self` is used.

If you change the return value from this method, call the `setNeedsStatusBarAppearanceUpdate` (page 65) method.

Availability
Available in iOS 7.0 and later.

See Also
– `preferredStatusBarUpdateAnimation` (page 59)
– `prefersStatusBarHidden` (page 60)

Declared in
`UIViewController.h`

childViewControllerForStatusBarStyle

Called when the system needs the view controller to use for determining status bar style.

```
- (UIViewController *)childViewControllerForStatusBarStyle
```

Return Value
The view controller whose status bar style should be used.

Discussion
Implement this method to return a child view controller or `nil`. If you return a non-nil value, that view controller's status bar style is used. If you return `nil`, the status bar style for `self` is used. If the return value from this method changes, call the `setNeedsStatusBarAppearanceUpdate` (page 65) method.

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewController.h`

decodeRestorableStateWithCoder:

Decodes and restores state-related information for the view controller.

```
- (void)decodeRestorableStateWithCoder:(NSCoder *)coder
```

Parameters
`coder`
The coder object to use to decode the state of the view.

Discussion
Do not call this method directly. The system calls this method during the state restoration process so that you can restore your view controller to its previous state.

If your app supports state restoration, override this method for any view controllers for which you also overrode the `encodeRestorableStateWithCoder:` method. Your implementation of this method should use any saved state information to restore the view controller to its previous configuration. If your `encodeRestorableStateWithCoder:` method called `super`, this method should similarly call `super` at some point in its implementation.

Availability
Available in iOS 6.0 and later.

See Also
– `encodeRestorableStateWithCoder:` (page 51)

Declared in
`UIViewController.h`

didMoveToParentViewController:

Called after the view controller is added or removed from a container view controller.

```
- (void)didMoveToParentViewController:(UIViewController *)parent
```

Parameters

`parent`

The parent view controller, or `nil` if there is no parent.

Discussion

Your view controller can override this method when it wants to react to being added to a container.

If you are implementing your own container view controller, it must call the `didMoveToParentViewController:` method of the child view controller after the transition to the new controller is complete or, if there is no transition, immediately after calling the `addChildViewController:` method.

The `removeFromParentViewController` method automatically calls the `didMoveToParentViewController:` method of the child view controller after it removes the child.

Availability

Available in iOS 5.0 and later.

See Also

– `willMoveToParentViewController:` (page 78)
– `shouldAutomaticallyForwardRotationMethods` (page 67)
– `shouldAutomaticallyForwardAppearanceMethods` (page 66)

Related Sample Code
State Restoration of Child View Controllers

Declared in

`UIViewController.h`

didReceiveMemoryWarning

Sent to the view controller when the app receives a memory warning.

```
- (void)didReceiveMemoryWarning
```

Discussion

Your app never calls this method directly. Instead, this method is called when the system determines that the amount of available memory is low.

You can override this method to release any additional memory used by your view controller. If you do, your implementation of this method must call the `super` implementation at some point.

Availability

Available in iOS 2.0 and later.

Related Sample Code
AddMusic

GKAchievements

iPhoneMultichannelMixerTest

PhotosByLocation

Regions

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. For possible values, see `UIInterfaceOrientation`.

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 25) property is already set to the new orientation. Your implementation of this method must call `super` at some point during its execution.

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 79)
– `willAnimateRotationToInterfaceOrientation:duration:` (page 77)

Declared in
`UIViewController.h`

disablesAutomaticKeyboardDismissal

Returns a Boolean indicating whether the current input view is dismissed automatically when changing controls.

```
- (BOOL)disablesAutomaticKeyboardDismissal
```

Return Value
`YES` to prevent the dismissal of the input view or `NO` if the input view may be dismissed.

Discussion
Override this method in a subclass to allow or disallow the dismissal of the current input view (usually the system keyboard) when changing from a control that wants the input view to one that does not. Under normal circumstances, when the user taps a control that requires an input view, the system automatically displays that view. Tapping in a control that does not want an input view subsequently causes the current input view to be dismissed but may not in all cases. You can override this method in those outstanding cases to allow the input view to be dismissed or use this method to prevent the view from being dismissed in other cases.

The default implementation of this method returns `YES` when the modal presentation style of the view controller is set to `UIModalPresentationFormSheet` (page 81) and returns `NO` for other presentation styles. Thus, the system normally does not allow the keyboard to be dismissed for modal forms.

Availability
Available in iOS 4.3 and later.

Declared in
`UIViewController.h`

dismissViewControllerAnimated:completion:

Dismisses the view controller that was presented by the receiver.

```
- (void)dismissViewControllerAnimated:(BOOL)flag completion:(void (^)(void))completion
```

Parameters
`flag`
Pass `YES` to animate the transition.

`completion`
A block called after the view controller has been dismissed.

Discussion
The presenting view controller is responsible for dismissing the view controller it presented. If you call this method on the presented view controller itself, it automatically forwards the message to the presenting view controller.

If you present several view controllers in succession, thus building a stack of presented 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 presented view controller, get the value in the `presentedViewController` (page 32) property before calling this method.

The completion handler is called after the `viewDidDisappear:` (page 73) method is called on the presented view controller.

Availability
Available in iOS 5.0 and later.

Related Sample Code
AVPlayerDemo

MultipeerGroupChat

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 29) property is set to the returned object, the associated navigation bar displays an Edit button if `editing` (page 24) is `NO` and a Done button if `editing` (page 24) is `YES`. The default button action invokes the `setEditing:animated:` (page 65) method.

Availability
Available in iOS 2.0 and later.

See Also
`@property editing` (page 24)
`– setEditing:animated:` (page 65)

Declared in
`UIViewController.h`

encodeRestorableStateWithCoder:

Encodes state-related information for the view controller.

```
- (void)encodeRestorableStateWithCoder:(NSCoder *)coder
```

Parameters

`coder`

The coder object to use to encode the state of the view controller.

Discussion

Do not call this method directly. The system calls this method during the state preservation process to give your view controller or view-controller subclass a chance to save state-related information.

When deciding what data to save, write the smallest amount of data needed to restore the view controller to its current configuration. The information you save should be data that you could not easily recreate, such as the user's current selection. You might also save references to any data objects that the view controller was using, but never write the data objects themselves to the coder. Instead, store enough information so that you can retrieve the data objects from your app's main data structures again.

Important: This method is not a substitute for saving your app's data structures persistently to disk. You should continue to save your app's actual data to iCloud or the local file system using existing techniques. This method is intended only for saving configuration state or other information related to your app's user interface. You should consider any data you write to the coder as purgeable and be prepared for it to be unavailable during subsequent launches.

It is strongly recommended that you call `super` at some point during your implementation to give parent classes an opportunity to save information. A `UIViewController` object saves a reference to the presented view controller and to the storyboard (if any) that was used to create the view controller. The view controller also asks the views in its view hierarchy to save any relevant information. However, a view controller does not automatically save references to contained child view controllers. If you are implementing a custom container view controller, you must encode the child view controller objects yourself if you want them to be preserved.

Your implementation of this method can encode other restorable objects—views, view controllers, and objects that adopt the `UIStateRestoring` protocol—using the `encodeObject:forKey:` method of the provided coder object. Encoding a restorable object writes that object's restoration identifier to the coder. That identifier is then used during the decode process to locate the new version of the object. If the view or view controller defines its own own version of this method, that method is also called at some point so that the object can encode its own state.

For objects that are not restorable, encoding the object writes its data (and not a restoration identifier) to the archive. Such objects must adopt the `NSCoding` protocol. During decoding, the system creates a new object that is initialized with the data from the archive.

Availability
Available in iOS 6.0 and later.

See Also
– `decodeRestorableStateWithCoder:` (page 46)

Related Sample Code
State Restoration of Child View Controllers

Declared in
`UIViewController.h`

endAppearanceTransition

Tells a child controller its appearance has changed.

```
– (void)endAppearanceTransition
```

Discussion
If you are implementing a custom container controller, use this method to tell the child that the view transition is complete.

Availability
Available in iOS 5.0 and later.

See Also
– `beginAppearanceTransition:animated:` (page 43)

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 to associate with the view controller. The nib file name should not contain any leading path information. If you specify `nil`, the `nibName` (page 30) property is set to `nil`.

`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 this parameter is `nil`, the method uses the heuristics described below to locate the nib file.

Return Value
A newly initialized `UIViewController` object.

Discussion
This is the designated initializer for this class.

The nib file you specify is not loaded right away. It is loaded the first time the view controller's view is accessed. If you want to perform additional initialization after the nib file is loaded, override the `viewDidLoad` (page 74) method and perform your tasks there.

If you specify `nil` for the `nibName` parameter and you do not override the `loadView` (page 56) method, the view controller searches for a nib file using other means. See `nibName` (page 30).

If your app uses a storyboard to define a view controller and its associated views, your app never initializes objects of that class directly. Instead, view controllers are either instantiated by the storyboard—either automatically by iOS when a segue is triggered or programmatically when your app calls the storyboard object's `instantiateViewControllerWithIdentifier:` method. When instantiating a view controller from a storyboard, iOS initializes the new view controller by calling its `initWithCoder:` method instead. iOS automatically sets the `nibName` (page 30) property to a nib file stored inside the storyboard.

For more information about how a view controller loads its view, see "Resource Management in View Controllers".

Availability
Available in iOS 2.0 and later.

See Also
`@property nibName` (page 30)
`@property nibBundle` (page 30)
– `loadView` (page 56)

Related Sample Code
AdvancedURLConnections
GeocoderDemo
iPhoneCoreDataRecipes
PhotosByLocation
UICatalog

Declared in
`UIViewController.h`

isBeingDismissed

Returns a Boolean value that indicates whether the view controller is in the process of being dismissed by one of its ancestors.

```
- (BOOL)isBeingDismissed
```

Return Value
`YES` if the view controller was previously presented and is in the process of being dismissed by one of its ancestors, otherwise `NO`.

Discussion
This method returns `YES` only when called from inside the following methods:

- `viewWillDisappear:` (page 76)
- `viewDidDisappear:` (page 73)

Availability
Available in iOS 5.0 and later.

Declared in
`UIViewController.h`

isBeingPresented

Returns a Boolean value that indicates whether the view controller is in the process of being presented by one of its ancestors.

```
- (BOOL)isBeingPresented
```

Return Value
`YES` if the view controller is appearing because it was presented by another view controller, otherwise `NO`.

Discussion
This method returns `YES` only when called from inside the following methods:

- `viewWillAppear:` (page 75)
- `viewDidAppear:` (page 73)

Availability
Available in iOS 5.0 and later.

Declared in
`UIViewController.h`

isMovingFromParentViewController

Returns a Boolean value that indicates that the view controller is in the process of being removed from its parent.

```
- (BOOL)isMovingFromParentViewController
```

Return Value
`YES` if the view controller is disappearing because it was removed from a container view controller, otherwise `NO`.

Discussion
This method returns `YES` only when called from inside the following methods:

`viewWillDisappear:` (page 76)

`viewDidDisappear:` (page 73)

Availability
Available in iOS 5.0 and later.

Declared in
`UIViewController.h`

isMovingToParentViewController

Returns a Boolean value that indicates that the view controller is in the process of being added to a parent.

```
- (BOOL)isMovingToParentViewController
```

Return Value
`YES` if the view controller is appearing because it was added as a child of a container view controller, otherwise `NO`.

Discussion
This method returns `YES` only when called from inside the following methods:

`viewWillAppear:` (page 75)

`viewDidAppear:` (page 73)

Availability

Available in iOS 5.0 and later.

Declared in

`UIViewController.h`

isViewLoaded

Returns a Boolean value indicating whether the view is currently loaded into memory.

```
- (BOOL)isViewLoaded
```

Return Value

A Boolean value indicating whether the view is currently loaded into memory.

Discussion

Calling this method 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.

Related Sample Code

AdvancedURLConnections

ListAdder

MVCNetworking

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 its `view` (page 40) property is requested but is currently `nil`. This method loads or creates a view and assigns it to the `view` (page 40) property.

If the view controller has an associated nib file, this method loads the view from the nib file. A view controller has an associated nib file if the `nibName` (page 30) property returns a non-`nil` value, which occurs if the view controller was instantiated from a storyboard, if you explicitly assigned it a nib file using the `initWithNibName:bundle:` (page 52) method, or if iOS finds a nib file in the app bundle with a name based on the view controller's class name. If the view controller does not have an associated nib file, this method creates a plain `UIView` object instead.

If you use Interface Builder to create your views and initialize the view controller, you must not override this method.

You can override this method in order to create your views manually. If you choose to do so, assign the root view of your view hierarchy to the `view` (page 40) 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 74) method.

Availability
Available in iOS 2.0 and later.

See Also
`@property view` (page 40)
`@property nibName` (page 30)
– `viewDidLoad` (page 74)

Related Sample Code
iPhoneCoreDataRecipes

Declared in
`UIViewController.h`

performSegueWithIdentifier:sender:

Initiates the segue with the specified identifier from the view controller's storyboard file.

```
- (void)performSegueWithIdentifier:(NSString *)identifier sender:(id)sender
```

Parameters

`identifier`

The string that identifies the segue inside the storyboard file.

In Interface Builder, you can associate an identifier string with each segue using the inspector. This string is used only for locating the segue inside the storyboard. This is the string that you pass to this parameter.

This method throws an exception if there is no segue with the specified identifier.

`sender`

The object that you want to use to initiate the segue. This object is made available for informational purposes during the actual segue.

Discussion

Apps normally do not need to trigger segues directly. Instead, you configure an object in Interface Builder associated with the view controller, such as a control embedded in its view hierarchy, to trigger the segue. However, you can call this method to trigger a segue programmatically, perhaps in response to some action that cannot be specified in the storyboard resource file. For example, you might call it from a custom action handler used to process shake or accelerometer events.

The view controller that receives this message must have been loaded from a storyboard. If the view controller does not have an associated storyboard, perhaps because you allocated and initialized it yourself, this method throws an exception.

Availability

Available in iOS 5.0 and later.

See Also

– `prepareForSegue:sender:` (page 60)

Related Sample Code

MultipeerGroupChat

Declared in

`UIViewController.h`

preferredInterfaceOrientationForPresentation

Returns the interface orientation to use when presenting the view controller.

```
- (UIInterfaceOrientation)preferredInterfaceOrientationForPresentation
```

Return Value

The interface orientation with which to present the view controller.

Discussion

The system calls this method when presenting the view controller full screen. You implement this method when your view controller supports two or more orientations but the content appears best in one of those orientations.

If your view controller implements this method, then when presented, its view is shown in the preferred orientation (although it can later be rotated to another supported rotation). If you do not implement this method, the system presents the view controller using the current orientation of the status bar.

Availability

Available in iOS 6.0 and later.

Declared in

`UIViewController.h`

preferredStatusBarStyle

The preferred status bar style for the view controller.

```
- (UIStatusBarStyle)preferredStatusBarStyle
```

Return Value

A `UIStatusBarStyle` key indicating your preferred status bar style for the view controller.

Discussion

You can override the preferred status bar style for a view controller by implementing the `childViewControllerForStatusBarStyle` (page 45) method.

If the return value from this method changes, call the `setNeedsStatusBarAppearanceUpdate` (page 65) method.

Availability

Available in iOS 7.0 and later.

Declared in

`UIViewController.h`

preferredStatusBarUpdateAnimation

Specifies the animation style to use for hiding and showing the status bar for the view controller.

```
- (UIStatusBarAnimation)preferredStatusBarUpdateAnimation
```

Return Value

The style of status bar animation to use; one of the constants from the `UIStatusBarAnimation` enum. Default value is `UIStatusBarAnimationFade`.

Discussion

This property comes into play only when you actively change the status bar's show/hide state by changing the return value of the `prefersStatusBarHidden` (page 60) method.

Availability

Available in iOS 7.0 and later.

Declared in

`UIViewController.h`

prefersStatusBarHidden

Specifies whether the view controller prefers the status bar to be hidden or shown.

```
- (BOOL)prefersStatusBarHidden
```

Return Value

A Boolean value of `YES` specifies the status bar should be hidden. Default value is `NO`.

Discussion

If you change the return value for this method, call the `setNeedsStatusBarAppearanceUpdate` (page 65) method.

To specify that a child view controller should control preferred status bar hidden/unhidden state, implement the `childViewControllerForStatusBarHidden` (page 45) method.

Availability

Available in iOS 7.0 and later.

See Also

– `preferredStatusBarUpdateAnimation` (page 59)

Declared in

`UIViewController.h`

prepareForSegue:sender:

Notifies the view controller that a segue is about to be performed.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
```

Parameters

`segue`

The segue object containing information about the view controllers involved in the segue.

`sender`

The object that initiated the segue. You might use this parameter to perform different actions based on which control (or other object) initiated the segue.

Discussion

The default implementation of this method does nothing. Your view controller overrides this method when it needs to pass relevant data to the new view controller. The segue object describes the transition and includes references to both view controllers involved in the segue.

Because segues can be triggered from multiple sources, you can use the information in the `segue` and `sender` parameters to disambiguate between different logical paths in your app. For example, if the segue originated from a table view, the sender parameter would identify the table view cell that the user tapped. You could use that information to set the data on the destination view controller.

Availability

Available in iOS 5.0 and later.

Declared in

`UIViewController.h`

presentViewController:animated:completion:

Presents a view controller.

```
- (void)presentViewController:(UIViewController *)viewControllerToPresent
animated:(BOOL)flag completion:(void (^)(void))completion
```

Parameters

`viewControllerToPresent`

The view controller being presented.

`flag`

Pass `YES` to animate the presentation; otherwise, pass `NO`.

`completion`

A completion handler or `NULL`.

Discussion

On iPhone and iPod touch, the presented view is always full screen. On iPad, the presentation depends on the value in the `modalPresentationStyle` (page 27) property.

This method sets the `presentedViewController` (page 32) property to the specified view controller, resizes that view controller's view and then adds the view to the view hierarchy. The view is animated onscreen according to the transition style specified in the `modalTransitionStyle` (page 28) property of the presented view controller.

The completion handler is called after the `viewDidAppear:` method is called on the presented view controller.

Availability

Available in iOS 5.0 and later.

Related Sample Code

AirDrop Examples

AVPlayerDemo

MyImagePicker

PhotosByLocation

Declared in

`UIViewController.h`

removeFromParentViewController

Removes the receiver from its parent in the view controller hierarchy.

```
- (void)removeFromParentViewController
```

Discussion

This method is only intended to be called by an implementation of a custom container view controller. If you override this method, you must call `super` in your implementation.

Availability

Available in iOS 5.0 and later.

Declared in

`UIViewController.h`

rotatingFooterView

Returns the footer view to transition during an interface orientation change.

```
- (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. The default implementation returns `nil`.

Discussion
In most cases, the header view is the navigation bar and the footer view is the tab bar. If you are implementing this method in a custom view controller that has its own custom footer view, you can override this method to return that footer view. The view returned from this method should already be part of your view controller's view hierarchy.

You are responsible for adjusting the size and position of the returned view to match the target orientation. You would make such a change in your view controller's rotation methods, such as the `willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.

Availability
Available in iOS 2.0 and later.

See Also
– `rotatingHeaderView` (page 63)

Declared in
`UIViewController.h`

rotatingHeaderView

Returns the header view to transition during an interface orientation change.

```
- (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 you are implementing this method in a custom view controller that has its own custom header view, you can override this method to return that header view. The view returned from this method should already be part of your view controller's view hierarchy.

You are responsible for adjusting the size and position of the returned view to match the target orientation. You would make such a change in your view controller's rotation methods, such as the `willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.

Availability
Available in iOS 2.0 and later.

See Also
– `rotatingFooterView` (page 62)

Declared in
`UIViewController.h`

segueForUnwindingToViewController:fromViewController:identifier:

Called when an unwind segue action needs to transition between two view controllers.

```
- (UIStoryboardSegue *)segueForUnwindingToViewController:(UIViewController
*)toViewController fromViewController:(UIViewController *)fromViewController
identifier:(NSString *)identifier
```

Parameters

`toViewController`
The target view controller.

`fromViewController`
The view controller initiating the unwind action.

`identifier`
An identifier for the segue.

Return Value
A custom segue object that, when invoked, transitions between the two view controllers.

Discussion
If you implement a custom container view controller that also uses segue unwinding, you must override this method. Your method implementation should instantiate and return a custom segue object that performs whatever animation and other steps that are necessary to unwind the view controllers.

Availability
Available in iOS 6.0 and later.

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 29) property is set to the value returned by the `editButtonItem` (page 50) 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 24) is `YES` and a non-editable state if it is `NO`. This method should invoke super's implementation before updating its view.

Availability

Available in iOS 2.0 and later.

See Also

`@property editing` (page 24)
`– editButtonItem` (page 50)

Declared in

`UIViewController.h`

setNeedsStatusBarAppearanceUpdate

Indicates to the system that the view controller status bar attributes have changed.

```
- (void)setNeedsStatusBarAppearanceUpdate
```

Discussion

Call this method if the view controller's status bar attributes, such as hidden/unhidden status or style, change. If you call this method within an animation block, the changes are animated along with the rest of the animation block.

Availability
Available in iOS 7.0 and later.

See Also
– `preferredStatusBarStyle` (page 59)
– `prefersStatusBarHidden` (page 60)

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`

shouldAutomaticallyForwardAppearanceMethods

Returns a Boolean value indicating whether appearance methods are forwarded to child view controllers.

```
– (BOOL)shouldAutomaticallyForwardAppearanceMethods
```

Return Value
`YES` if appearance methods are forwarded or `NO` if they are not.

Discussion
This method is called to determine whether to automatically forward appearance-related containment callbacks to child view controllers.

The default implementation returns `YES`. Subclasses of the `UIViewController` class that implement containment logic may override this method to control how these methods are forwarded. If you override this method and return `NO`, you are responsible for telling the child when its views are going to appear or disappear. You do this by calling the child view controller's `beginAppearanceTransition:animated:` (page 43) and `endAppearanceTransition` (page 52) methods.

Availability
Available in iOS 6.0 and later.

Declared in
`UIViewController.h`

shouldAutomaticallyForwardRotationMethods

Returns a Boolean value indicating whether rotation methods are forwarded to child view controllers.

```
- (BOOL)shouldAutomaticallyForwardRotationMethods
```

Return Value
`YES` if rotation methods are forwarded or `NO` if they are not.

Discussion
This method is called to determine whether to automatically forward rotation-related containment callbacks to child view controllers.

The default implementation returns `YES`. Subclasses of the `UIViewController` class that implement containment logic may override this method to control how these methods are forwarded. If you override this method and return `NO`, you are responsible for forwarding the following methods to child view controllers at the appropriate times:

`willRotateToInterfaceOrientation:duration:` (page 79)

`willAnimateRotationToInterfaceOrientation:duration:` (page 77)

`didRotateFromInterfaceOrientation:` (page 48)

Availability
Available in iOS 6.0 and later.

Declared in
`UIViewController.h`

shouldAutorotate

Returns whether the view controller's contents should auto rotate.

```
- (BOOL)shouldAutorotate
```

Return Value
`YES` if the content should rotate, otherwise `NO`. Default value is `YES`.

Special Considerations
In iOS 5 and earlier, the default return value was `NO`.

Availability
Available in iOS 6.0 and later.

Related Sample Code
AlternateViews

AVLoupe

GLPaint

Declared in
`UIViewController.h`

shouldPerformSegueWithIdentifier:sender:

Determines whether the segue with the specified identifier should be triggered.

```
- (BOOL)shouldPerformSegueWithIdentifier:(NSString *)identifier sender:(id)sender
```

Parameters
`identifier`

The string that identifies the triggered segue.

In Interface Builder, you can associate an identifier string with each segue using the inspector. This string is used only for locating the segue inside the storyboard.

`sender`

The object that initiated the segue. This object is made available for informational purposes during the actual segue.

Return Value

This method should return `YES` if the segue should be executed, `NO` if it should be ignored.

Discussion

Your view controller overrides this method when it wants to control whether a segue should be performed. The default behavior is to permit all segues to be triggered.

Availability

Available in iOS 6.0 and later.

Declared in

`UIViewController.h`

supportedInterfaceOrientations

Returns all of the interface orientations that the view controller supports.

```
- (NSUInteger)supportedInterfaceOrientations
```

Return Value

A bit mask specifying which orientations are supported. See `UIInterfaceOrientationMask` for valid bit-mask values. The value returned by this method must not be `0`.

Discussion

When the user changes the device orientation, the system calls this method on the root view controller or the topmost presented view controller that fills the window. If the view controller supports the new orientation, the window and view controller are rotated to the new orientation. This method is only called if the view controller's `shouldAutorotate` (page 68) method returns `YES`.

Override this method to report all of the orientations that the view controller supports. The default values for a view controller's supported interface orientations is set to `UIInterfaceOrientationMaskAll` for the iPad idiom and `UIInterfaceOrientationMaskAllButUpsideDown` for the iPhone idiom.

The system intersects the view controller's supported orientations with the app's supported orientations (as determined by the Info.plist file or the app delegate's `application:supportedInterfaceOrientationsForWindow:` method) to determine whether to rotate.

Availability

Available in iOS 6.0 and later.

See Also

– `shouldAutorotate` (page 68)

Related Sample Code
AlternateViews

AVSimpleEditoriOS

Real-time Video Processing Using AVPlayerItemVideoOutput

Simple Gesture Recognizers

Declared in
`UIViewController.h`

transitionCoordinator

Returns a transition coordinator.

```
- (id<UIViewControllerTransitionCoordinator>)transitionCoordinator
```

Return Value
If called during an active view controller presentation or dismissal, returns a transition coordinator—a custom object that adopts the `UIViewControllerTransitionCoordinator` protocol. If called at other times, passes the request on to the parent view controller.

Discussion
You can override this method, but only custom container view controllers should ever have a need to do so.

Your override should typically first check if there is an appropriate transition coordinator to return, and, if there is, return it. If there is no appropriate coordinator to return, call `super`.

For more information on transition coordinators, see *UIViewControllerTransitionCoordinator Protocol Reference* .

Availability
Available in iOS 7.0 and later.

Declared in
`UIViewControllerTransitionCoordinator.h`

transitionFromViewController:toViewController:duration:options:animations: completion:

Transitions between two of the view controller's child view controllers.

```
- (void)transitionFromViewController:(UIViewController *)fromViewController
toViewController:(UIViewController *)toViewController
duration:(NSTimeInterval)duration options:(UIViewAnimationOptions)options
animations:(void (^)(void))animations completion:(void (^)(BOOL finished))completion
```

Parameters

`fromViewController`

A view controller whose view is currently visible in the parent's view hierarchy.

`toViewController`

A child view controller whose view is not currently in the view hierarchy.

`duration`

The total duration of the animations, in seconds. If you pass zero, the changes are made without animating them.

`options`

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

`animations`

A block object containing the changes to commit to the views. Here 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 to be called when the animation completes.

The block takes the following parameters:

finished

`YES` if the animation finished; `NO` if it was skipped.

Discussion

This method adds the second view controller's view to the view hierarchy and then performs the animations defined in your animations block. After the animation completes, it removes the first view controller's view from the view hierarchy.

This method is only intended to be called by an implementation of a custom container view controller. If you override this method, you must call `super` in your implementation.

Availability

Available in iOS 5.0 and later.

Declared in

`UIViewController.h`

updateViewConstraints

Called when the view controller's view needs to update its constraints.

```
- (void)updateViewConstraints
```

Discussion

You may override this method in a subclass in order to add constraints to the view or its subviews. If you override this method, your implementation must invoke super's implementation.

Availability

Available in iOS 6.0 and later.

Declared in

`UIViewController.h`

viewControllerForUnwindSegueAction:fromViewController:withSender:

Called when an unwind segue action wants to search a container's children for a view controller to handle the unwind action.

```
- (UIViewController *)viewControllerForUnwindSegueAction:(SEL)action
fromViewController:(UIViewController *)fromViewController withSender:(id)sender
```

Parameters

`action`
: The action that triggered the unwind action.

`fromViewController`
: The view controller that is the source of the unwinding action.

`sender`
: The object that initiated the action.

Return Value

The view controller that wants to handle the unwind action.

Discussion

A custom container view controller should override this method and use it to search its children for a view controller to handle the unwind action. It does this by invoking the `canPerformUnwindSegueAction:fromViewController:withSender:` (page 44) method on each child. If a view controller wants to handle the action, your method should return it. If none of the container's children want to handle the unwind action, invoke the super's implementation and return the result of that method.

Availability

Available in iOS 6.0 and later.

Declared in
`UIViewController.h`

viewDidAppear:

Notifies the view controller that its view was added to a view hierarchy.

```
- (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.

> **Note:** If a view controller is presented by a view controller inside of a popover, this method is not invoked on the presenting view controller after the presented controller is dismissed.

Availability
Available in iOS 2.0 and later.

See Also
– `viewWillAppear:` (page 75)

Related Sample Code
UIKit Dynamics Catalog

Declared in
`UIViewController.h`

viewDidDisappear:

Notifies the view controller that its view was removed from a view hierarchy.

```
- (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 76)

Declared in

`UIViewController.h`

viewDidLayoutSubviews

Called to notify the view controller that its view has just laid out its subviews.

```
- (void)viewDidLayoutSubviews
```

Discussion

When the bounds change for a view controller's view, the view adjusts the positions of its subviews and then the system calls this method. However, this method being called *does not* indicate that the individual layouts of the view's subviews have been adjusted. Each subview is responsible for adjusting its own layout.

Your view controller can override this method to make changes after the view lays out its subviews. The default implementation of this method does nothing.

Availability

Available in iOS 5.0 and later.

Related Sample Code

HeadsUpUI

iAdInterstitialSuite

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 view hierarchy into memory. This method is called regardless of whether the view hierarchy was loaded from a nib file or created programmatically in the `loadView` (page 56) method. You usually override this method to perform additional initialization on views that were loaded from nib files.

Availability

Available in iOS 2.0 and later.

See Also

`@property view` (page 40)
– `loadView` (page 56)

Related Sample Code

GeocoderDemo
iPhoneCoreDataRecipes
NavBar
UICatalog
UIKit Dynamics Catalog

Declared in

`UIViewController.h`

viewWillAppear:

Notifies the view controller that its view is about to be added to a view hierarchy.

```
- (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 added to a view hierarchy and before any animations are configured for showing the view. You can override this method to perform custom tasks associated with displaying 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 view hierarchies by a view controller, and the sequence of messages that occur, see "Responding to Display-Related Notifications".

Note: If a view controller is presented by a view controller inside of a popover, this method is not invoked on the presenting view controller after the presented controller is dismissed.

Availability
Available in iOS 2.0 and later.

See Also
– `viewDidAppear:` (page 73)

Related Sample Code
iPhoneCoreDataRecipes

Declared in
`UIViewController.h`

viewWillDisappear:

Notifies the view controller that its view is about to be removed from a view hierarchy.

```
- (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 a view hierarchy. This method is called before the view is actually removed 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 73) 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 73)

Declared in
`UIViewController.h`

viewWillLayoutSubviews

Called to notify the view controller that its view is about to layout its subviews.

```
- (void)viewWillLayoutSubviews
```

Discussion
When a view's bounds change, the view adjusts the position of its subviews. Your view controller can override this method to make changes before the view lays out its subviews. The default implementation of this method does nothing.

Availability
Available in iOS 5.0 and later.

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`.

`duration`
The duration of the pending rotation, measured in seconds.

Discussion
This method is called from within the animation block 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 25) property is already set to the new orientation, and the bounds of the view have been changed. 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 79)
– `didRotateFromInterfaceOrientation:` (page 48)

Declared in
`UIViewController.h`

willMoveToParentViewController:

Called just before the view controller is added or removed from a container view controller.

```
- (void)willMoveToParentViewController:(UIViewController *)parent
```

Parameters
`parent`
The parent view controller, or `nil` if there is no parent.

Discussion
Your view controller can override this method when it needs to know that it has been added to a container.

If you are implementing your own container view controller, it must call the `willMoveToParentViewController:` method of the child view controller before calling the `removeFromParentViewController` (page 62) method, passing in a parent value of `nil`.

When your custom container calls the `addChildViewController:` (page 42) method, it automatically calls the `willMoveToParentViewController:` (page 78) method of the view controller to be added as a child before adding it.

Availability
Available in iOS 5.0 and later.

See Also

– `didMoveToParentViewController:` (page 46)
– `shouldAutomaticallyForwardRotationMethods` (page 67)
– `shouldAutomaticallyForwardAppearanceMethods` (page 66)

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`.

`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 25) property still contains the view's original orientation. Your implementation of this method must call `super` at some point during its execution.

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 77)
– `didRotateFromInterfaceOrientation:` (page 48)

Declared in

`UIViewController.h`

Constants

Presentation Transition Styles

Transition styles available when presenting view controllers.

```
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 presented view underneath. On dismissal, the curled up page unfurls itself back on top of the presented view. A view presented using this transition is itself prevented from presenting any additional views.

This transition style is supported only if the parent view controller is presenting a full-screen view and you use the `UIModalPresentationFullScreen` (page 81) 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`.

Modal Presentation Styles

Modal presentation styles available when presenting view controllers.

```
typedef enum {
   UIModalPresentationFullScreen = 0,
   UIModalPresentationPageSheet,
   UIModalPresentationFormSheet,
   UIModalPresentationCurrentContext,
   UIModalPresentationCustom,
   UIModalPresentationNone
} UIModalPresentationStyle;
```

Constants

`UIModalPresentationFullScreen`

A view presentation style in which the presented view covers the screen, taking into account the value of the `wantsFullScreenLayout` (page 92) property.

Available in iOS 3.2 and later.

Declared in `UIViewController.h`.

`UIModalPresentationPageSheet`

A view presentation style in which 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`

A view presentation style in which the width and height of the presented view are smaller than those of the screen and the view is centered onscreen. If the device is in a landscape orientation and the keyboard is visible, the position of the view is adjusted upward so 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 same view presentation style used by the view's parent view controller.

When presenting a view controller in a popover, this presentation style is supported only if the transition style is `UIModalTransitionStyleCoverVertical` (page 80). 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`.

`UIModalPresentationCustom`

A custom view presentation style that is managed by a custom animator object and an optional interactive controller object. Both of the animator and interactive controller object are provided by the view controller's transitioning delegate, which is stored in the `transitioningDelegate` (page 40) property.

Available in iOS 7.0 and later.

Declared in `UIViewController.h`.

`UIModalPresentationNone`

A nonmodal view presentation or dismissal.

Available in iOS 7.0 and later.

Declared in `UIViewController.h`.

Exceptions

Exceptions raised by view controllers.

```
NSString *const UIViewControllerHierarchyInconsistencyException
```

Constants

`UIViewControllerHierarchyInconsistencyException`

Raised if the view controller hierarchy is inconsistent with the view hierarchy.

When a view controller's view is added to the view hierarchy, the system walks up the view hierarchy to find the first parent view that has a view controller. That view controller must be the parent of the view controller whose view is being added. Otherwise, this exception is raised. This consistency check is also performed when a view controller is added as a child by calling the `addChildViewController:` method.

It is also allowed for a view controller that has no parent to add its view to the view hierarchy. This is generally not recommended, but is useful in some special cases.

Available in iOS 5.0 and later.

Declared in `UIViewController.h`.

Deprecated UIViewController Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in iOS 5.0

didAnimateFirstHalfOfRotationToInterfaceOrientation:

Sent to the view controller after the completion of the first half of the user interface rotation. (Deprecated in iOS 5.0. Use the one-step rotation technique instead. See the `willAnimateRotationToInterfaceOrientation:duration:` *(page 77) method.)*

```
- (void)didAnimateFirstHalfOfRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
```

Parameters

`toInterfaceOrientation`

The state of the app's user interface orientation after the rotation. The possible values are described in the `UIInterfaceOrientation` enum.

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 2.0 and later.

Deprecated in iOS 5.0.

See Also

– `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 84)
– `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 85)

Declared in

`UIViewController.h`

willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:

Sent to the view controller before performing the first half of a user interface rotation. (Deprecated in iOS 5.0. Use the one-step rotation technique instead. See the `willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.)

```
- (void)willAnimateFirstHalfOfRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
 duration:(NSTimeInterval)duration
```

Parameters

`toInterfaceOrientation`

The state of the app's user interface orientation before the rotation. The possible values are described in `UIInterfaceOrientation`.

`duration`

The duration of the first half of the pending rotation, measured in seconds.

Discussion

The default implementation of this method does nothing.

This method is called from within the animation block 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 25) property is still set to the old orientation.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 5.0.

See Also

– `willRotateToInterfaceOrientation:duration:` (page 79)
– `willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:` (page 85)
– `didRotateFromInterfaceOrientation:` (page 48)

Declared in

`UIViewController.h`

willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:

Sent to the view controller before the second half of the user interface rotates. (Deprecated in iOS 5.0. Use the one-steprotationtechniqueinstead.Seethe`willAnimateRotationToInterfaceOrientation:duration:` (page 77) method.)

```
–
(void)willAnimateSecondHalfOfRotationFromInterfaceOrientation:(UIInterfaceOrientation)fromInterfaceOrientation
 duration:(NSTimeInterval)duration
```

Parameters

`fromInterfaceOrientation`
The state of the app's user interface orientation before the rotation. The possible values are described in the `UIInterfaceOrientation` enum.

`duration`
The duration of the second half of the pending rotation, measured in seconds.

Discussion

The default implementation of this method does nothing.

This method is called from inside the animation block 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 25) property is set to the new orientation.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 5.0.

See Also

– `willRotateToInterfaceOrientation:duration:` (page 79)
– `willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:` (page 84)
– `didRotateFromInterfaceOrientation:` (page 48)

Declared in

`UIViewController.h`

Deprecated in iOS 6.0

modalViewController

The controller for the active presented view–that is, the view that is temporarily displayed on top of the view managed by the receiver. (read-only) (Deprecated in iOS 6.0. Use `presentedViewController` (page 32) instead.)

```
@property(nonatomic, readonly) UIViewController *modalViewController
```

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 6.0.

Related Sample Code
GKAchievements

Declared in
`UIViewController.h`

automaticallyForwardAppearanceAndRotationMethodsToChildViewControllers

Returns a Boolean value that indicates whether appearance and rotation methods are forwarded. (Deprecated in iOS 6.0. Use `shouldAutomaticallyForwardRotationMethods` (page 67) and `shouldAutomaticallyForwardAppearanceMethods` (page 66) instead.)

```
- (BOOL)automaticallyForwardAppearanceAndRotationMethodsToChildViewControllers
```

Return Value
A Boolean value that indicates whether appearance and rotation methods are forwarded.

Discussion
This method is called to determine whether to automatically forward containment callbacks to the child view controllers.

The default implementation returns `YES`. Subclasses of the `UIViewController` class that implement containment logic may override this method to control how these methods are forwarded. If you override this method and return `NO`, you are responsible for forwarding the following methods to child view controllers at the appropriate times:

- `viewWillAppear:` (page 75)
- `viewDidAppear:` (page 73)
- `viewWillDisappear:` (page 76)

`viewDidDisappear:` (page 73)

`willRotateToInterfaceOrientation:duration:` (page 79)

`willAnimateRotationToInterfaceOrientation:duration:` (page 77)

`didRotateFromInterfaceOrientation:` (page 48)

Availability
Available in iOS 5.0 and later.

Deprecated in iOS 6.0.

Declared in
`UIViewController.h`

dismissModalViewControllerAnimated:

Dismisses the view controller that was presented by the receiver. (Deprecated in iOS 6.0. Use `dismissViewControllerAnimated:completion:` (page 49) instead.)

```
- (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` (page 28) property of the view controller being dismissed.

Discussion
The presenting view controller is responsible for dismissing the view controller it presented. If you call this method on the presented view controller itself, however, it automatically forwards the message to the presenting view controller.

If you present several view controllers in succession, and thus build a stack of presented 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 presented view controller, get the value in the `modalViewController` (page 86) property before calling this method.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 6.0.

Related Sample Code
CryptoExercise

GKAchievements

LocateMe

NavBar

Declared in

`UIViewController.h`

presentModalViewController:animated:

Presents a modal view managed by the given view controller to the user. (Deprecated in iOS 6.0. Use `presentViewController:animated:completion:` (page 61) instead.)

```
- (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, the view of `modalViewController` is always presented full screen. On iPad, the presentation depends on the value in the `modalPresentationStyle` (page 27) property.

Sets the `modalViewController` (page 86) 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` (page 28) property of the controller in the `modalViewController` parameter.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 6.0.

See Also

– `dismissModalViewControllerAnimated:` (page 87)

Related Sample Code
AddMusic

GKTapper

ListAdder

LocateMe

MVCNetworking

Declared in
`UIViewController.h`

shouldAutorotateToInterfaceOrientation:

Returns a Boolean value indicating whether the view controller supports the specified orientation. (Deprecated in iOS 6.0. Override the `supportedInterfaceOrientations` (page 69) and `preferredInterfaceOrientationForPresentation` (page 58) methods instead.)

```
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
```

Parameters

`interfaceOrientation`
: The orientation of the app's user interface after the rotation. The possible values are described in `UIInterfaceOrientation`.

Return Value
`YES` if the view controller auto-rotates its view to the specified orientation; otherwise, `NO`.

Discussion
By default, this method returns `YES` for the `UIInterfaceOrientationPortrait` 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 25) 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.

Deprecated in iOS 6.0.

See Also

– `rotatingFooterView` (page 62)
– `rotatingHeaderView` (page 63)

Declared in

`UIViewController.h`

viewDidUnload

Called when the controller's view is released from memory. (Deprecated in iOS 6.0. Views are no longer purged under low-memory conditions and so this method is never called.)

```
- (void)viewDidUnload
```

Discussion

In iOS 5 and earlier, when a low-memory condition occurred and the current view controller's views were not needed, the system could opt to call this method after the view controller's view had been released. This method was your chance to perform any final cleanup. If your view controller stored separate references to the view or its subviews, you could use this method to release those references. You could also use this method to remove references to any objects that you created to support the view but that are no longer needed now that the view is gone. You would not use this method to release user data or any other information that cannot be easily recreated.

In iOS 6 and later, clearing references to views and other objects in your view controller is unnecessary.

At the time this method is called, the `view` property is `nil`.

Availability

Available in iOS 3.0 and later.

Deprecated in iOS 6.0.

Related Sample Code
AdvancedURLConnections

iPhoneCoreDataRecipes

LocateMe

SimpleFTPSample

SimpleNetworkStreams

Declared in

`UIViewController.h`

viewWillUnload

Called just before releasing the controller's view from memory. (Deprecated in iOS 6.0. Views are no longer purged under low-memory conditions and so this method is never called.)

```
- (void)viewWillUnload
```

Discussion
In iOS 5 and earlier, when a low-memory condition occurred and the current view controller's views were not needed, the system could opt to remove those views from memory. This method was called prior to releasing the actual views so you could perform any cleanup prior to the view being deallocated. For example, you could use this method to remove views as observers of notifications or record the state of the views so it can be reestablished when the views are reloaded.

In iOS 6 and later, clearing references to views is no longer necessary. As a result, any other cleanup related to those views, such as removing them as observers, is also not necessary.

At the time this method is called, the `view` property is still valid (it has not yet been set to `nil`).

Availability
Available in iOS 5.0 and later.

Deprecated in iOS 6.0.

Declared in
`UIViewController.h`

Deprecated in iOS 7.0

contentSizeForViewInPopover

The size of the view controller's view while displayed in a popover. (Deprecated in iOS 7.0.)

```
@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.

If the popover controller displaying the view controller sets its `popoverContentSize` property, the popover controller overrides the values set in the view controller's `contentSizeForViewInPopover` property.

Availability
Available in iOS 3.2 and later.

Deprecated in iOS 7.0.

Declared in
`UIPopoverController.h`

wantsFullScreenLayout

A Boolean value indicating whether the view should underlap the status bar. (Deprecated in iOS 7.0.)

```
@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 underlap a tab bar but is resized to underlap translucent toolbars. Regardless of the value of this property, navigation controllers always allow views to underlap translucent navigation bars.

The default value of this property is `NO`, which causes the view to be laid out so it does not underlap the status bar.

Availability
Available in iOS 3.0 and later.

Deprecated in iOS 7.0.

Related Sample Code
UIImagePicker Video Recorder

Declared in
`UIViewController.h`

Document Revision History

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

Date	Notes
2013-09-18	Added descriptions for new properties in iOS 7: `automaticallyAdjustsScrollViewInsets` (page 20), `bottomLayoutGuide` (page 21), `edgesForExtendedLayout` (page 24), `extendedLayoutIncludesOpaqueBars` (page 25), `modalPresentationCapturesStatusBarAppearance` (page 27), `preferredContentSize` (page 31), `topLayoutGuide` (page 38), and `transitioningDelegate` (page 40). Added descriptions for new instance methods in iOS 7: `applicationFinishedRestoringState` (page 43), `childViewControllerForStatusBarHidden` (page 45), `childViewControllerForStatusBarStyle` (page 45), `preferredStatusBarStyle` (page 59), `preferredStatusBarUpdateAnimation` (page 59), `prefersStatusBarHidden` (page 60), `setNeedsStatusBarAppearanceUpdate` (page 65), and `transitionCoordinator` (page 70). Added descriptions for new constants in iOS 7: `UIModalPresentationCustom` (page 82) and `UIModalPresentationNone` (page 82). Clarified the description for the `viewDidLayoutSubviews` (page 74) method.
2012-09-19	Added properties and methods related to state preservation and restoration. Added method to determine whether a segue should be executed.

Date	Notes
2012-02-16	Reorganized tasks section to better reflect common usage. Substantial edits and rewrites to clarify specific behaviors, particularly for methods and properties added in iOS 5.
2011-10-12	Updated the document to include new methods related to view controller containment.
2011-01-07	Added the disablesAutomaticKeyboardDismissal method.
2010-10-13	Updated document to address iPad issues, fixed typos, and made other minor corrections.
2010-05-20	Updated information about when navigation items may be retrieved.
2010-03-01	Updated for iOS 3.2.
2009-08-19	Updated the descriptions of the viewWillAppear, viewDidAppear:, viewWillDisappear, and viewDidDisappear: methods.
2009-06-17	Updated for iOS 3.0.
2008-11-13	Added new API and warnings to will/did appear messages.
2008-09-09	Added additional information to rotatingHeaderView and rotatingFooterView methods.
2008-07-06	New document that describes the class providing the fundamental view-management model for iPhone.