iOS 7 UI Transition Guide

Developer

Contents

Tables

Preparing for Transition

Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology.

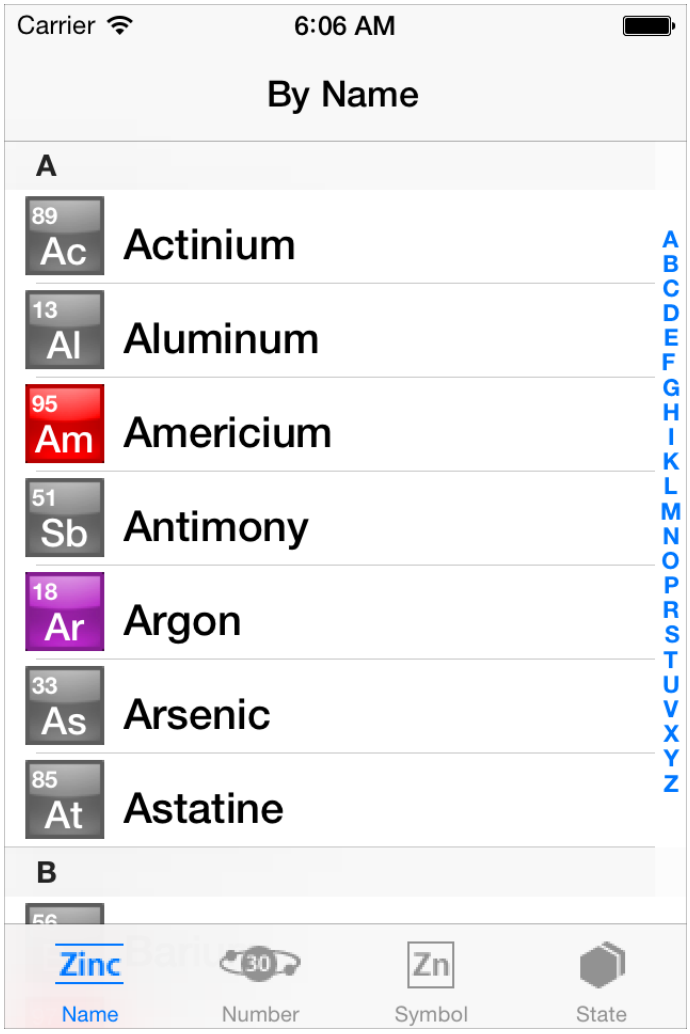
Before You Start

Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology.

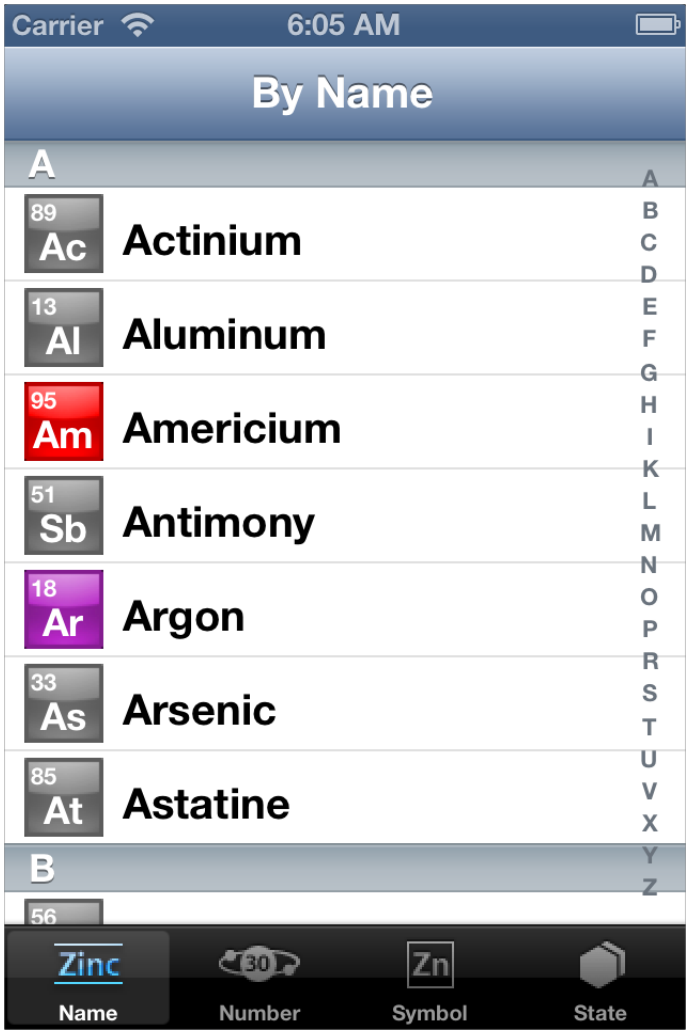
iOS 7 introduces many UI changes, such as borderless buttons, translucent bars, and full-screen layout for view controllers. Using Xcode 5, you can build a project for iOS 7 and run it in iOS 7 Simulator to get a first glimpse of the way the app looks with iOS 7 UI.

For example, the only differences between the two versions of *TheElements* sample project shown below are the deployment target and the simulator.

TheElements sample app in iOS 7 Simulator



TheElements sample app in iOS 6 Simulator



It's tempting to dive straight into the work of updating your app, but there are a few things to think about before beginning the process.

As you interact with the built-in apps, it becomes clear that the changes in iOS 7 are both subtle and profound. Familiar UI elements are easily recognizable but look very different. Visual touches of physicality and realism are muted and refined, while realism in motion is enhanced.

> **Note:** Although all UI elements look different in iOS 7—and many support new functionality—the UIKit APIs you're familiar with remain mostly the same.

As you continue to explore, you begin to discern the main themes of iOS 7:

- **Deference.** The UI helps users understand and interact with the content, but never competes with it.

- **Clarity.** Text is legible at every size, icons are precise and lucid, adornments are subtle and appropriate, and a sharpened focus on functionality motivates the design.
- **Depth.** Visual layers and realistic motion heighten users' delight and understanding.

By bringing fundamental and pervasive changes to the iOS experience, iOS 7 provides a rare opportunity to revisit the way apps communicate their core purpose and functionality to users. Although you might not be prepared to take full advantage of this opportunity today, keep it in mind as you update your app to run in iOS 7. (If you are ready to revisit your app design—or you're beginning a new project—read "Designing for iOS 7" for some guidance.)

Characterizing Your App

Whether you decide to redesign an app or update its current design, you need to know how the app's characteristics can influence the process. First, use the following questions to help shape your strategy:

- **Did you use Auto Layout to design the app?**

  If your app uses Auto Layout, your job is easier. In Xcode 5, Auto Layout can help an app accommodate new UI element metrics and respond appropriately to dynamic changes in text size. Auto Layout is particularly helpful if you're transitioning an iOS 6 app or you need to support both iOS 6 and iOS 7.

  If you didn't use Auto Layout, now may be the perfect time to start, especially if you need to support more than one version of an app. If you use manual or programmatic layout techniques, you're responsible for ensuring that the layout adjusts appropriately when text size changes.

- **Does the app need to support iOS 6?**

  Remember that iOS users tend to be very quick to update their devices, and they expect their favorite apps to follow suit.

  If business reasons require you to support iOS 6, it's still best to begin by updating the current app for iOS 7. Then—as much as possible—apply the design changes to the iOS 6 version of the app. For some details of this process, see "Supporting iOS 6" (page 12).

The next step in shaping a transition strategy is to examine the ways in which the app is customized. The amount of customization—and the specific customization techniques you use—impact the type of work you have to do.

Think of apps as being divided into the following three types:

- **Standard.** The app contains only standard, uncustomized UI elements provided by UIKit.
- **Custom.** The app presents a completely custom UI that doesn't include any UIKit UI elements.

- **Hybrid.** The app contains a mix of standard and custom elements, including standard elements that you customized using UIKit tinting and appearance-customization APIs.

For a standard app, you need to decide whether your visual and user experience designs still make sense in the iOS 7 environment. If you decide to keep the current layout and interaction model, most of the work involves making minor adjustments and ensuring that the app handles the new systemwide gestures correctly.

Custom apps—that is, apps that use no UIKit UI elements—require a more nuanced approach. For example, if you feel that the current UI and experience of the app is still appropriate, there may be very little to do. On the other hand, if you feel that the app's personality and user experience should change in order to delight iOS 7 users, you have more work to do.

Hybrid apps vary in the amount of work required, depending on the customizations you did and how you combined custom and standard elements. In addition to revisiting the overall design of a hybrid app, you need to make sure that your customizations still work well and look good when they're integrated with standard elements.

> **Note:** An app that mimics standard iOS 6 UI in a completely custom way is likely to require a lot of work because it will simply look out of date.

Scoping the Project

Knowing your app's compatibility requirements and customization characteristics gives you some idea of the path to take. Use the following checklists to fill in more details and to scope the project.

Things Every App Must Do

- ✓ Update the app icon.

  In iOS 7, app icons for high-resolution iPhone and iPod touch are 120 x 120 pixels; for high-resolution iPad, app icons are 152 x 152 pixels. (To learn more about all icon sizes, see "Icon and Image Sizes".)

  Note that iOS 7 doesn't apply shine or a drop shadow to the app icon. And, although iOS 7 still applies a mask that rounds the corners of an app icon, it uses a different corner radius than earlier versions of iOS.

- ✓ Update the launch image to include the status bar area if it doesn't already do so.
- ✓ Support Retina display and iPhone 5 in all your artwork and designs, if you're not already doing so.

Things Every App Should Do

- ✓ Make sure that app content is discernible through translucent UI elements—such as bars and keyboards—and the transparent status bar. In iOS 7, view controllers use full-screen layout (to learn more, see "Using View Controllers" (page 17)).
- ✓ Redesign custom bar button icons. In iOS 7, bar button icons are lighter in weight and have a different style.
- ✓ Prepare for borderless buttons by reassessing the utility of button background images and bezels in your layout.
- ✓ Examine your app for hard-coded UI values—such as sizes and positions—and replace them with those you derive dynamically from system-provided values. Use Auto Layout to help your app respond when layout changes are required. (If you're new to Auto Layout, learn about it by reading *Cocoa Auto Layout Guide*.)

- ✓ Examine your app for places where the metrics and style changes of UIKit controls and views affect the layout and appearance. For example, switches are wider, grouped tables are no longer inset, and progress views are thinner. For more information on specific UI elements, see "Bars and Bar Buttons" (page 23), "Content Views" (page 31), "Controls" (page 42), and "Temporary Views" (page 50).
- ✓ Adopt Dynamic Type. In iOS 7, users can adjust the text size they see in apps. When you adopt Dynamic Type, you get text that responds appropriately to user-specified size changes. For more information, see "Using Fonts" (page 19).
- ✓ Expect users to swipe up from the bottom of the screen to reveal Control Center. If iOS determines that a touch that begins at the bottom of the screen should reveal Control Center, it doesn't deliver the gesture to the currently running app. If iOS determines that the touch should not reveal Control Center, the touch may be slightly delayed before it reaches the app.
- ✓ Revisit the use of drop shadows, gradients, and bezels. Because the iOS 7 aesthetic is smooth and layered—with much less emphasis on using visual effects to make UI elements look physical—you may want to rethink these effects.
- ✓ If necessary, update your app to best practices for iOS 6—such as Auto Layout and storyboards—and ensure that the app doesn't use deprecated APIs.

Now that you have a better idea of the types of things you need to do, learn more about changes in view controllers, tinting, and fonts by reading "Appearance and Behavior" (page 17).

If You Must Continue to Support iOS 6

If you must support both iOS 6 and iOS 7, you can detect which OS version the app is running in and load the appropriate resources. For more information, see "Supporting iOS 6" (page 12).

Supporting iOS 6

If business reasons require you to continue supporting iOS 6 or earlier, you need to choose the most practical way to update the app for iOS 7. The techniques you choose can differ, but the overall advice remains the same: First, focus on redesigning the app for iOS 7. Then, bring the changes to the iOS 6 version as appropriate.

> **Note:** On a device running iOS 7, all of the system UI—such as alerts and notifications—uses the iOS 7 appearance, even if your app is currently using an earlier appearance.

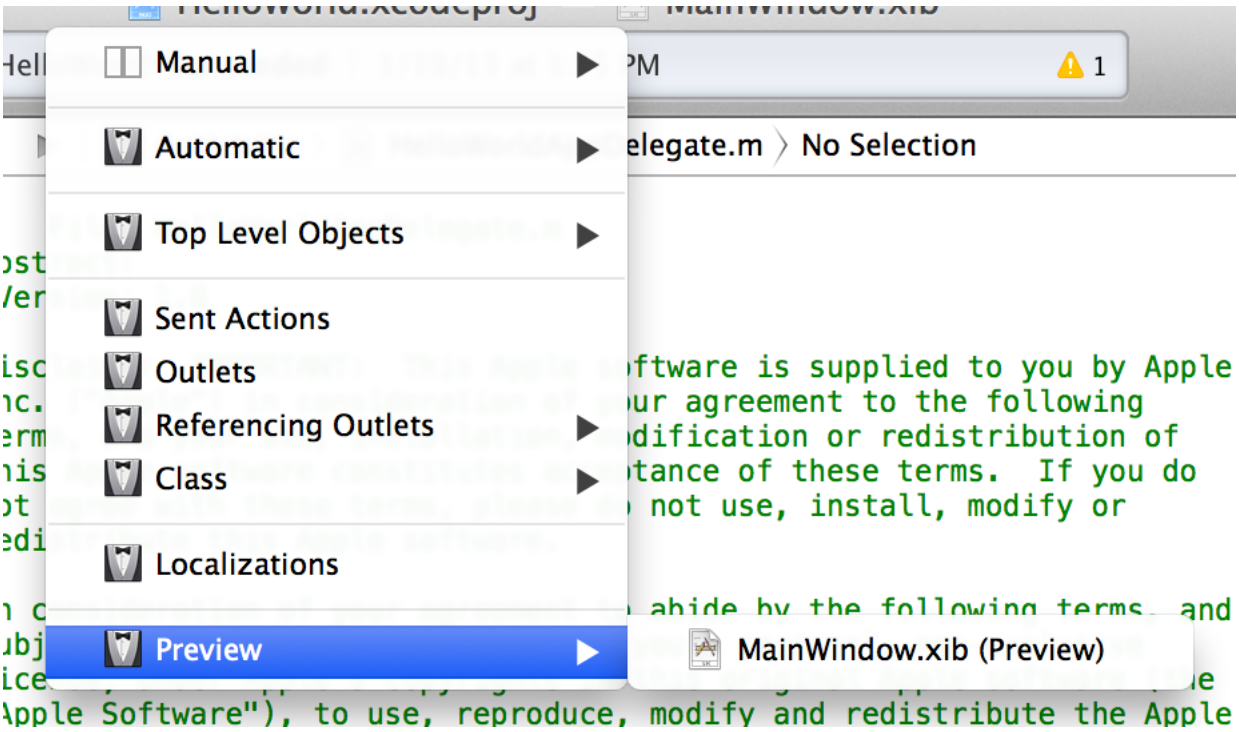
Using Interface Builder to Support Multiple App Versions

Interface Builder in Xcode 5 includes new features that help you transition an app to iOS 7 while continuing to support earlier versions.

Get a preview of how the UI updates you make affect an earlier version. Using the assistant editor, you can make changes to an iOS 7 storyboard or XIB file on the canvas and simultaneously see how those changes look in the iOS 6 version of the file.

Follow these steps to preview an earlier storyboard or XIB file:

1. While viewing the iOS 7 storyboard or XIB file on the canvas, open the assistant editor.
2. Open the Assistant pop-up menu. (The Assistant pop-up menu is the first item to the right of the back and forward arrows in the assistant editor jump bar.)
3. In the menu, scroll to the Preview item and choose the storyboard or XIB file.



Toggle between viewing app UI in iOS 7 and iOS 6.1 or earlier. If your app needs to support iOS 6.1 or earlier, use this feature to make sure the UI looks correct in all versions of the app.

Follow these steps to switch between two versions of the UI:

1. Open the File inspector in Interface Builder.
2. Open the "View as" menu.
3. Choose the version of the UI you want to view.

For more information about new Interface Builder features in Xcode 5, see *What's New in Xcode*.

Supporting Two Versions of a Standard App

If both versions of a standard app should have a similar layout, use Auto Layout to create a UI that works correctly in both versions of iOS. To support multiple versions of iOS, specify a single set of constraints that Auto Layout can use to adjust the views and controls in the storyboard or XIB files (to learn more about constraints, see "Constraints Express Relationships Between Views").

If both versions of a standard app should have a similar layout and you're not using Auto Layout, use offsets. To use offsets, first update the UI for iOS 7. Next, specify values that define the origin, height, and width of each element in the earlier UI as offsets from the element's new position in the iOS 7 UI.

To learn more about Auto Layout, see *Cocoa Auto Layout Guide*.

Managing Multiple Images in a Hybrid App

Hybrid apps often include custom image assets, such as bar button icons, and background views for bars or other controls. Apps can use one or more asset catalogs to manage these resources. (To learn more about asset catalogs, see *Asset Catalog Help*.)

> **Note:** An asset catalog contains resources that are displayed within an app; an asset catalog doesn't hold the app icon, launch image, or any other image that an outside process needs to access.

In a hybrid app that must support multiple versions of iOS, you manage the images yourself. Images that differ depending on an app's version should have unique names; otherwise, you can use the same image in both versions.

If your storyboard or XIB file contains an embedded image, consider creating an outlet to the image view and loading the appropriate resource as needed. To learn how to load different assets in code, see "Loading Resources Conditionally."

Loading Resources Conditionally

In some cases, you need to determine the iOS version your app is currently running in so you can respond to version differences appropriately in code. For example, if different versions of an app use significantly different layouts, you can load different storyboard or XIB files for each version. You may also need to use different code paths to handle API differences, such using `barTintColor` instead of `tintColor` to tint a bar's background.

If you need to load different resources for different app versions—and you currently identify a storyboard or XIB file in your `Info.plist` file—you can use the version of the Foundation framework to determine the current system version and load the appropriate resource in `application:didFinishLaunchingWithOptions:`. The code below shows how to check the Foundation framework version:

```
if (floor(NSFoundationVersionNumber) <= NSFoundationVersionNumber_iOS_6_1) {
   // Load resources for iOS 6.1 or earlier
} else {
   // Load resources for iOS 7 or later
```

```
}
```

Updating the UI

Appearance and Behavior

Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology.

iOS 7 brings several changes to how you lay out and customize the appearance of your UI. The changes in view-controller layout, tint color, and font affect all the UIKit objects in your app. In addition, enhancements to gesture recognizer APIs give you finer grained control over gesture interactions.

Using View Controllers

In iOS 7, view controllers use full-screen layout. At the same time, iOS 7 gives you more granular control over the way a view controller lays out its views. In particular, the concept of full-screen layout has been refined to let a view controller specify the layout of each edge of its view.

The `wantsFullScreenLayout` view controller property is deprecated in iOS 7. If you currently specify `wantsFullScreenLayout = NO`, the view controller may display its content at an unexpected screen location when it runs in iOS 7.

To adjust how a view controller lays out its views, `UIViewController` provides the following properties:

- `edgesForExtendedLayout`

  The `edgesForExtendedLayout` property uses the `UIRectEdge` type, which specifies each of a rectangle's four edges, in addition to specifying none and all.

  Use `edgesForExtendedLayout` to specify which edges of a view should be extended, regardless of bar translucency. By default, the value of this property is `UIRectEdgeAll`.

- `extendedLayoutIncludesOpaqueBars`

If your design uses opaque bars, refine `edgesForExtendedLayout` by also setting the `extendedLayoutIncludesOpaqueBars` property to `NO`. (The default value of `extendedLayoutIncludesOpaqueBars` is `YES`.)

- `automaticallyAdjustsScrollViewInsets`

  If you don't want a scroll view's content insets to be automatically adjusted, set `automaticallyAdjustsScrollViewInsets` to `NO`. (The default value of `automaticallyAdjustsScrollViewInsets` is `YES`.)

- `topLayoutGuide`, `bottomLayoutGuide`

  The `topLayoutGuide` and `bottomLayoutGuide` properties indicate the location of the top or bottom bar edges in a view controller's view. If bars should overlap the top or bottom of a view, you can use Interface Builder to position the view relative to the bar by creating constraints to the bottom of `topLayoutGuide` or to the top of `bottomLayoutGuide`. (If no bars should overlap the view, the bottom of `topLayoutGuide` is the same as the top of the view and the top of `bottomLayoutGuide` is the same as the bottom of the view.) Both properties are lazily created when requested.

In iOS 7, view controllers can support custom animated transitions between views. In addition, you can use iOS 7 APIs to support user interaction during an animated transition. To learn more, see *UIViewControllerAnimatedTransitioningProtocolReference* and *UIViewControllerInteractiveTransitioningProtocolReference*.

iOS 7 gives view controllers the ability to adjust the style of the status bar while the app is running. A good way to change the status bar style dynamically is to implement `preferredStatusBarStyle` and—within an animation block—update the status bar appearance and call `setNeedsStatusBarAppearanceUpdate`.

> **Note:** If you prefer to opt out of having view controllers adjust the status bar style—and instead set the status bar style by using the `UIApplicationstatusBarStyle` method—add the `UIViewControllerBasedStatusBarAppearance` key to an app's `Info.plist` file and give it the value `NO`.

Using Tint Color

In iOS 7, tint color is a property of `UIView`. iOS 7 apps often use a tint to define a key color that indicates interactivity and selection state for elements throughout the app.

When you specify a tint for a view, the tint is automatically propagated to all subviews in the view's hierarchy. Because `UIWindow` inherits from `UIView`, you can specify a tint color for the entire app by setting the window's tint property using code like this:

```
window.tintColor = [UIColor purpleColor];
```

Tip: You can also set an app's tint color in Interface Builder. The Global Tint menu in the Interface Builder Document section of the File inspector lets you open the Colors window or choose a specific color.

If you don't specify a tint for the window, it uses the system default color.

By default, a view's tint color is `nil`, which means that the view uses its parent's tint. It also means that when you ask a view for its tint color, it always returns a color value, even if you haven't set one.

In general, it's best to change a view's tint color while the view is offscreen. To cause a view to revert to using its parent's tint, set the tint color to `nil`.

Important: Setting the `tintColor` property by using the appearance proxy APIs is not supported in iOS 7.

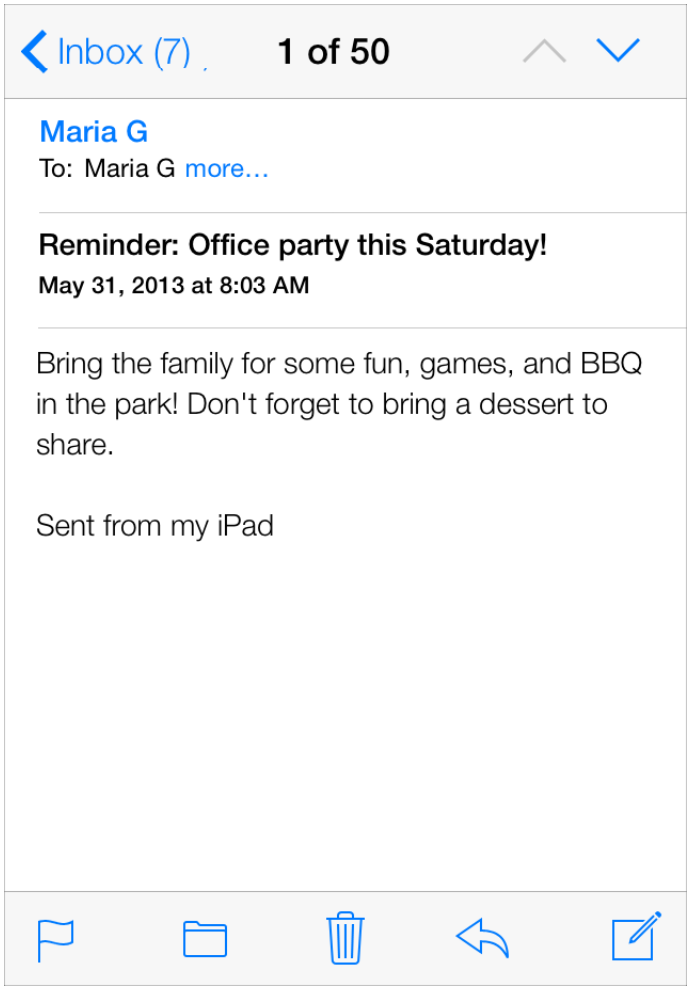
When an alert or action sheet appears, iOS 7 automatically dims the tint color of the views behind it. To respond to this color change, a custom view subclass that uses `tintColor` in its rendering should override `tintColorDidChange` to refresh the rendering when appropriate.

Note: In iOS 6, `tintColor` tinted the background of navigation bars, tab bars, toolbars, search bars, and scope bars. To tint a bar background in iOS 7, use the `barTintColor` property instead.

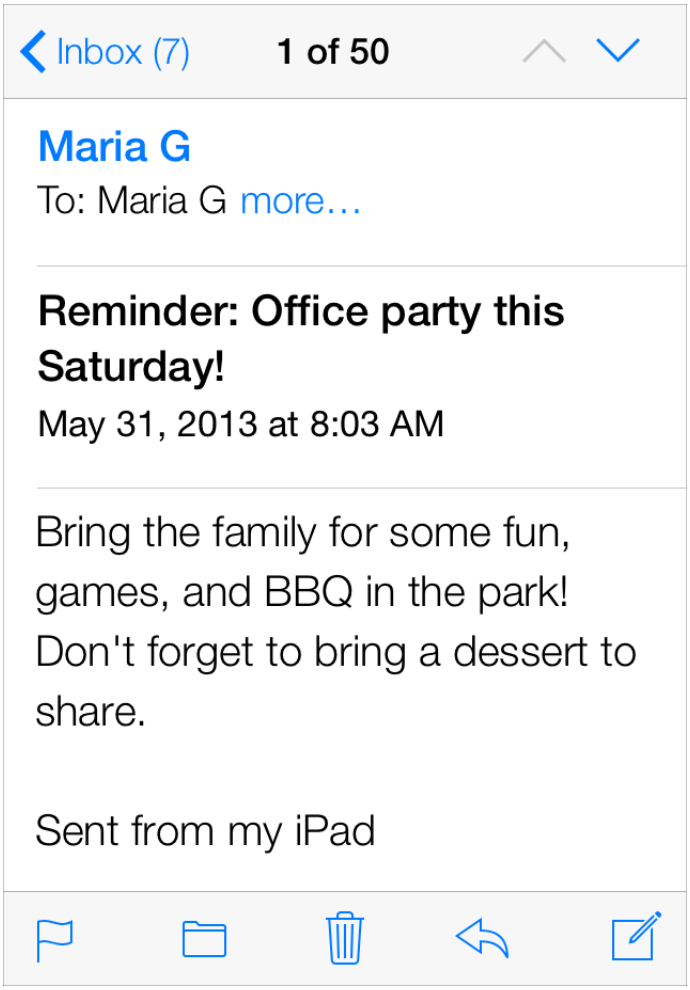
Using Fonts

iOS 7 introduces Dynamic Type, which makes it easy to display great-looking text in your app.

A message at the smallest size



A message at the largest non Accessibility size



When you adopt Dynamic Type, you get:

- Automatic adjustments to weight, letter spacing, and line height for every font size.
- The ability to specify different text styles for semantically distinct blocks of text, such as `Body`, `Footnote`, or `Headline`.
- Text that responds appropriately to changes in both the Dynamic Type and accessibility settings for user-specified text sizes.

To take advantage of these features, use the `UIFont` method `preferredFontForTextStyle` to get a font, instead of specifying font names or sizes. iOS 7 optimizes fonts that are specified by this method for maximum legibility at every size.

Using Gesture Recognizers

Prior to iOS 7, if a gesture recognizer requires another gesture recognizer to fail, you use `requireGestureRecognizerToFail:` to set up a permanent relationship between the two objects at creation time. This works fine when gesture recognizers aren't created elsewhere in the app—or in a framework—and the set of gesture recognizers remains the same.

In iOS 7, `UIGestureRecognizerDelegate` introduces two methods that allow failure requirements to be specified at runtime by a gesture recognizer delegate object:

- `gestureRecognizer:gestureRecognizer shouldRequireFailureOfGestureRecognizer:otherGestureRecognizer`
- `gestureRecognizer:gestureRecognizer shouldBeRequiredToFailByGestureRecognizer:otherGestureRecognizer`

> **Note:** iOS 7 introduces similar methods in `UIGestureRecognizerSubclass` (described in *UIGestureRecognizer Class Reference*), to give subclasses the ability to define class-wide failure requirements.

For both methods, the gesture recognizer delegate is called once per recognition attempt, which means that failure requirements can be determined lazily. It also means that you can set up failure requirements between recognizers in different view hierarchies.

An example of a situation where dynamic failure requirements are useful is in an app that attaches a screen-edge pan gesture recognizer to a view. In this case, you might want all other relevant gesture recognizers associated with that view's subtree to require the screen-edge gesture recognizer to fail so you can prevent any graphical glitches that might occur when the other recognizers get canceled after starting the recognition process. To do this, you could use code similar to the following:

```
UIScreenEdgePanGestureRecognizer *myScreenEdgePanGestureRecognizer;

...

myScreenEdgePanGestureRecognizer = [[UIScreenEdgePanGestureRecognizer alloc]
initWithTarget:self action:@selector(handleScreenEdgePan:)];

myScreenEdgePanGestureRecognizer.delegate = self;

// Configure the gesture recognizer and attach it to the view.

...

 - (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
shouldBeRequiredToFailByGestureRecognizer:(UIGestureRecognizer
*)otherGestureRecognizer {

    BOOL result = NO;
```

```
    if ((gestureRecognizer == myScreenEdgePanGestureRecognizer) &&
[[otherGestureRecognizer view] isDescendantOfView:[gestureRecognizer view]]) {
        result = YES;
    }
    return result;
 }
```

Bars and Bar Buttons

In iOS 7, the status bar is transparent, and other bars—that is, navigation bars, tab bars, toolbars, search bars, and scope bars—are translucent. As a general rule, you want to make sure that content fills the area behind the bars in your app.

Most bars also draw a blur behind them, unless you provide a custom background image for the bar.

iOS 7 introduces the `barPosition` property for identifying bar position, which helps you specify when a custom background image should extend behind the status bar. The `UIBarPositionTopAttached` value means that a bar is at the top of the screen and its background extends upward into the status bar area. In contrast, the `UIBarPositionTop` value means that a bar is at the top of its local context—for example, at the top of a popover—and that it doesn't provide a background for the status bar.

By default, all bar buttons are borderless. For details, see "Bar Buttons" (page 29).

The Status Bar

Because the status bar is transparent, the view behind it shows through. The style of the status bar refers to the appearance of its content, which includes items such as time, battery charge, and Wi-Fi signal. Use a `UIStatusBarStyle` constant to specify whether the status bar content should be dark (`UIStatusBarStyleDefault`) or light (`UIStatusBarStyleLightContent`):

`UIStatusBarStyleDefault` displays dark content. Use when light content is behind the status bar.

`UIStatusBarStyleLightContent` displays light content. Use when dark content is behind the status bar.

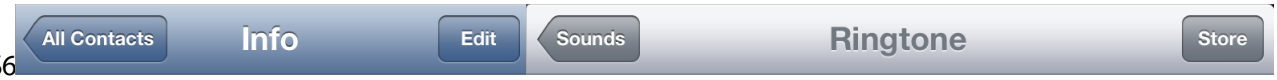
In some cases, the background image for a navigation bar or a search bar can extend up behind the status bar (for details, see "Navigation Bar" (page 24) and "Search Bar and Scope Bar" (page 25)). If there are no bars below the status bar, the content view should use the full height of the screen. To learn how to ensure that a view controller lays out its views properly, see "Using View Controllers" (page 17).

In iOS 7, you can control the style of the status bar from an individual view controller and change it while the app runs. If you prefer to opt out of this behavior and set the status bar style by using the `UIApplicationstatusBarStyle` method, add the `UIViewControllerBasedStatusBarAppearance` key to an app's `Info.plist` file and give it the value `NO`.

Navigation Bar

A navigation bar helps users navigate through an information hierarchy and, optionally, manage screen contents.

iOS7 

iOS6 

	iOS 7	iOS 6
Bar style	Translucent light (default) or translucent dark. By default, the `translucent` property is `YES`.	Opaque gradient blue (default) or opaque black. By default, the `translucent` property is `NO`.
Appearance	A one-pixel hairline appears at the bottom edge.	A drop shadow appears at the bottom edge.
Tinting	Use `tintColor` to tint bar button items. Use `barTintColor` to tint the bar background.	Use `tintColor` to tint the bar background.
Back button	The Back control is a chevron plus the title of the previous screen. *	The Back button is a bordered button that contains the title of the previous screen.

* If you want to use a custom image to replace the default chevron, you also need to create a custom mask image. iOS 7 uses the mask to make the previous screen's title appear to emerge from—or disappear into—the chevron during navigation transitions. To learn about the properties that control the Back button and mask image, see *UINavigationBar Class Reference*.

iOS 7 makes it easy to add a search bar to a navigation bar. For details, see "Search Bar and Scope Bar" (page 25).

If you create a background image for a navigation bar that uses the `UIBarPositionTopAttached` bar position—or for a navigation bar within a navigation controller—make sure the image includes the status bar area. Specifically, create a background image that has a height of 64 points.

The following table describes how iOS 7 treats resizable navigation bar background images of various heights. (To learn how to specify a resizing mode for an image, see *UIImage Class Reference*.)

Table 5-1 Treatment of resizable background images for bars at the top of the screen

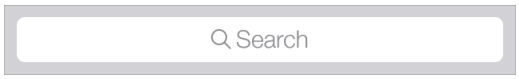
Height	Resizing treatment	Status bar background appearance
44 points	Horizontally resized as appropriate (the image is not vertically tiled or stretched).	Black, if using `UIBarPosition-TopAttached`. Provided by the window background, if using `UIBarPositionTop`.
Less than 44 points	Vertically resized to 64 points if using `UIBarPositionTopAttached` or 44 points if using `UIBarPositionTop`. Horizontally resized as appropriate.	Provided by the bar background.
64 points	Horizontally resized as appropriate.	Provided by the bar background.
1 point	Vertically resized to 64 points if using `UIBarPositionTopAttached` or if using a navigation controller. Vertically resized to 44 points if using `UIBarPositionTop`. Horizontally resized as appropriate.	Provided by the bar background.

Avoid using an extra-tall background image to display a custom drop shadow below the navigation bar. This technique won't work in iOS 7, because the extra height extends into the status bar area instead of below the navigation bar. If you want to add a drop shadow to your navigation bar, create a custom background image and use the `shadowImage` property to specify a custom shadow image.

Search Bar and Scope Bar

A search bar accepts users' text, which can be used as input for a search. A search bar can have a scope bar attached below it.

iOS 7



iOS 6

	iOS 7	iOS 6
Bar style	Translucent light (default) or translucent dark. By default, the `translucent` property is `YES`.	Opaque gradient blue (default) or opaque black. By default, the `translucent` property is `NO`.
Search bar style	Prominent (default) or minimal. A prominent search bar has a translucent background and an opaque search field. A minimal search bar has no background and a translucent search field.	–
Appearance	A one-pixel hairline appears at the bottom edge.	A drop shadow appears at the bottom edge.
Tinting	Use `tintColor` to tint foreground elements. Use `barTintColor` to tint the bar background.	Use `tintColor` to tint the bar background.

If you create a background image for a search bar that uses the `UIBarPositionTopAttached` bar position, make sure the image height includes the height of the status bar. If you create a resizable background image, see Table 5-1 (page 25) for details on how iOS 7 resizes images of various sizes.

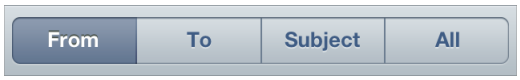
In iOS 7, `UISearchDisplayController` includes the `displaysSearchBarInNavigationBar` property, which you can use to put a search bar in a navigation bar, similar to the one in Calendar on iPhone:

A scope bar lets users define the scope of a search.

Note: A scope bar can't appear by itself; it must be attached to a search bar.

iOS 7

iOS 6

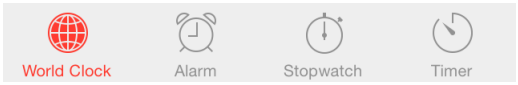


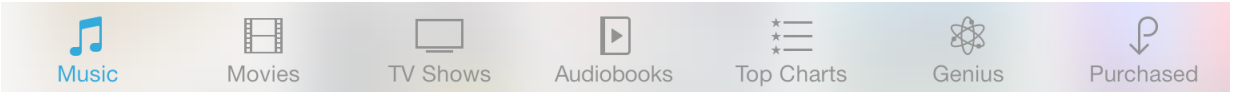
	iOS 7	iOS 6
Bar style	Translucent light (default) or translucent dark. By default, the `translucent` property is `YES`.	Opaque gradient blue (default) or opaque black. By default, the `translucent` property is `NO`.
Appearance	A one-pixel hairline appears at the bottom edge.	A drop shadow appears at the bottom edge.
Tinting	Use `tintColor` to tint scope button contents. Use `barTintColor` to tint the bar background.	Use `tintColor` to tint the bar background.

Tab Bar

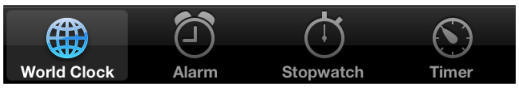
A tab bar gives people the ability to switch between different subtasks, views, and modes.

iOS 7





iOS 6



Note: In iOS 7, a tab bar on iPad has a height of 56 points.

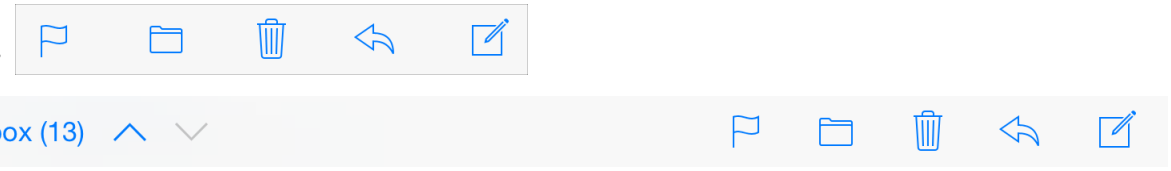
	iOS 7	iOS 6
Bar style	`UITabBar` includes the `barStyle` property. Translucent light (default) or translucent dark. By default, the `translucent` property is `YES`.	Opaque gradient black (default). In iOS 6, the tab bar doesn't include the `barStyle` or `translucent` properties.
Appearance	If provided, a custom selection indicator image is used.	A selection indicator image is drawn behind the tab icon.
Item positioning	Use `itemPositioning` to change tab layout. By default, tabs fill the width of a tab bar on iPhone; on iPad, tabs are centered by default. In a centered-item bar, you can use `itemWidth` and `itemSpacing` to customize tab layout.	In iOS 6, the tab bar doesn't include the `itemPositioning`, `itemWidth`, or `itemSpacing` properties.
Tinting	Use `tintColor` to tint selected tab bar items. Use `barTintColor` to tint the bar background.	Use `tintColor` to tint the bar background.

If you create a custom icon for a tab bar item, you should also use the `selectedImage` property of `UITabBarItem` to specify a second icon that represents the selected state of the item. If you don't provide a selected version of a custom icon, the same icon is used in both states.

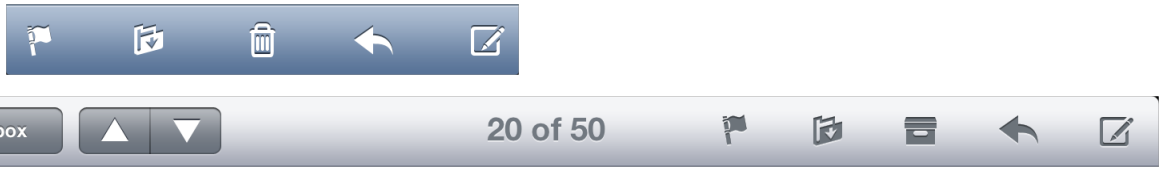
Toolbar

A toolbar contains controls that perform actions related to objects in the current screen or view.

iOS 7



iOS 6



	iOS 7	iOS 6
Bar style	Translucent light (default) or translucent dark. By default, the `translucent` property is `YES`.	Opaque gradient blue (default) or opaque black. By default, the `translucent` property is `NO`.
Appearance	A one-pixel hairline appears at the top edge.	A drop shadow appears at the top edge.
Tinting	Use `tintColor` to tint bar button items. Use `barTintColor` to tint the bar background.	Use `tintColor` to tint the bar background.
Related information	`UIToolbarPosition` constants are deprecated; use `UIBarPosition` constants instead.	–

If you create a resizable background image, see Table 5-1 (page 25) for details on how iOS 7 resizes images of various sizes.

Bar Buttons

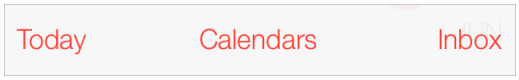
In iOS 6, bar buttons are either bordered or borderless. In iOS 7, all bar buttons are borderless.

Navigation bar buttons in iOS 7

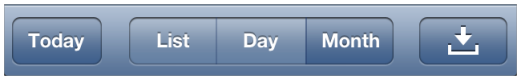
Navigation bar buttons in iOS 6

For clarity, iOS 7 apps often use titles in bar buttons instead of icons. For example, Calendar in iOS 7 uses Inbox instead of a custom icon:

iOS 7



iOS 6



In earlier versions of iOS, custom bar button art was automatically treated as a template image. (A template image is used as a mask to create the final image.) In iOS 7, you can use the following `UIImage` properties to specify whether custom art should be treated as a template image or be fully rendered:

- `UIImageRenderingModeAlwaysTemplate`. The image should be treated as a template image.
- `UIImageRenderingModeAlwaysOriginal`. The image should be rendered as is.

If you don't specify a treatment for your image—or you opt out of a treatment in a particular situation—the image receives the default treatment defined by the enclosing view. For example, by default bars use the template treatment, whereas by default a slider uses the fully rendered treatment.

> **Note:** A template image acquires the tint color of its parent (to learn more about tint color in iOS 7, see "Using Tint Color" (page 18)). If you don't want a bar button item to receive tinting, set the `UIImageRenderingModeAlwaysOriginal` property for the image.

Content Views

Content views display custom app content. Because the look of most content views is not provided by the system, visual changes in iOS 7 have almost no effect on them. The main exception is the grouped-style table view, which has a significantly changed default appearance in iOS 7.

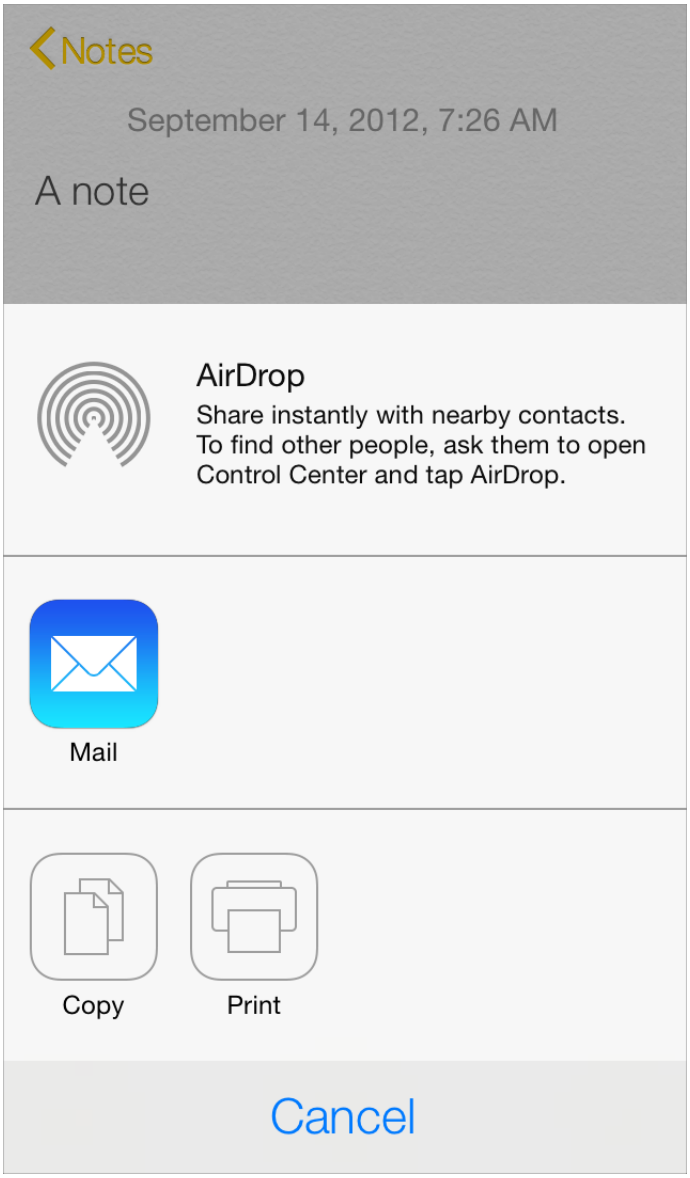
Activity

An activity represents a system-provided or custom feature that can act on the currently selected content. Users can access these features in the system-provided activity view controller that appears when they tap the Share button.

System-provided activities can use either of two icon styles:

- A style that looks like a fully rendered app icon, such as the Mail icon shown below
- A style that looks similar to a bar button item, such as the Copy and Print icons shown below

Third-party features always use the second style.



To offer a service within your app, create a simple, streamlined template image that represents it and a short title that describes it. Follow these guidelines to create a template image that looks good in the activity view controller:

- Use black or white with appropriate alpha transparency.
- Don't include a drop shadow.
- Use anti-aliasing.

Center an activity template image in an area that measures about 70 x 70 pixels (high resolution).

Collection View

A collection view manages an ordered collection of items and presents them in a customizable layout.

In iOS 7, collection views support custom animated transitions between layouts. To learn more, see *UICollectionViewTransitionLayout Class Reference*.

Photos uses collection views to display groups of photos and support transitions between them.

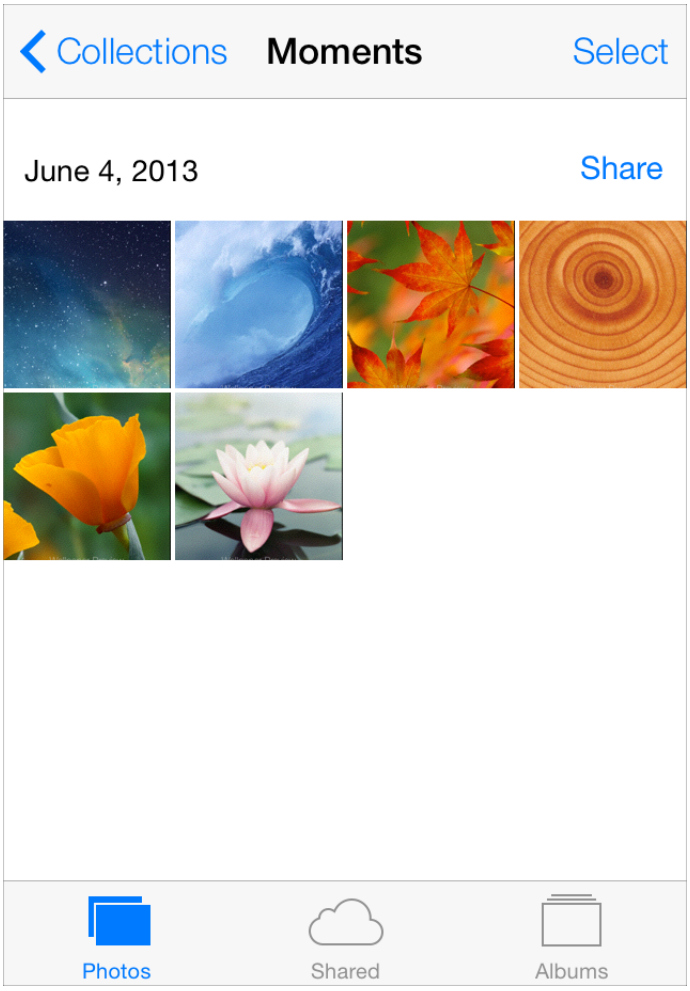


Image View

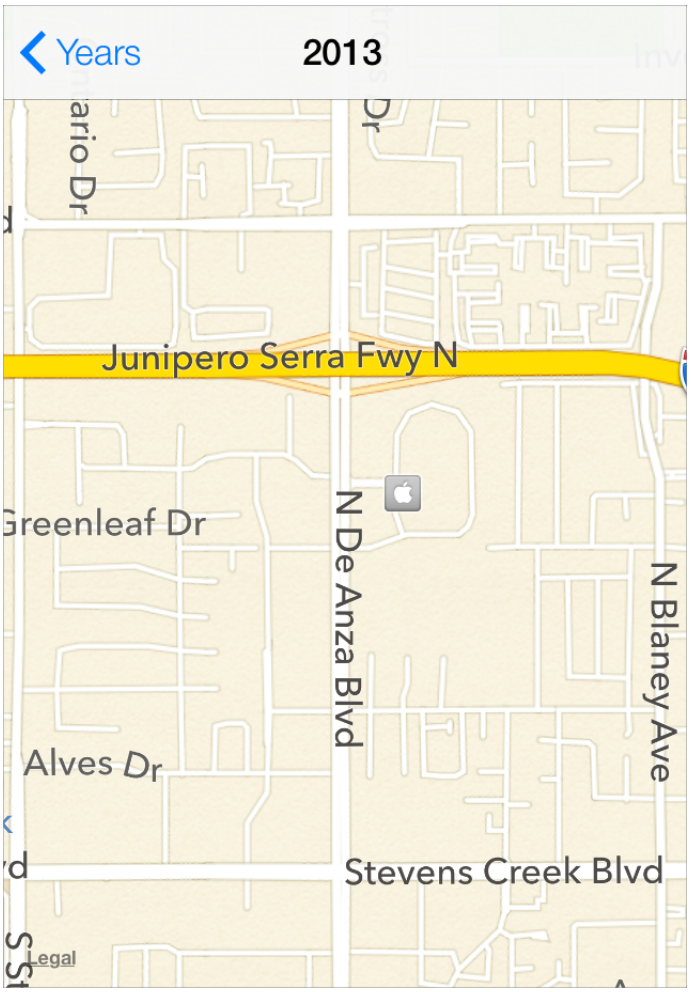
An image view displays a single image or an animated series of images.

In iOS 7, `UIImageView` includes the `tintColor` property. When the image view contains a template image—that is, an image that specifies the `UIImageRenderingModeAlwaysTemplate` rendering mode—`tintColor` is applied to the image.

Map View

A map view presents geographical data and supports most of the features provided by the built-in Maps app.

Photos uses a map view to help users view location information for a photo.



In iOS 7, use the new `MKOverlayRenderer` class to create an overlay to draw on top of a map view.

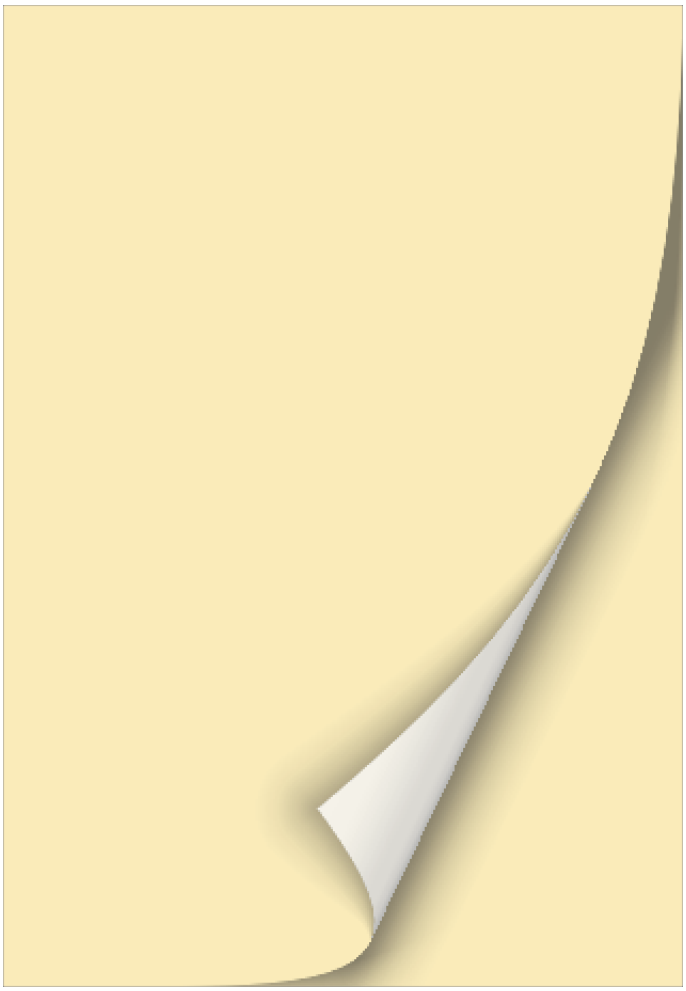
To add a 3D appearance to a map view, assign a camera object—that is, an instance of `MKMapCamera`—to a map view's `camera` property. To learn more, see *MKMapView Class Reference*.

Page View Controller

A page view controller manages multipage content using either a scrolling or page-curl transition style.

In iOS 7, use the `pageViewControllerPreferredInterfaceOrientationForPresentation` and `pageViewControllerSupportedInterfaceOrientations` methods to specify preferred and supported orientations, respectively.

Below, you can see the default appearance of a page view controller in iOS 7 Simulator:

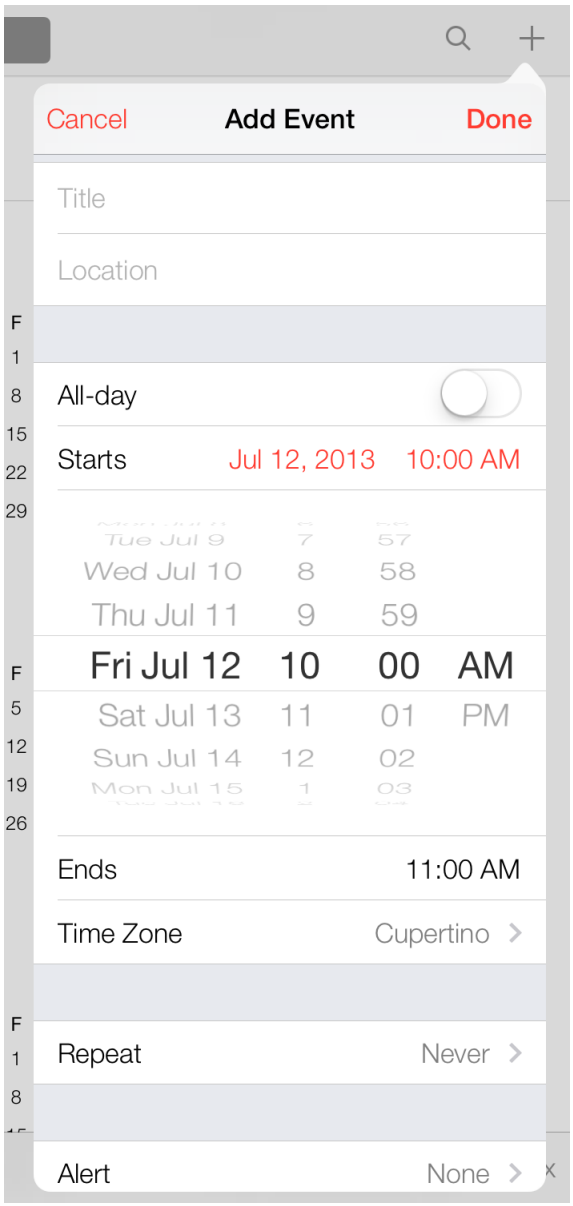


Popover (iPad Only)

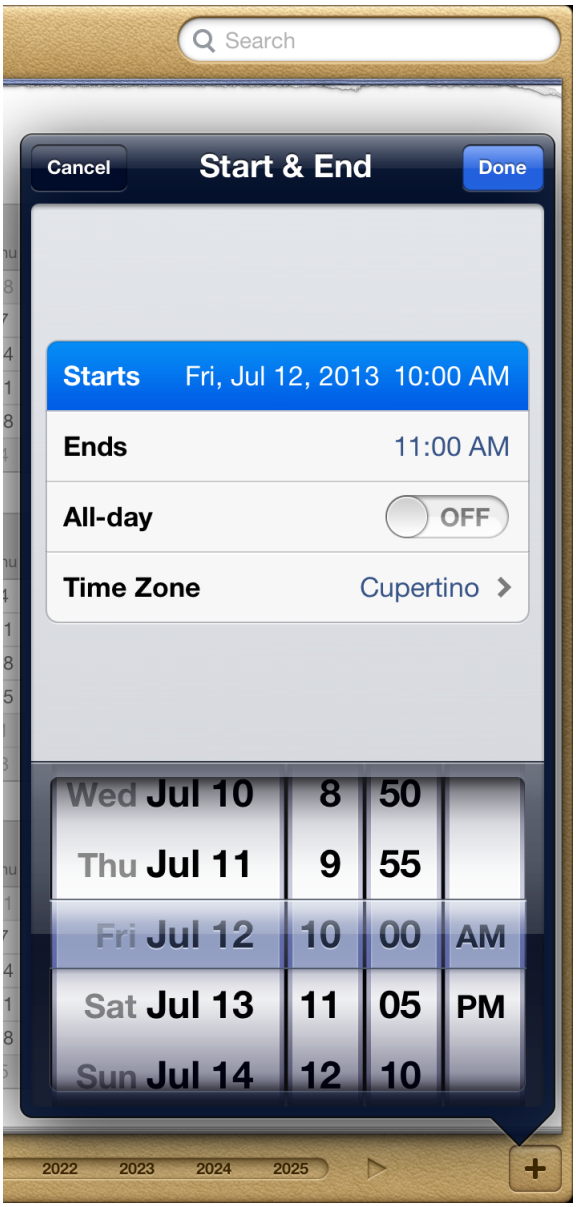
A popover is a transient view that can be revealed when people tap a control or an onscreen area.

In iOS 7, the popover background is a white blur, which means that the background of the popover's content view can be transparent. A table view inside a popover automatically uses a translucent appearance; custom content inside a popover should use a translucent appearance.

iOS 7



iOS 6

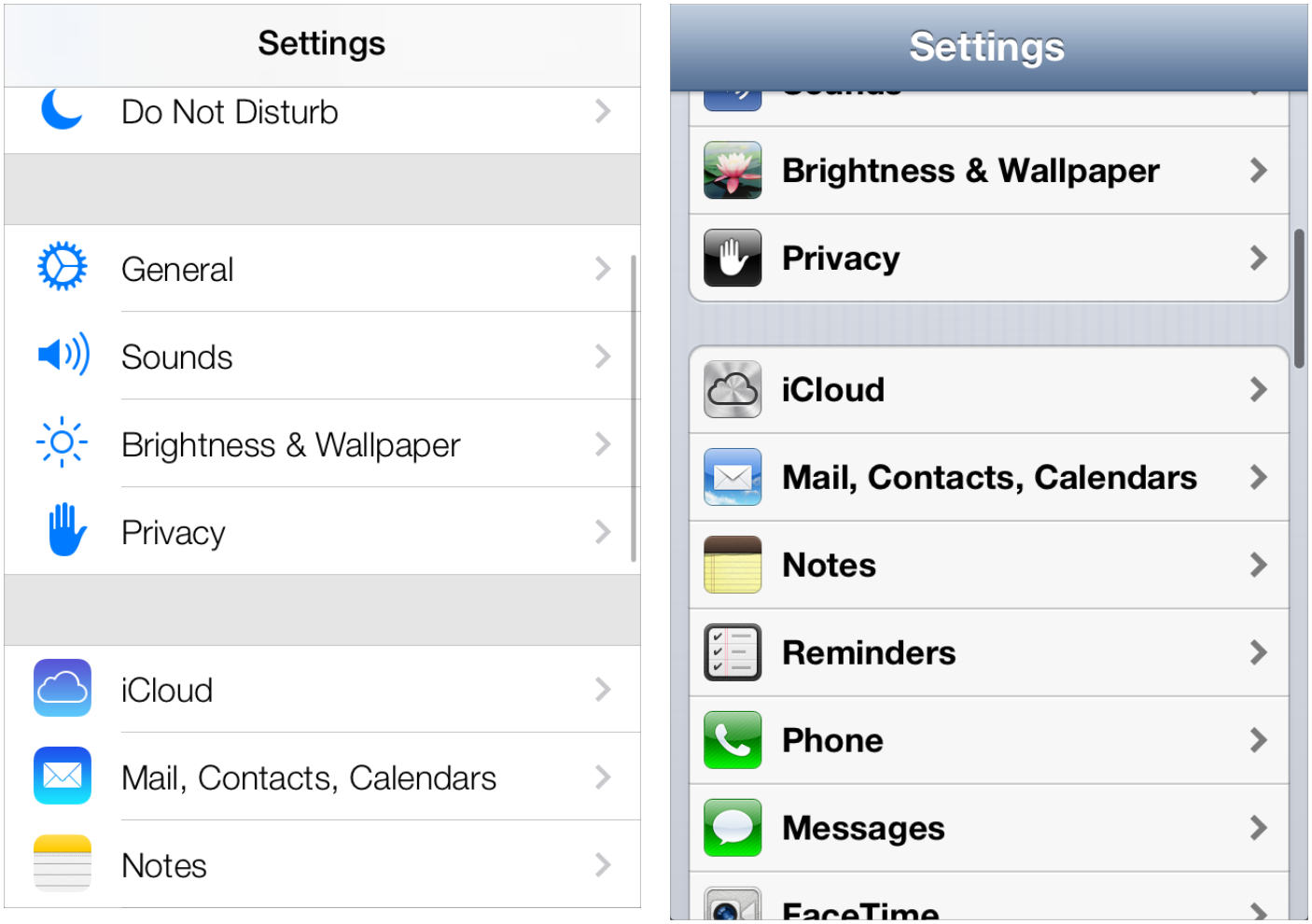


Scroll View

A scroll view helps people see content that is larger than the scroll view's boundaries.

The only visual difference in the scroll view between iOS 7 and iOS 6 is the appearance of the scroller.

iOS 7 iOS 6



In iOS 7, you can manage scroll view insets yourself by using the `automaticallyAdjustsScrollViewInsets` property of `UIViewController`.

Split View Controller (iPad Only)

A split view controller is a full-screen view controller that manages the presentation of two side-by-side view controllers.

The appearance of a split view controller is provided by the views of its child view controllers, which may be affected by the iOS 7 UI.

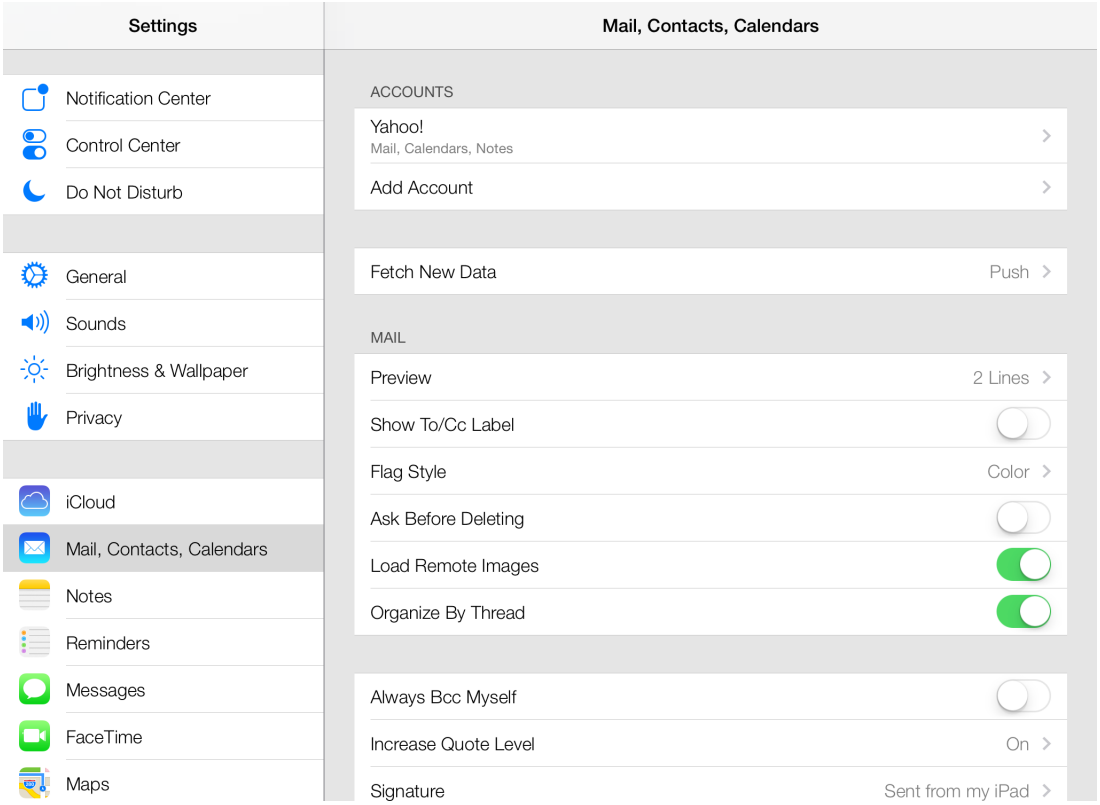
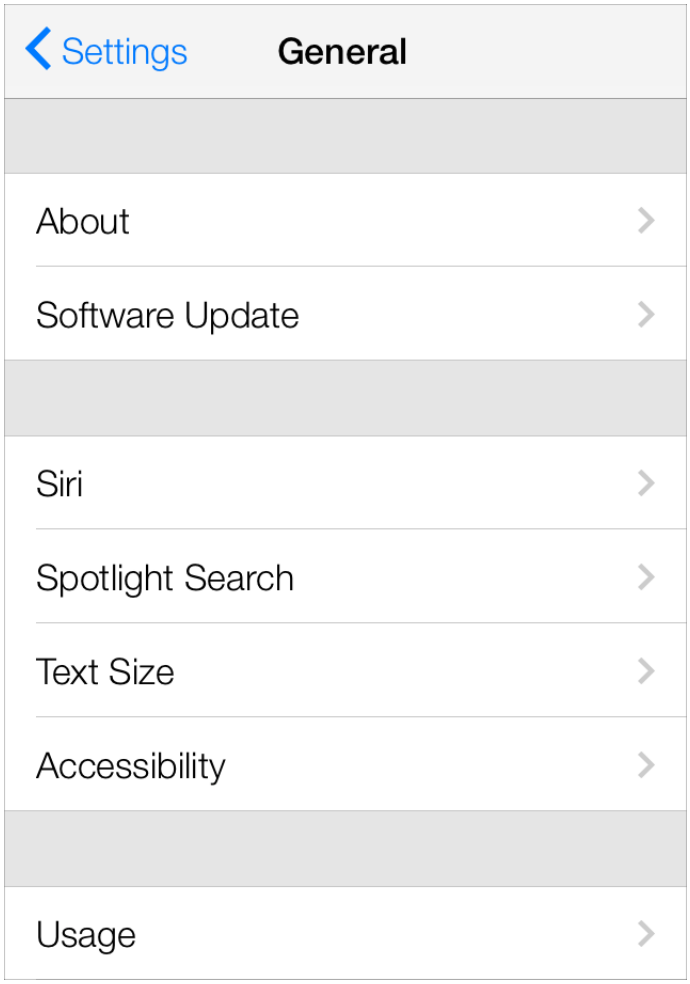


Table View

A table view presents data in a single-column list of multiple rows.

iOS 7 introduces several changes to the appearances of both plain and grouped table views.

iOS 7 (grouped table)



iOS 6 (grouped table)

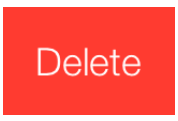
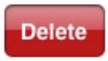
Feature	Appearance in iOS 7
Cell separator (both styles)	Separators in iOS 7 are thinner, narrower, and lighter in color than separators in iOS 6. By default, the separator is inset from the left edge of the table view. *
Section index (plain style)	By default, the section index includes a translucent white background (use the `sectionIndexBackgroundColor` property to specify a different color).
Header and footer text (both styles)	By default, headers display text using all capital letters; footers display text using sentence-style capitalization. You can use a custom `UITableViewHeaderFooterView` object to provide a different look.

Feature	Appearance in iOS 7
Cell group (grouped style)	Each group extends the full width of the screen.
Cell selection appearance (both styles)	Noninverted content is displayed on a subtle gray background.
Background appearance (grouped style)	The background is a solid light gray.

* If every cell in a table contains an image view of the same size, by default iOS vertically aligns the leading edge of all separators. In a table that mixes text-only cells with cells that contain image views, you can use the `separatorInset` property to ensure that the separators are vertically aligned.

> **Note:** The `separatorInset` property is of type `UIEdgeInsets`. By default, `UIEdgeInsets` uses `UITableViewAutomaticDimension` as the value for the `top`, `left`, `bottom`, and `right` parameters.

Table-view elements also have a different appearance in iOS 7.

Table-view element	Appearance in iOS 7	Appearance in iOS 6
Checkmark		
Disclosure indicator		
Detail Disclosure button		
Row reorder		
Row insert		
Delete button control		
Delete button		

Text View

A text view accepts and displays multiple lines of text.

Be sure to use the `UIFont` method `preferredFontForTextStyle` to get the text for display in a text view.

Web View

A web view is a region that can display rich HTML content.

In iOS 7, `UIWebView` supports displaying content in a paginated layout.

Controls

Controls are UI elements that users either view (to get information) or interact with (to perform an action). All iOS 7 controls have an updated appearance, and most of them also have different metrics.

Because `UIControl` inherits from `UIView`, you can use a control's `tintColor` property to tint the control. For more information about tinting views, see "Using Tint Color" (page 18).

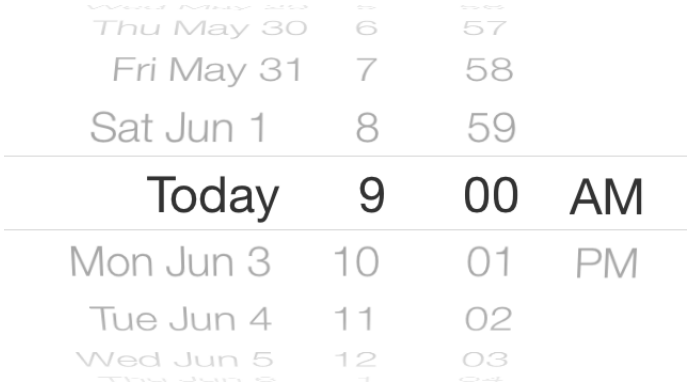
By default, system-provided controls support system-defined animations and appearance changes that indicate highlighted and selected states.

Date Picker

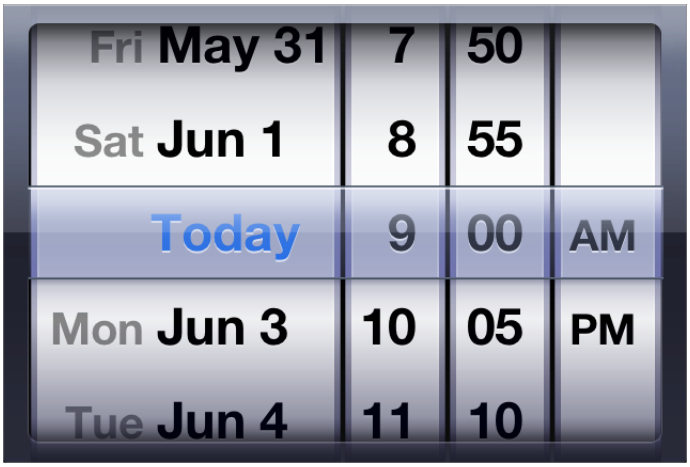
A date picker displays components of date and time, such as minutes, hours, days, and years.

The overall size of a date picker is the same in iOS 7 as it is in iOS 6, but the appearance is very different.

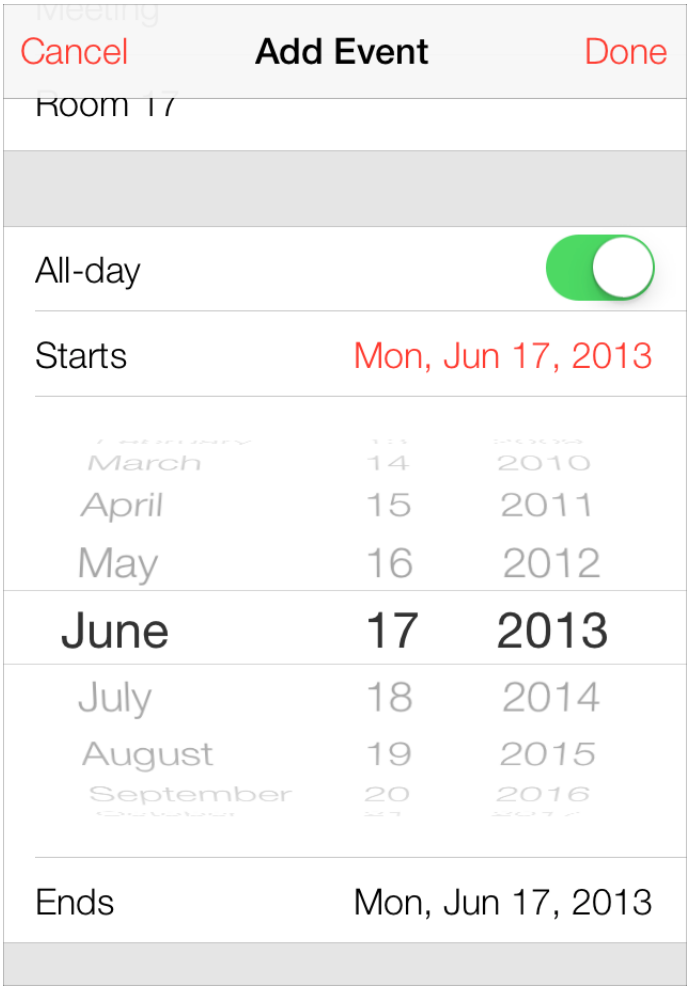
iOS 7



iOS 6



iOS 7 apps tend to embed date pickers within the content instead of displaying them in a different view. For example, Calendar dynamically expands a table row to let users specify a time without leaving the event-creation view:



Contact Add Button

A Contact Add button—that is, a `UIButton` object of type `UIButtonTypeContactAdd`—lets users add a contact's information to a text field or other text-based view.

The size and appearance of a Contact Add button have changed in iOS 7.

iOS 7	iOS 6
	

Detail Disclosure Button

A Detail Disclosure button—that is, a `UIButton` object of type `UIButtonTypeDetailDisclosure`—reveals additional details or functionality related to an item in a table or other view. In iOS 7, a Detail Disclosure button uses the same symbol as the Info button.

The size and appearance of a Detail Disclosure button has changed in iOS 7:

iOS 7 iOS 6


When a Detail Disclosure button appears in a table row, tapping elsewhere in the row doesn't activate the button; instead, it selects the row item or triggers app-defined behavior.

Info Button

An Info button—that is, a `UIButton` object of type `UIButtonTypeInfoLight` or `UIButtonTypeInfoDark`—reveals configuration details about an app, sometimes on the back of the current view. In iOS 7, an Info button uses the same symbol as a Detail Disclosure button.

The size and appearance of an Info button have changed in iOS 7.

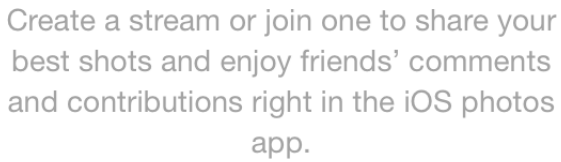
iOS 7 iOS 6 (shown in Weather)


Label

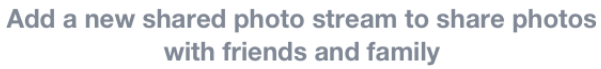
A label displays static text.

By default a label uses the system font, so it looks different in iOS 7 than it does in iOS 6.

iOS 7



iOS 6



Be sure to use the `UIFont` method `preferredFontForTextStyle` to get the text for display in a label.

Page Control

A page control indicates how many views are open and which one is currently visible.

The size and appearance of a page control have changed in iOS 7.

iOS 7 (control shown in Weather)

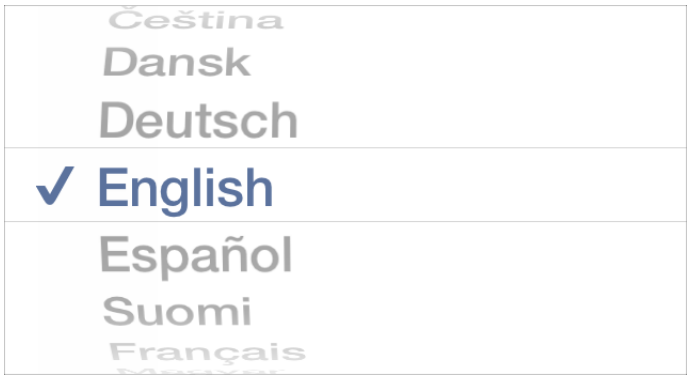
iOS 6 (control shown in Weather)

Picker

A picker displays a set of values from which a user can choose.

The overall size of a picker is the same in iOS 7 as it is in iOS 6; its appearance and behavior in iOS 7 are similar to the appearance and behavior of a date and time picker.

iOS 7



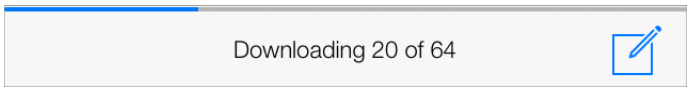
iOS 6

Progress View

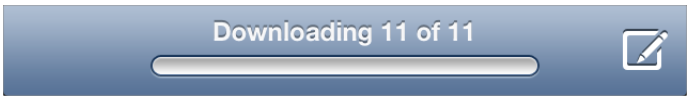
A progress view shows the progress of a task or process that has a known duration.

The size and appearance of a progress view—shown here in the Mail toolbar—have changed in iOS 7.

iOS 7



iOS 6



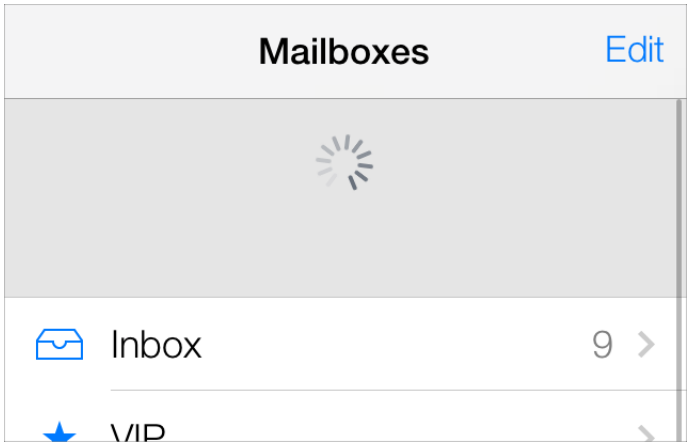
If you currently use the `trackTintColor` property to specify a tint for a progress view's track, the track continues to use this tint in iOS 7. If you set `trackTintColor` to `nil`, the track uses the tint of its parent.

Refresh Control

A refresh control performs a user-initiated content refresh, typically in a table.

The size and appearance of a refresh control have changed in iOS 7.

iOS 7



iOS 6

Rounded Rectangle Button

The rounded rectangle button is deprecated in iOS 7. Instead, use a system button—that is, a `UIButton` object of type `UIButtonTypeSystem`.

iOS 7 system buttons don't include a bezel or a background appearance. A system button can contain a graphical symbol or a text title, and it can specify a tint color or receive its parent's color.

iOS 7 system button	iOS 6 rounded rectangle button
Button	Button

> **Note:** In iOS 7, `UIButtonTypeRoundedRect` has been redefined as `UIButtonTypeSystem`. If an app uses a rounded rectangle button in iOS 6, the system button appearance is applied automatically when it links against iOS 7.

If you need to display a button that includes a bezel, use a button of type `UIButtonTypeCustom` and supply a custom background image.

Segmented Control

A segmented control is a linear set of segments, in which each segment functions as a button that can display a different view.

The size and appearance of the segmented control have changed in iOS 7.

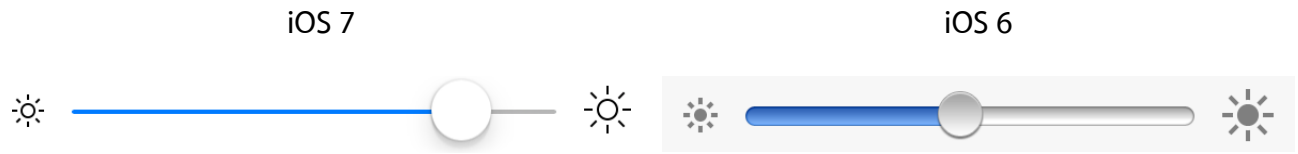
iOS 7	iOS 6
	

In iOS 7, the segmented control uses a single style, and the `segmentedControlStyle` property is unused.

Slider

A slider lets users make adjustments to a value or process throughout a range of allowed values.

The size and appearance of the slider have changed in iOS 7.



iOS 7 continues to use the tints you specify for the minimum and maximum track images and for the thumb, using the properties `minimumTrackTintColor`, `maximumTrackTintColor`, and `thumbTintColor`. If you set the `minimumTrackColor` property to `nil`, the area uses the tint of its parent; if you set the `maximumTrackTintColor` or `thumbTintColor` properties to `nil`, both areas use their default color.

Stepper

A stepper increases or decreases a value by a constant amount.

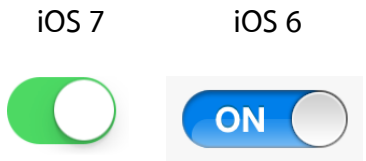
The size and appearance of the stepper have changed in iOS 7.

In iOS 7, by default, a stepper treats custom increment and decrement images as template images.

Switch

A switch presents two mutually exclusive choices or states (typically used only in table views).

The size and appearance of the switch have changed in iOS 7.



iOS 7 continues to use the tints you specify for the on and off—or disabled—states and for the thumb, using the properties `onTintColor`, `tintColor`, and `thumbTintColor`.

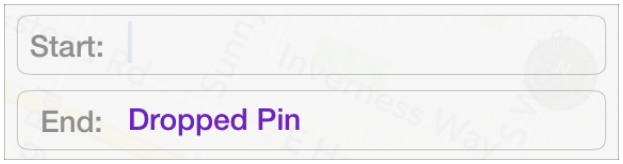
In iOS 7, by default, custom on and off images are ignored.

Text Field

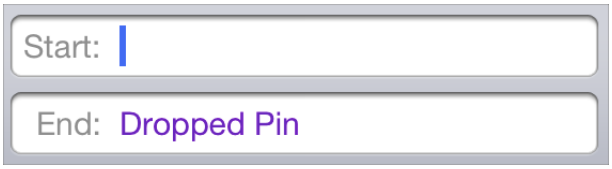
A text field accepts a single line of user input.

The size and appearance of the text field have changed in iOS 7.

iOS 7 (two text fields shown in Maps)



iOS 6 (two text fields shown in Maps)



Be sure to use the `UIFont` method `preferredFontForTextStyle` to get the text for display in a text view.

Temporary Views

Action sheets, alerts, and modal views are temporary views that appear when something requires a user's attention or when additional choices or functionality need to be offered.

Although action sheets and alerts can display custom content, only a few modifications can be made to their appearance. For this reason, you have little to do to make sure these elements look good in your iOS 7 app.

Because a modal view is just a view that's presented modally, you may want to redesign the modal views in your app so that they look appropriate in iOS 7.

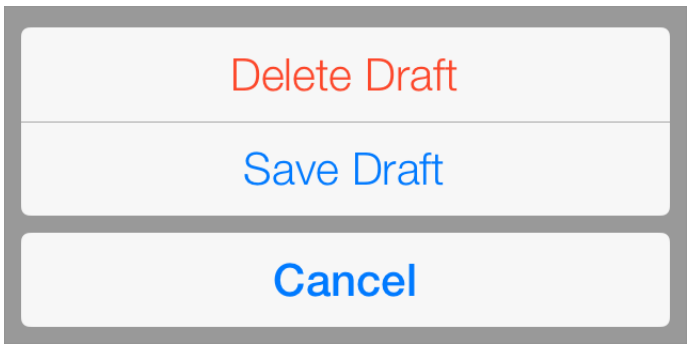
> **Note:** When a temporary view appears, iOS 7 automatically dims the color of the standard views behind it. You may need to adjust your code to handle this color change in custom views; to learn more, see "Using Tint Color" (page 18).

Action Sheet

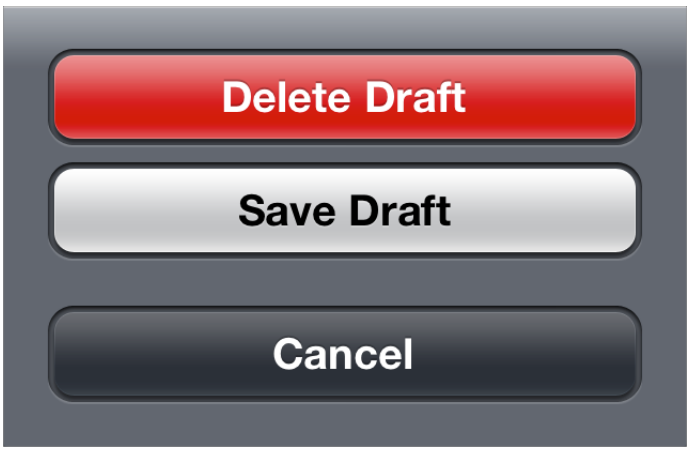
An action sheet displays a set of choices related to a task the user initiates.

In iOS 7, by default an action sheet has a translucent background and contains borderless buttons.

iOS 7



iOS 6



The `UIActionSheetStyle` constants are unused in iOS 7. On an iOS 7 device, system-provided UI—such as an action sheet—uses the iOS 7 appearance regardless of the appearance of the currently running app.

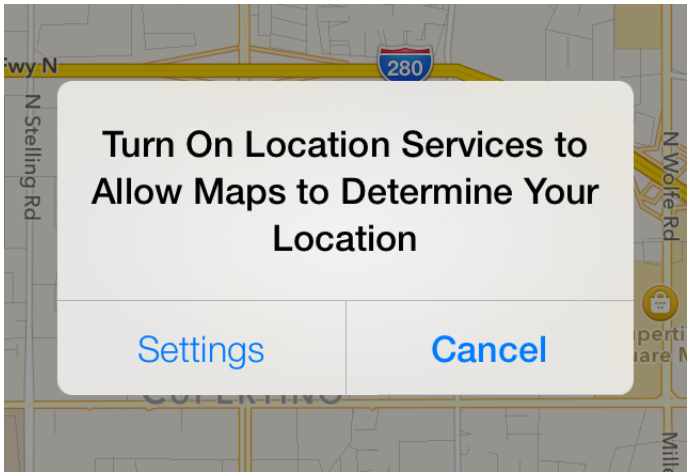
Note that a potentially dangerous option in an action sheet—specified by the `destructiveButtonTitle` parameter in `initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:`—automatically uses the system red color.

Alert

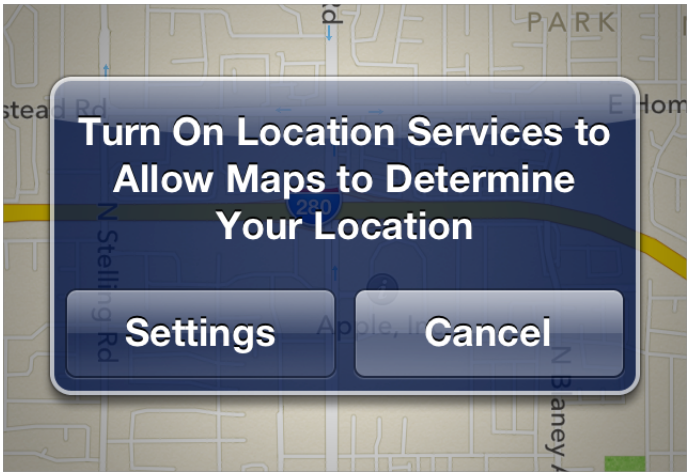
An alert gives people important information that affects their use of the app or the device.

The appearance of an alert has changed in iOS 7.

iOS 7



iOS 6



On an iOS 7 device, system-provided UI—such as an alert—uses the iOS 7 appearance regardless of the appearance of the currently running app.

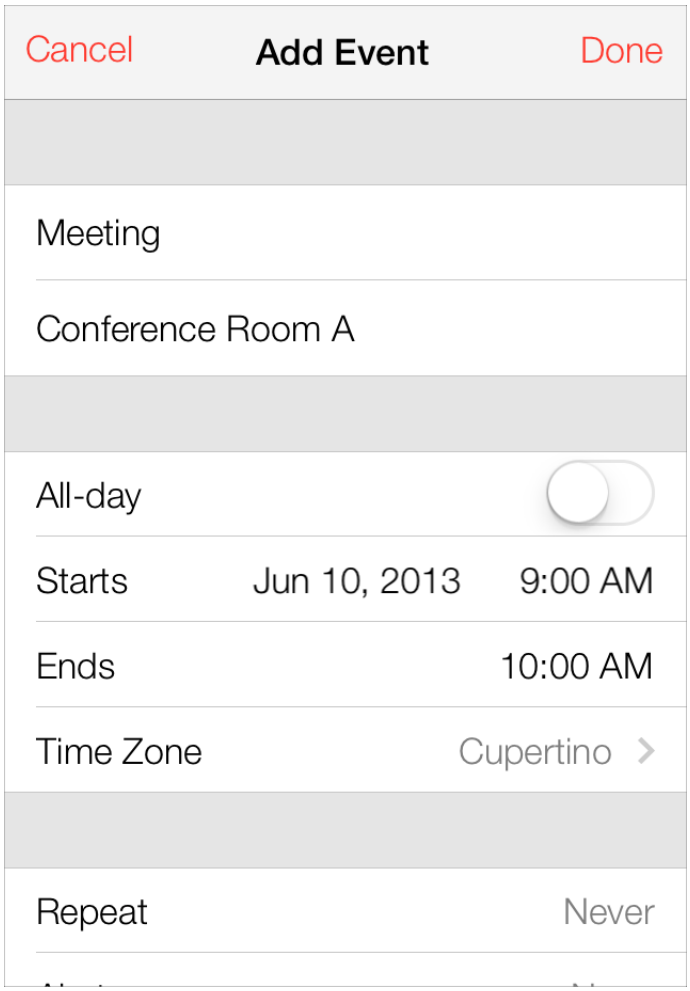
If you supply a third button for an alert, this button is displayed above the two primary buttons at the bottom of the alert.

Modal View

A modal view—that is, a view presented modally—provides self-contained functionality in the context of the current task or workflow.

System-provided modal views use the same appearances as other, similar views in iOS 7.

iOS 7



iOS 6

In iOS 7, you can use a custom animator object and an optional interactive controller object to manage modal presentation. To learn more about custom view controller transitions, see *UIViewControllerAnimatedTransitioning Protocol Reference* and *UIViewControllerInteractiveTransitioning Protocol Reference*.

Document Revision History

This table describes the changes to *iOS 7 UI Transition Guide*.

Date	Notes
2013-07-25	New document that describes how to transition the UI of an iOS 6 app to iOS 7.

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