# UI Elements

Bars

The Status Bar

The *status bar* displays important information about the device and the current environment (shown below on iPhone).

Default (dark) content

mage: ../Art/status_bar_default_2x.png

Light content

mage: ../Art/status_bar_light_2x.png

The status bar:

* Is transparent
* When present, always appears at the upper edge of the screen

API NOTE

You can set the style of the status bar globally for the entire app or you can let individual view controllers set the style as appropriate. To learn more, read[*UIApplication Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/uid/TP40006728) for information about the [UIStatusBarStyle](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/c/tdef/UIStatusBarStyle)constant and [*UIViewController Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/index.html#//apple_ref/doc/uid/TP40006926) for information about the[preferredStatusBarStyle](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/index.html#//apple_ref/occ/instm/UIViewController/preferredStatusBarStyle) property.

**Don’t create a custom status bar.** Users depend on the consistency of the system-provided status bar. Although you might hide the status bar in your app, it’s not appropriate to create custom UI that takes its place.

**Prevent scrolling content from showing through the status bar.** As users scroll, you don’t want them to see a confusing mix of app content and status bar items in the status bar area. To give users the impression of spaciousness while still ensuring maximum readability, make sure the status bar has a background that obscures the content behind it. Here are a few ways to keep scrolling content from showing through the status bar:

* Use a navigation controller to display content. A navigation controller automatically displays a status bar background and it ensures that its content views don’t appear behind the status bar. (To learn more about navigation controllers, see [Navigation Controllers](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/NavigationControllers.html#//apple_ref/doc/uid/TP40011313-CH2).)
* Create a nondistracting custom image—such as a gradient—and display it behind the status bar. To ensure that the image stays behind the status bar, you could use a view controller to keep the image above a scrolling view or you could use a scrolling view to keep it pinned to the top.
* Position content to avoid the status bar area (that is, the area defined by the app’s[statusBarFrame](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/occ/instp/UIApplication/statusBarFrame) property). If you do this, you should use the window’s background color to provide a solid color behind the status bar.

**Avoid putting distracting content behind the status bar.** In particular, you don’t want to imply that users should tap the status bar to access content or activate controls in your app.

**Think twice before permanently hiding the status bar.** Because the status bar is transparent, it’s not usually necessary to hide it. Permanently hiding the status bar means that users must switch away from your app to read the time or to find out whether they have a Wi-Fi connection.

**Consider hiding the status bar—and all other app UI—while people are actively viewing full-screen media.** If you hide the status bar, be sure to let people retrieve it (and the appropriate app UI) with a single tap. Unless you have a compelling reason to do so, avoid defining a custom gesture to redisplay the status bar because users are unlikely to discover such a gesture or to remember it.

**Choose a status bar content color that coordinates with your app.** The default appearance displays dark content, which looks good on top of light-colored app content. The light status bar content looks good on top of dark-colored app content.

**When appropriate, display the network activity indicator.** The network activity indicator can appear in the status bar to show users that lengthy network access is occurring. To learn how to implement this indicator in your code, see [Network Activity Indicator](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW44).

Navigation Bar

A *navigation bar* enables navigation through an information hierarchy and, optionally, management of screen contents.

mage: ../Art/nav_bar_iphone_2x.pngmage: ../Art/nav_bar_ipad_7_2x.png

A navigation bar:

* Is translucent
* Generally appears at the top of an app screen, just below the status bar

In a horizontally regular environment, a navigation bar can also display within a view that doesn’t extend across the screen, such as one pane of a split view controller.

* Can hide when the keyboard appears, the user makes a gesture, or when the containing view controller transitions to a vertically compact environment.
* Can be tinted. (Use [tintColor](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/occ/instp/UIBarButtonItem/tintColor) to tint bar button items; use [barTintColor](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UINavigationBar_Class/index.html#//apple_ref/occ/instp/UINavigationBar/barTintColor) to tint the bar background.)

API NOTE

A navigation bar is contained in a navigation controller, which is a programmatic object that manages the display of a hierarchy of custom views. To learn more about defining a navigation bar in your code, see [Navigation Controllers](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/NavigationControllers.html#//apple_ref/doc/uid/TP40011313-CH2),[*UINavigationController Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UINavigationController_Class/index.html#//apple_ref/doc/uid/TP40006934), and [*UINavigationBar Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UINavigationBar_Class/index.html#//apple_ref/doc/uid/TP40006887).

Use a navigation bar to enable navigation among different views and—if appropriate—to provide a control that manages the items in a view. If you need to provide a larger set of controls and you don’t need to enable navigation, consider using a toolbar instead (to learn more, see[Toolbar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW4)).

When the user goes to a new level in a navigation hierarchy, two things should happen:

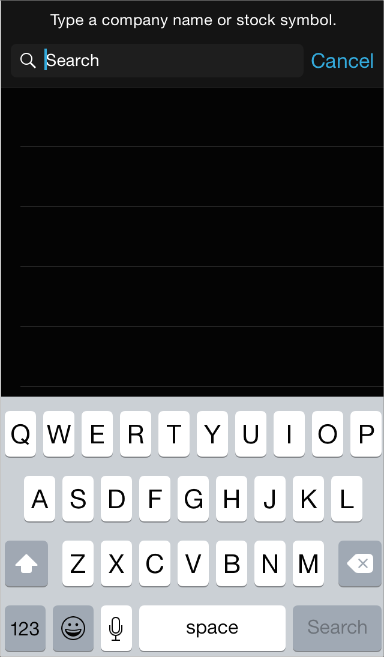
* The navigation bar title should change to the new level’s title, if appropriate.
* A back button should appear in the left end of the bar; it can be labeled with the previous level’s title if it adds value.



**When it adds value, use the title of the current view as the title of the navigation bar.** If titling a navigation bar seems redundant, you can leave the title empty. For example, Notes doesn’t title the current note because the first line of content supplies all the context users need.

**Consider putting a segmented control in a navigation bar at the top level of an app.** This is especially useful if doing so helps to flatten your information hierarchy, making it easier for people to find what they’re looking for. If you use a segmented control in a navigation bar, be sure to choose accurate back-button titles. (For usage guidelines, see [Segmented Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW27).)

**If necessary, consider using a prompt to clarify what users can do in the current screen.** A prompt is a brief sentence that appears near the top of the navigation bar. For example, Stocks uses a prompt to make sure users understand how to find the information they want.



If you need to use a prompt, write a succinct, one-line sentence that uses appropriate ending punctuation.

**Avoid crowding a navigation bar with additional controls, even if it looks like there’s enough space.** In general, a navigation bar should contain no more than the view’s current title, the back button, and one control that manages the view’s contents. If you use a segmented control in the navigation bar, the bar shouldn’t display a title and it shouldn’t contain any controls other than the segmented control.

**Make sure text-titled buttons have enough space between them.** If there isn’t enough space between multiple left or right bar button items in a navigation bar, the text titles can appear to run together, making it difficult for users to distinguish them. If button titles look too close together in your navigation bar, use UIBarButtonSystemItemFixedSpace to add the appropriate spacing between them. (To learn more about this constant, see [*UIBarButtonItem Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/doc/uid/TP40007519).)

**As much as possible, make sure that the look of a customized navigation bar is consistent throughout your app.** For example, don’t combine an opaque navigation bar with a translucent toolbar. Also, it’s best to avoid changing the image, color, or translucency of the navigation bar in different screens in the same orientation.

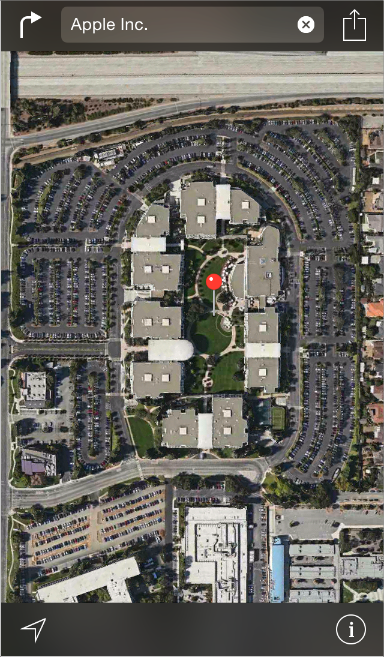
**Make sure that a customized back button still looks and behaves like a back button.** Users know that the standard back button allows them to retrace their steps through a hierarchy of information. If you decide to replace the system-provided chevron with a custom image, be sure to supply a custom mask image, too. iOS uses the mask to make the button title appear to emerge from—or disappear into—the chevron during transitions.

IMPORTANT

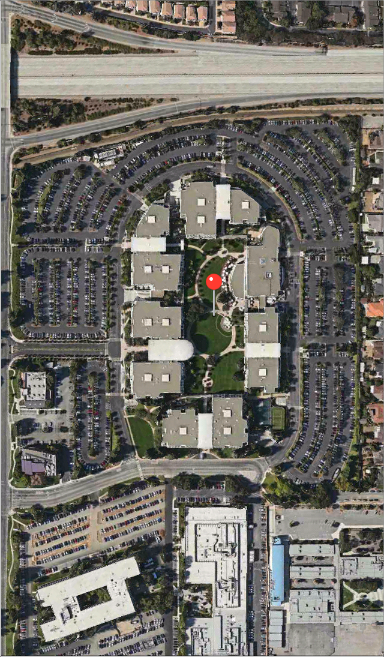
Don’t create a multisegment back button. The back button always takes the user to the current screen’s parent. If you think users might get lost without a multisegment control that displays a type of breadcrumb path, it probably means that you should flatten the information hierarchy.

**Consider hiding the navigation bar when users want to focus on content.** If you do this, be sure to let users restore the navigation bar with a simple gesture, such as a tap.

Maps with navigation bar (and toolbar) visible



Maps with navigation bar (and toolbar) hidden



Toolbar

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

mage: ../Art/mail_toolbar_iphone_2x.pngmage: ../Art/mail_toolbar_ipad_2x.png

A toolbar:

* Is translucent
* Always appears at the bottom edge of a screen or view on iPhone

Can also appear at the top edge of the screen or view on iPad.

* Can hide when the keyboard appears, the user makes a gesture, or when the containing view controller transitions to a vertically compact environment.

API NOTE

A toolbar is typically contained in a navigation controller, which is an object that manages the display of a hierarchy of custom views. To learn more about defining a toolbar in your code, see [Displaying a Navigation Toolbar](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/NavigationControllers.html#//apple_ref/doc/uid/TP40011313-CH2-SW4) and [*UIToolbar Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIToolbar_Class/index.html#//apple_ref/doc/uid/TP40006927).

Use a toolbar to provide a set of actions users can take in the current context.

**Include the most frequently used commands that make sense in the current context.** As much as possible, avoid using a toolbar to provide commands that are only occasionally useful.

**Consider using a segmented control to provide access to different perspectives or modes in the current context.** It’s not a good idea to use a segmented control in a toolbar to show app-level tasks or modes, because a toolbar is specific to the current screen or view. If you need to give people access to primary tasks, views, or modes in your app, use a tab bar instead. To learn more about segmented controls, see [Segmented Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW27); to learn more about tab bars, see [Tab Bar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW52).

**Use icons if you need to put more than three items in a toolbar.** Because text-titled buttons typically use more space than icons, it can be difficult to keep the titles from running together.

**Make sure text-titled buttons have enough space between them.** If there isn’t enough space between two or more buttons in a toolbar, the text titles can appear to run together and users can find it difficult to distinguish between them. If button titles look too close in your toolbar, useUIBarButtonSystemItemFixedSpace to add the appropriate spacing between them. (To learn more about this constant, see [*UIBarButtonItem Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/doc/uid/TP40007519).)

Toolbar and Navigation Bar Buttons

iOS provides many of the standard toolbar and navigation bar buttons that are used in the built-in apps. To learn how to design custom bar icons, see [Bar Button Icons](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/BarIcons.html#//apple_ref/doc/uid/TP40006556-CH21-SW1). Items in the toolbar and navigation bar can be tinted using the [tintColor](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/occ/instp/UIBarButtonItem/tintColor) property.

To find out which symbol names to use to specify the buttons described in Table 37-1, see the documentation for [UIBarButtonSystemItem](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/c/tdef/UIBarButtonSystemItem) in [*UIBarButtonItem Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/doc/uid/TP40007519).

IMPORTANT

As with all standard buttons and icons, it’s essential that you base your usage of a button on its semantic meaning, not on its appearance. This will help your app’s UI make sense even if the button associated with a specific meaning changes its appearance.

| Button | Name | Meaning |
| --- | --- | --- |
| mage: ../Art/UIBarButtonAction_2x.png | Action | Open a modal view that lists system-provided and app-provided actions that can work with the current content. |
| mage: ../Art/UIBarButtonCamera_2x.png | Camera | Open an action sheet that displays a photo picker in camera mode. |
| mage: ../Art/UIBarButtonCompose_2x.png | Compose | Open a new message view in edit mode. |
| mage: ../Art/UIBarButtonBookmarks_2x.png | Bookmarks | Show app-specific bookmarks. |
| mage: ../Art/UIBarButtonSearch_2x.png | Search | Display a search field. |
| mage: ../Art/UIBarButtonAdd_2x.png | Add | Create a new item. |
| mage: ../Art/UIBarButtonTrash_2x.png | Trash | Delete current item. |
| mage: ../Art/UIBarButtonOrganize_2x.png | Organize | Move or route an item to a destination within the app, such as a folder. |
| mage: ../Art/UIBarButtonReply_2x.png | Reply | Send or route an item to another location. |
| mage: ../Art/UIBarButtonRefresh_2x.png | Refresh | Refresh contents (use only when necessary; otherwise, refresh automatically). |
| mage: ../Art/UIBarButtonPlay_2x.png | Play | Begin media playback or slides. |
| mage: ../Art/UIBarButtonFastForward_2x.png | Fast Forward | Fast forward through media playback or slides. |
| mage: ../Art/UIBarButtonPause_2x.png | Pause | Pause media playback or slides (note that this implies context preservation). |
| mage: ../Art/UIBarButtonRewind_2x.png | Rewind | Move backwards through media playback or slides. |
| **Table 37-1**Standard buttons available for toolbars and navigation bars | | |

In addition to the buttons shown in Table 37-1, you can also use the system-provided Edit, Cancel, Save, Done, Redo, and Undo buttons to support editing or other types of content manipulation in your app. The appearance of each of these buttons is provided by its text title. To find out which symbol names to use to specify these buttons, see the documentation for[UIBarButtonSystemItem](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/c/tdef/UIBarButtonSystemItem) in [*UIBarButtonItem Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIBarButtonItem_Class/index.html#//apple_ref/doc/uid/TP40007519).

Finally, you can also use the system-provided Info button in a toolbar:

mage: ../Art/info_button_2x.png

Tab Bar

A *tab bar* gives people the ability to switch between different subtasks, views, or modes in an app.



API NOTE

A tab bar is contained in a tab bar controller, which is an object that manages the display of a set of custom views. To learn more about defining a tab bar in your code, see [Tab Bar Controllers](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/TabBarControllers.html#//apple_ref/doc/uid/TP40011313-CH3) and [UITabBar](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITabBar_Class/index.html#//apple_ref/occ/cl/UITabBar).

A tab bar:

* Is translucent
* Always appears at the bottom edge of the screen
* Displays no more than five tabs at one time in a horizontally compact environment (if there are more tabs, the tab bar displays four of them and adds the More tab, which reveals the additional tabs in a list)
* Maintains the same height in all orientations
* Can display a badge on a tab to communicate app-specific information (a badge is a red oval containing white text and either a number or exclamation point)

Use a tab bar to give users access to different perspectives on the same set of data or different subtasks related to the overall function of your app.

**In general, use a tab bar to organize information at the app level.** A tab bar is well suited for use in the main app view because it’s a good way to flatten your information hierarchy and provide access to several peer information categories or modes at one time.

**Don’t use a tab bar to give users controls that act on elements in the current screen or app mode.** If you need to provide controls, including a control that displays a modal view, use a toolbar instead (for usage guidelines, see [Toolbar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW4)).

**Don’t remove a tab when its function is unavailable.** If you remove a tab in some cases but not in others, you make your app’s UI unstable and unpredictable. The best solution is to ensure that all tabs are enabled, but explain why a tab’s content is unavailable. For example, if the user doesn’t have any songs on an iOS device, the Songs tab in the Music app displays a screen that explains how to download songs.

**Consider badging a tab bar icon to communicate unobtrusively.** You can display a badge on a tab bar icon to indicate that there is new information associated with that view or mode.

**In a horizontally regular environment, you might use a tab bar in a popover or the secondary pane of a split view controller.** Do so if the tabs switch or filter the content within that view. However, it often works better to use a segmented control at the bottom edge of a popover or split view controller’s pane, because the appearance of a segmented control coordinates better with the appearance of these UI elements. (For more information on using a segmented control, see [Segmented Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW27).)

**Avoid crowding the tab bar with too many tabs.** Putting too many tabs in a tab bar can make it physically difficult for people to tap the one they want. And with each additional tab you display, you increase the complexity of your app.

**In a horizontally regular environment, avoid creating a More tab.** In an app running in a horizontally regular environment, a screen devoted solely to a list of additional tabs is a poor use of space.

**As much as possible, display the same tabs in every orientation.** It’s best when you can give users a sense of visual stability by providing the same tabs in every orientation. In a horizontally regular environment, you might need to center the same tabs that fill the space in a horizontally compact environment.

Tab Bar Icons

iOS provides the standard icons described in Table 37-2 for use in tab bars. To learn how to design custom tab bar icons, see [Bar Button Icons](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/BarIcons.html#//apple_ref/doc/uid/TP40006556-CH21-SW1). Tab bar icons can be tinted using the[tintColor](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/index.html#//apple_ref/occ/instp/UIView/tintColor) property.

To find out which symbol names to use to specify these icons, see the documentation for[UITabBarSystemItem](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITabBarItem_Class/index.html#//apple_ref/c/tdef/UITabBarSystemItem) in [*UITabBarItem Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITabBarItem_Class/index.html#//apple_ref/doc/uid/TP40006928).

IMPORTANT

As with all standard buttons and icons, it’s essential that you base your usage of an icon on its semantic meaning, not its appearance. This will help your app’s UI make sense even if the icon associated with a specific meaning changes its appearance.

| Icon | Name | Meaning |
| --- | --- | --- |
| mage: ../Art/UITabBarBookmarks_2x.png | Bookmarks | Show app-specific bookmarks. |
| mage: ../Art/UITabBarContacts_2x.png | Contacts | Show contacts. |
| mage: ../Art/UITabBarDownloads_2x.png | Downloads | Show downloads. |
| mage: ../Art/UITabBarFavorites_2x.png | Favorites | Show user-determined favorites. |
| mage: ../Art/UITabBarFeatured_2x.png | Featured | Show content featured by the app. |
| mage: ../Art/UITabBarHistory_2x.png | History | Show history of user actions. |
| mage: ../Art/UITabBarMore_2x.png | More | Show additional tab bar items. |
| mage: ../Art/UITabBarMostRecent_2x.png | Most Recent | Show the most recent item. |
| mage: ../Art/UITabBarMostViewed_2x.png | Most Viewed | Show items most popular with all users. |
| mage: ../Art/UITabBarRecents_2x.png | Recents | Show the items accessed by the user within an app-defined period. |
| mage: ../Art/UITabBarSearch_2x.png | Search | Enter a search mode. |
| mage: ../Art/UITabBarTopRated_2x.png | Top Rated | Show the highest-rated items, as determined by the user. |
| **Table 37-2**Standard icons for use in the tabs of a tab bar | | |

Search Bar

A *search bar* accepts text from users, which can be used as input for a search (shown here with placeholder text).

mage: ../Art/search_bar_2x.png

API NOTE

To learn how to define a search bar in your code, see [UISearchBar](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISearchBar_Class/index.html#//apple_ref/occ/cl/UISearchBar). To learn more about displaying a search bar, see [UISearchDisplayController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISearchDisplayController_Class/index.html#//apple_ref/occ/cl/UISearchDisplayController).

A search bar can display optional elements, such as these:

* *Placeholder text.* This text might state the function of the control (for example, “Search” as shown above) or remind users in what context they are searching (for example, “Google”).
* *The Bookmarks button.* This button can provide a shortcut to information users want to easily find again. For example, the Bookmarks button in the Maps search mode gives access to bookmarked locations, recent searches, and contacts.

mage: ../Art/search_bar_bookmarks_2x.png

The Bookmarks button is visible only when there is no user-supplied or nonplaceholder text in the search bar. When the search bar contains such text, the Clear button appears so that users can erase the text.

* **The Clear button.** Most search bars include a Clear button that lets users erase the contents of the search bar with one tap.

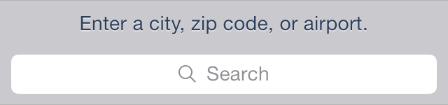
mage: ../Art/search_bar_clear_2x.png

When the search bar contains any nonplaceholder text, the Clear button is visible so users can erase the text. If there is no user-supplied or nonplaceholder text in the search bar, the Clear button is hidden.

* *The results list icon.* This icon indicates the presence of search results. When users tap the results list icon, an app can display the results of their most recent search.

mage: ../Art/search_bar_results_2x.png

* **A prompt.** A descriptive title, called a *prompt*, can be placed above the search bar. A prompt is a short, complete sentence that provides introductory or app-specific context for the search bar.

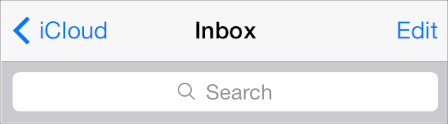


Use a search bar to enable search in your app. Don’t use a text field to enable search because it doesn’t have the standard search bar appearance that users expect.

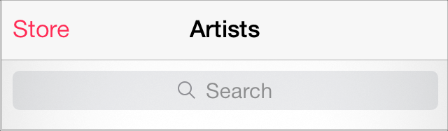
In iOS 8 and later, using [UISearchController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISearchController/index.html#//apple_ref/occ/cl/UISearchController) makes it easy to put a search bar in a navigation bar. Note that when a search controller’s view controller is contained within a navigation controller—as is the case in Mail—the search bar automatically transitions into the navigation bar when users initiate a search.

**Choose a search bar style that complements the importance of search in your app.** If search is a primary function in your app, you may want to use the prominent style; if users don’t need to search very often, you may want to use the minimal style.

The prominent search bar style (shown in Mail)



The minimal search bar style (shown in Music)



Scope Bar

A *scope bar*—which is available only in conjunction with a search bar—helps users define the scope of a search.



API NOTE

To learn more about defining a search bar and scope bar in your code, see[UISearchBar](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISearchBar_Class/index.html#//apple_ref/occ/cl/UISearchBar).

When a search bar is present, a scope bar can appear near it. A scope bar adopts the same appearance that you specify for the search bar.

It can be useful to display a scope bar when there are clearly defined or typical categories in which users might want to search. However, it’s even better to improve search results so that users don’t need to scope their search.

Content Views

Activity

An *activity* represents a system-provided or custom task—accessible through an activity view controller—that can work with the current content.



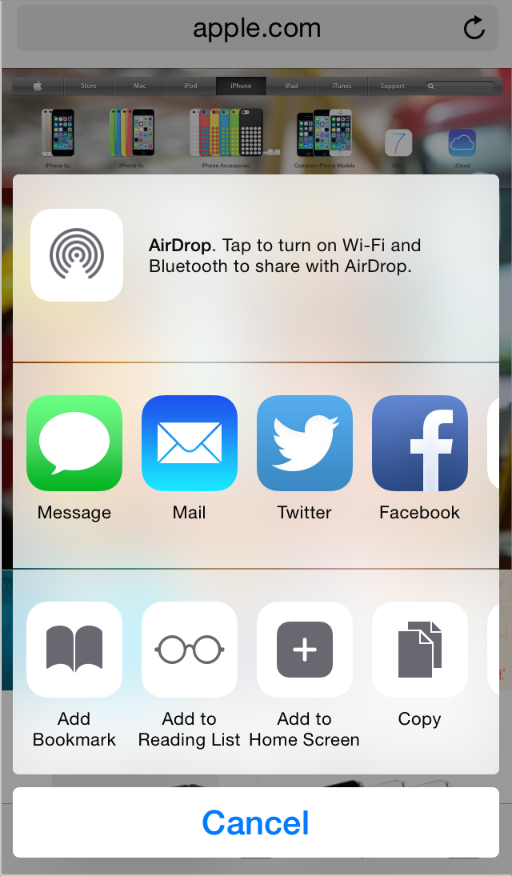
API NOTE

To learn more about defining an activity in your code, see [*UIActivity Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIActivity_Class/index.html#//apple_ref/doc/uid/TP40011974); to learn how to incorporate an activity view controller into your app, see [Activity View Controller](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ContentViews.html#//apple_ref/doc/uid/TP40006556-CH13-SW121).

Action and Share extensions are also displayed in the activity view controller. To learn more about these extensions, see [Share and Action Extensions](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/AppExtensions.html#//apple_ref/doc/uid/TP40006556-CH67-SW3).

An *activity*:

* Is a customizable object representing a task that an app can perform while users are in the app
* Is represented by an icon that looks similar to a bar button icon



Users initiate an activity by tapping its icon in the activity view controller. In response, the activity either performs the task immediately, or if the task is complicated, it can request more information before performing the task.

Use an activity to give users access to a custom service or task that your app can perform. Note that iOS provides several built-in activities and app extensions, such as Print, Twitter, Message, and AirPlay. You don’t need to create a custom activity that performs a built-in task.

**Create a streamlined template image that represents your task.** A template image is an image that iOS uses as a mask to create the final icon that users see. To create a template image that looks good in the final icon, follow these guidelines:

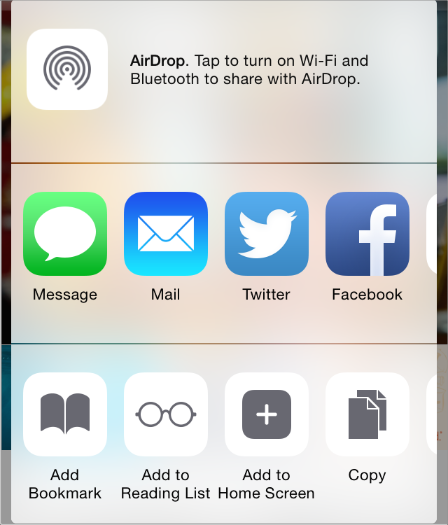
* Use black or white with appropriate alpha transparency.
* Don’t include a drop shadow.
* Use antialiasing.

An activity template image should be centered in an area that measures about 70 x 70 pixels (high resolution).

**Craft an activity title that succinctly describes your task.** The title is displayed below the activity’s icon in the activity view controller. A short title is best, because it looks better onscreen and it’s easier to localize. When a title is too long, iOS first shrinks the text and then—if the title is still too long—truncates it. In general, it’s a good idea to avoid including your company or product name in the activity title.

Activity View Controller

An *activity view controller* presents a transient view listing system-provided and custom tasks that can act on some specified content.



API NOTE

To learn more about defining an activity view controller in your code, see[*UIActivityViewController Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIActivityViewController_Class/index.html#//apple_ref/doc/uid/TP40011976); to learn how to design an activity that provides a custom task, see [Activity](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ContentViews.html#//apple_ref/doc/uid/TP40006556-CH13-SW122).

An activity view controller:

* Displays a configurable list of tasks that users can perform on the specified content
* Can appear in an action sheet or a popover, depending on the environment

Use an activity view controller to give people a list of tasks they can perform on content that is specified in some way. The tasks can be system-provided—such as Copy, Twitter, and Print—or custom. A common way to use an activity view controller is to allow users to post selected content to a social media account.

**Don’t create a custom button that reveals an activity view controller.** People are accustomed to accessing system-provided tasks when they tap the Action button. You want to take advantage of this learned behavior and avoid confusing users by providing an alternative way to do the same thing.

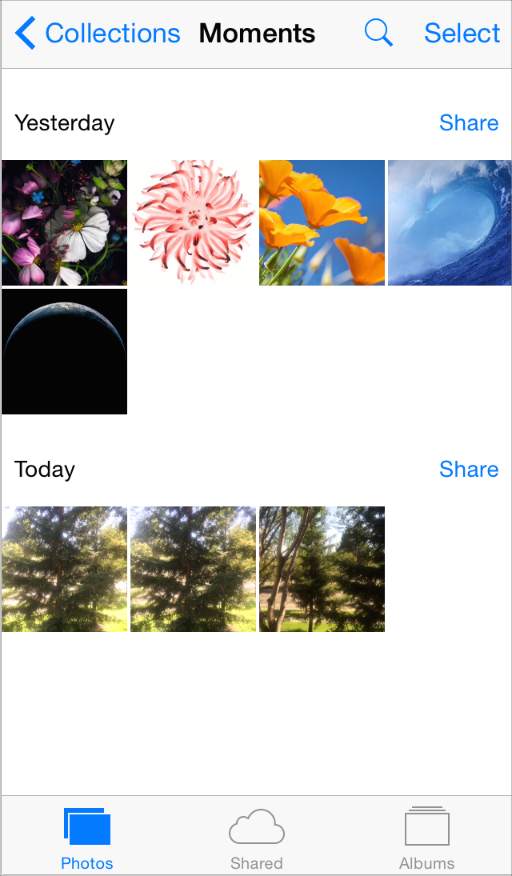
**Ensure that the listed tasks are appropriate in the current context.** You can change the tasks listed in an activity view controller by specifying system-provided tasks to exclude and by identifying custom tasks to include. For example, to prevent users from printing an image, you exclude the Print activity from the activity view controller.

NOTE

You can’t change the order in which the system-provided tasks are listed in an activity view controller. Also, all system-provided tasks appear above any custom tasks.

Collection View

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



API NOTE

To learn more about defining a collection view in your code, see [*Collection View Programming Guide for iOS*](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/CollectionViewPGforIOS/Introduction/Introduction.html#//apple_ref/doc/uid/TP40012334).

A collection view:

* Can contain optional views that visually distinguish subsets of items or provide decorative items, such as custom backgrounds
* Supports custom animated transitions between layouts (by default, a collection view provides animations when users insert, move, or delete items)
* Supports the addition of gesture recognizers to perform custom actions. By default, a collection view recognizes tap (to select an item) and touch-and-hold (to edit an item).

Use a collection view to give users a way to view and manipulate a set of items that don’t need to be displayed in a list. Because a collection view doesn’t enforce a strictly linear layout, it’s particularly well suited to display items that differ in size.

A collection view supports extensive customization, so it’s essential to avoid becoming distracted by your ability to create radical new designs. You want a collection view to enhance the user’s task; you don’t want a collection view to become the focus of the user experience. The following guidelines can help you create collection views that people appreciate.

**Don’t use a collection view when a table view is a better choice.** Sometimes it’s easier for people to view and understand information when it’s presented in a list. For example, it can be simpler and more efficient for people to view and interact with textual information when it’s in a scrolling list.

**Make it easy for people to select an item.** If it’s hard for users to tap an item in your collection view, they’re less likely to enjoy using your app. As with all UI objects that users might want to tap, ensure that the minimum target area for each item in a collection view is 44 x 44 points.

**Use caution if you make dynamic layout changes.** A collection view allows you to change the layout of items while users are viewing and interacting with them. If you decide to dynamically adjust a collection view’s layout, be sure that the change makes sense and is easy for users to track. Changing a collection view’s layout without an obvious motivation can give people the impression that your app is unpredictable and hard to use. And if the current focus or context is lost during a dynamic layout change, users are likely to feel that they’re no longer in control of your app.

Container View Controller

A *container view controller* manages and presents its set of child views—or view controllers—in a custom way. Examples of system-defined container view controllers are tab bar view controllers, navigation view controllers, and split view controllers (you can learn more about these elements in [Tab Bar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW52), [Navigation Bar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW3), and [Split View Controller](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ContentViews.html#//apple_ref/doc/uid/TP40006556-CH13-SW51)).

API NOTE

To learn more about defining a custom container view controller in your code, see[*UIViewController Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/index.html#//apple_ref/doc/uid/TP40006926).

A container view controller has no predefined appearance or behavior.

Use a container view controller to present content through which users navigate in a custom way.

**Ask yourself whether a custom container view controller is really necessary.** Users are comfortable with the appearance and behavior of standard container view controllers, such as split view controllers and tab bar view controllers. You need to be sure that the potential advantages of your custom container view outweigh the fact that users won’t recognize it or instantly know how it works.

**Make sure that your custom container view controller works in both orientations.** It’s important to design a container view controller that gives users a consistent experience in both portrait and landscape.

**In general, avoid flashy view transitions.** When you use storyboards to design a custom view controller, it’s easy to define custom animations for the transitions between content views. But in most cases, flamboyant view transitions distract people from their task and often decrease the aesthetic appeal of your app.

Image View

An *image view* displays one image or an animated series of images.

API NOTE

To learn more about defining an image view in your code, see [UIImageView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImageView_Class/index.html#//apple_ref/occ/cl/UIImageView).

An image view:

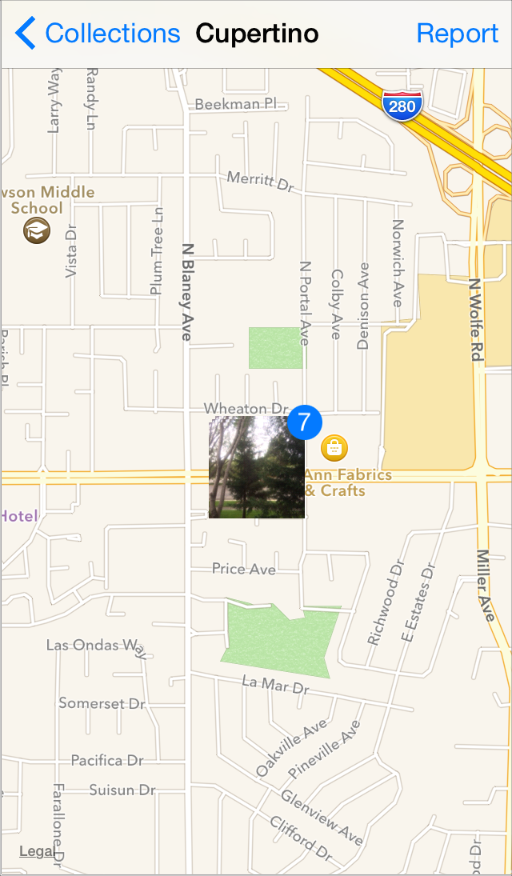
* Has no predefined appearance and it doesn’t enable user interaction by default
* Examines properties of both the image and its parent view to determine whether the image should be stretched, scaled, sized to fit, or pinned to a specific location

In iOS 7 and later, an image view that contains a template image applies the current tint color to the image.

**As much as possible, ensure that all images in an image view have the same size and use the same scale.** If your images have different sizes, the image view will adjust them separately; if your images use different scale factors, they may render incorrectly.

Map View

A *map view* presents geographical data and supports most of the functionality provided by the built-in Maps app (shown below in Photos).



API NOTE

To learn more about defining a map view in your code, see [*MapKit Framework Reference*](https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference/index.html#//apple_ref/doc/uid/TP40008210).

A map view:

* Displays a geographical area using standard map data, satellite imagery, or a combination of both
* Can display annotations (which mark single points) and overlays (which delineate paths or two-dimensional areas)
* Supports both programmatic and user-controlled zooming and panning

Use a map view to give users an interactive view of a geographical area. If you’re developing a routing app, use a map view to display the user’s route (for more information about creating a routing app, see [Routing](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Routing.html#//apple_ref/doc/uid/TP40006556-CH32-SW1)).

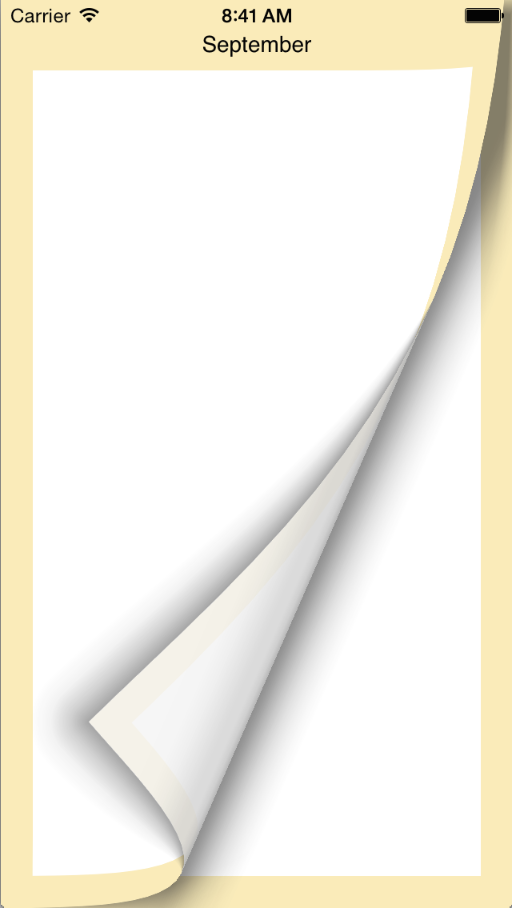
**In general, let users interact with the map.** People are accustomed to interacting with the built-in Maps app, and they expect to be able to interact with your map in similar ways.

**Use the standard pin colors in a consistent way.** A map pin shows the location of a point of interest in your map. People are familiar with the pin colors in the built-in Maps app, so it’s best to avoid redefining the meaning of these colors in your app. When you use the standard pin colors, be sure to use them in the following ways:

* Use red for a destination point
* Use green for a starting point
* Use purple for a user-specified point

Page View Controller

A *page view controller* uses one of two styles to manage transitions through multipage content—scrolling or page-curl. Here’s how a page curl looks in iOS Simulator:



API NOTE

To learn more about defining a page view controller in your code, see [Page View Controllers](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/PageViewControllers.html#//apple_ref/doc/uid/TP40011313-CH4).

A page view controller:

* Has no default appearance for the scrolling style

For the page curl style, a page view controller can add the appearance of the inside of a book spine between pairs of pages

* Animates the transition from one page to another, according to the specified style

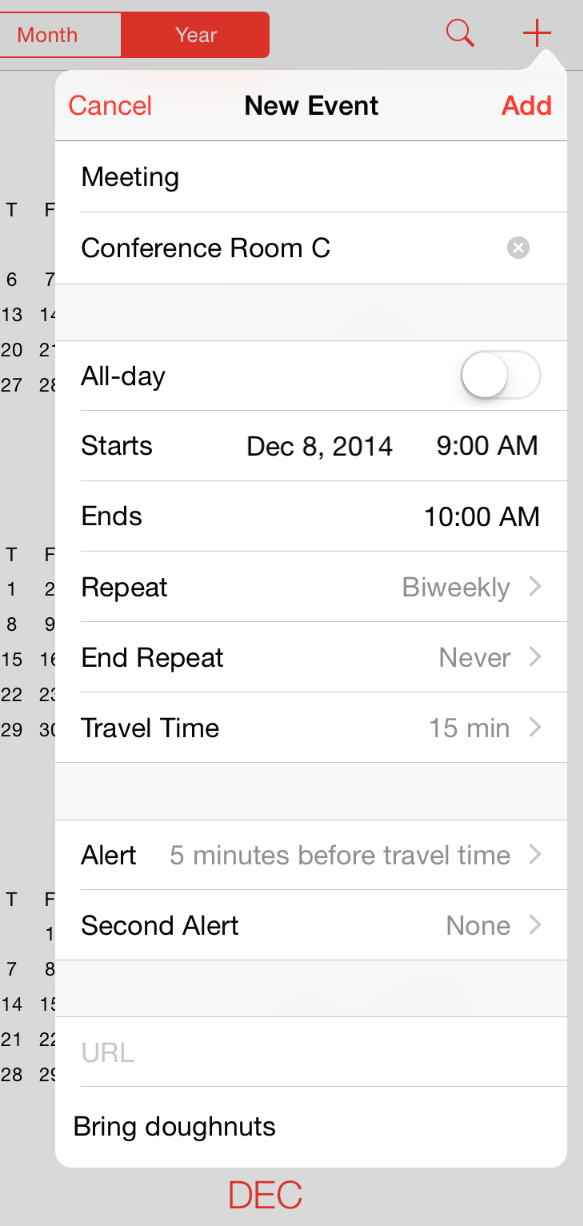
For the scrolling style, the current page scrolls to the next page; for the page curl style, the current page appears to turn like a page in a book or a notepad.

Use a page view controller to present content that users access in a linear fashion (such as the text of a story) or content that naturally breaks into chunks (such as a calendar).

**If necessary, create a custom way to let users access content in a nonlinear way.** A page view controller lets users move from one page to the next or previous page; it doesn’t give users a way to jump among nonadjoining pages. If you want to use a page view controller to present content that users might access in a nonlinear fashion—such as a dictionary or a book’s table of contents—you must implement a custom way to let users move to different areas of the content.

Popover

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



API NOTE

In iOS 8 and later, you use a [UIPopoverPresentationController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPopoverPresentationController_class/index.html#//apple_ref/occ/cl/UIPopoverPresentationController) to present a popover. UIPopoverPresentationController defines a delegate that lets you adjust the display style of your popover content to suit the current display environment. For example, in a horizontally regular environment, your content can display inside a popover; in a horizontally compact environment, your content can display in a full-screen modal view.

A popover:

* Is a self-contained modal view
* In a horizontally regular environment, displays an arrow that indicates the point from which it emerged
* Has a translucent background that blurs content behind it
* Can contain a wide variety of objects and views, such as:
  + Table, image, map, text, web, or custom views
  + Navigation bars, toolbars, or tab bars
  + Controls or objects that act upon objects in the current app view

(By default, table views, navigation bars, and toolbars in a popover use a transparent background to let the popover’s blurring show through.)

In a horizontally regular environment, an action sheet always appears inside a popover.

Use a popover to display additional information or a list of items related to the focused or selected object.

NOTE

The guidelines in this section cover the UI and user experience of a popover that is displayed in a horizontally regular environment. If you’re presenting a popover in a horizontally compact environment where it’s displayed full screen, see [Modal View](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW3) for guidelines that apply to other full-screen modal views.

**Avoid providing a “dismiss popover” button.** A popover should close automatically when its presence is no longer necessary. To determine when a popover’s presence is no longer necessary, consider the following scenarios:

| If a popover… | Do this... |
| --- | --- |
| Provides options that affect the main view, but doesn’t implement an inspector | Close the popover as soon as people make a choice or when they tap anywhere outside its bounds, including the control that reveals the popover. |
| Implements an inspector | Close the popover when people tap anywhere outside its bounds, including the control that reveals the popover.  In this scenario, don’t close the popover as soon as people make a choice, because they might want to make an additional choice or change the attributes of the current choice. |
| Enables a task | Close the popover when people complete or cancel the task by tapping a button in the popover, such as Done or Cancel.  In this scenario, you may not want to close the popover when people tap outside its borders, because it might be important that people finish—or explicitly abandon—the task. Otherwise, save people’s input when they tap outside a popover’s borders, just as you would if they tapped Done. |

**In general, save users’ work when they tap outside a popover’s borders.** Not every popover requires an explicit dismissal, so people might dismiss them mistakenly. Discard the work people do in a popover only if they tap a Cancel button.

**Make the popover arrow point as directly as possible to the element that revealed it.** Doing this helps people remember where the popover came from and what task or object it’s associated with.

**Make sure people can use a popover without seeing the app content behind it.** A popover obscures the content behind it, and people can’t drag a popover to another location.

**Ensure that only one popover is visible onscreen at a time.** You shouldn’t display more than one popover (or custom view designed to look and behave like a popover) at the same time. In particular, you should avoid displaying a cascade or hierarchy of popovers simultaneously, in which one popover emerges from another.

**Don’t display a modal view on top of a popover.** Except for an alert, nothing should be displayed on top of a popover.

**When possible, allow people to close one popover and open a new one with one tap.** This behavior is especially desirable when several different bar buttons each open a popover, because it prevents people from having to make extra taps.

**Avoid making a popover too big.** A popover shouldn’t appear to take over the entire screen. Instead, it should be just big enough to display its contents and still point to the place it came from. The height of a popover is not constrained, so you can use it to display a long list of items. In general, though, you should try to avoid scrolling in a popover that enables a task. Note that the system might adjust both the height and the width of a popover to ensure that it fits well on the screen.

**Use standard UI controls and views within a popover.** In general, popovers look best, and are easier for users to understand, when they contain standard controls and views.

**Make sure a custom popover still looks like a popover.** Although it’s easy to customize many of the visual aspects of a popover by using the [UIPopoverBackgroundView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPopoverBackgroundView_class/index.html#//apple_ref/occ/cl/UIPopoverBackgroundView) APIs, avoid creating a design that people might not recognize as a popover. If you change the appearance of a popover too much, users can’t rely on their prior experience to help them understand how to use it in your app.

**Be cautious if you change a popover’s size while it remains visible.** You might want to change a popover’s size if you use it to display both a minimal and an expanded view of the same information. When you adjust the size of a visible popover, it’s usually a good idea to animate the change because it avoids giving the impression that a new popover has replaced the old one.

Scroll View

A *scroll view* helps people see content that is larger than the scroll view’s boundaries (the image shown below is both taller and wider than the scroll view that contains it).



API NOTE

To learn more about defining a scroll view in your code, see [UIScrollView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIScrollView_Class/index.html#//apple_ref/occ/cl/UIScrollView).

A scroll view:

* Has no predefined appearance
* Flashes transient scroll indicators when it first appears or when users interact with it
* Responds to the speed and direction of gestures to reveal content in a way that feels natural to people

When users drag content in a scroll view, the content follows the touch; when users flick content, the scroll view reveals the content quickly and stops scrolling when the user touches the screen or when the end of the content is reached.

* Can operate in paging mode, in which each drag or flick gesture reveals one app-defined page of content

Use a scroll view to give people access to large views—or to large numbers of views—in a constrained space.

**Support zoom behavior appropriately.** If it makes sense in your app, let users pinch or double-tap to zoom into and out of a scroll view. When you enable zoom, you should also set maximum and minimum scale values that make sense in the context of the user’s task. For example, letting users zoom in on text until one character fills the screen is unlikely to make it easier for them to read the content.

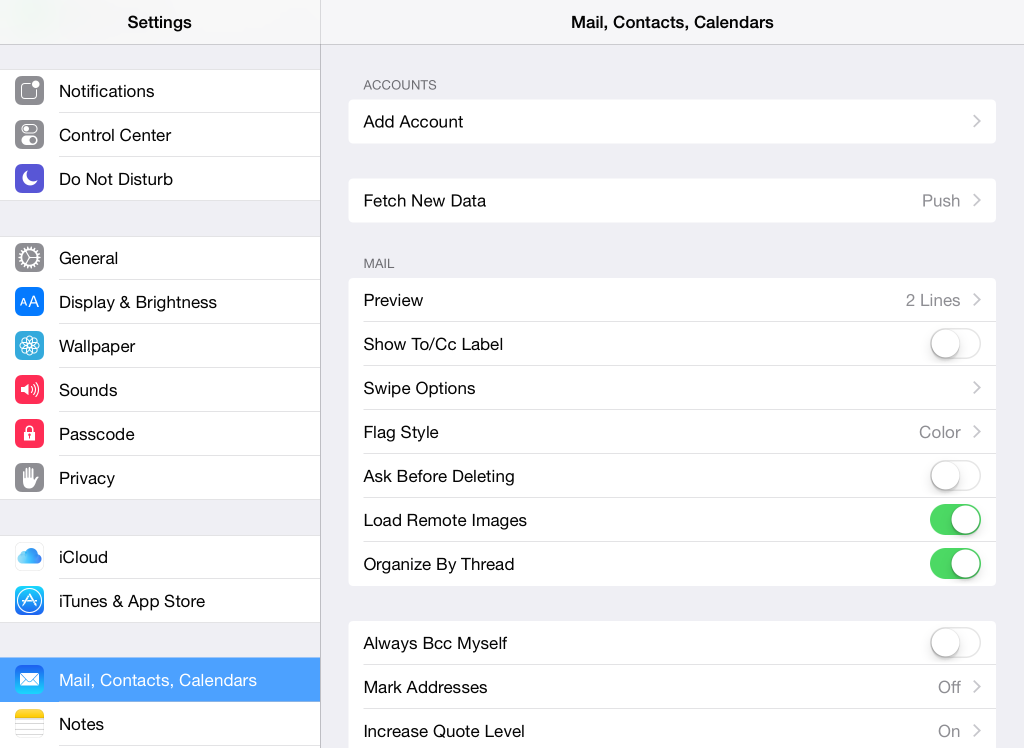
**Consider using a page control with a paging-mode scroll view.** When you want to display content that’s divided into pages, screenfuls, or other chunks, you can use a page control to show users how many chunks are available and which one they’re currently viewing.

When you use a page control with a paging-mode scroll view, it’s a good idea to disable the scroll indicator that’s on the same axis as the page control. Removing the scroll indicator focuses attention on the page control and gives people one unambiguous way to page through the content. For more information about using a page control in your app, see [Page Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW6).

**In general, display only one scroll view at a time.** People often make large swipe gestures when they scroll, so it can be difficult for them to avoid interacting with a neighboring scroll view on the same screen. If you decide to put two scroll views on one screen, consider allowing them to scroll in different directions so that one gesture is less likely to scroll both views. For example, Stocks in portrait orientation on iPhone displays stock quotes in a vertically scrolling view above company-specific information, which is in a horizontally scrolling view.

Split View Controller

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



API NOTE

Each child view controller of a split view controller is responsible for managing the display of one pane. The split view controller itself presents these child view controllers and manages transitions between different orientations. To learn more about defining a split view controller in your code, see [*UISplitViewController Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISplitViewController_class/index.html#//apple_ref/doc/uid/TP40009277) and [Split View Controllers](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/SplitViewControllers.html#//apple_ref/doc/uid/TP40011313-CH7).

In iOS 7 and earlier, split view controller was available only on iPad.

By default, a split view controller uses the current size class to decide how to arrange its child view controllers. For example, a split view controller:

* Tries to display both panes side-by-side in a horizontally regular environment
* Can display the primary pane layered on top of the secondary pane, or can hide the primary pane offscreen until it’s needed, typically in a horizontally compact environment

You can influence the arrangement of panes by asking the split view controller to pay attention to a preferred layout that you specify for a particular display environment.

A split view controller can contain a wide variety of objects and views, such as:

* Table, image, map, text, web, or custom views
* Navigation bars, toolbars, or tab bars

NOTE

Even though the primary pane is often called the *master pane* and the secondary pane is often called the *detail pane*, this relationship is not enforced in code.

Use a split view controller to display persistent information in the primary pane and related details or subordinate information in the secondary pane. In this design pattern, when people select an item in the primary pane, the secondary pane should display the information related to that item. (You’re responsible for making this happen in code.)

**Avoid creating a secondary pane that is narrower than the primary pane.** If the secondary pane is narrower than the primary pane, the split view controller no longer fills the width of the screen and the overall appearance is unbalanced.

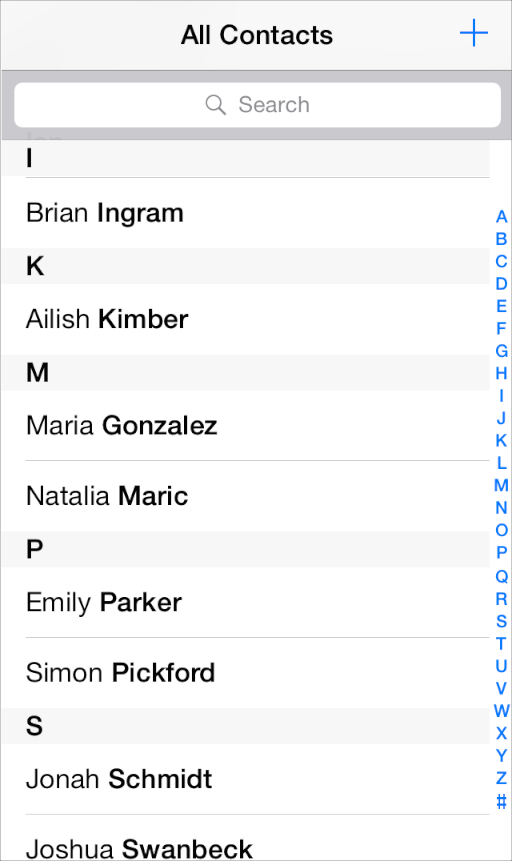
**Avoid displaying a navigation bar in both panes at the same time.** Doing this would make it very difficult for users to discern the relationship between the two panes.

**In general, indicate the current selection in the primary pane in a persistent way.** Even though the content of the secondary pane can change, it should always remain related to the item selected in the primary pane. This viewing experience helps people understand the relationship between the item in the primary pane and the contents of the secondary pane.

**Give people alternative ways to access the primary pane, if appropriate.** By default, only the secondary pane is displayed in a horizontally compact environment and you provide users with a button (typically located in a navigation bar) to reveal and hide the primary pane. The split view controller also supports the swipe gesture to perform the reveal/hide action. Unless your app uses the swipe gesture to perform other functions, you should let people swipe to access the primary pane.

Table View

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



API NOTE

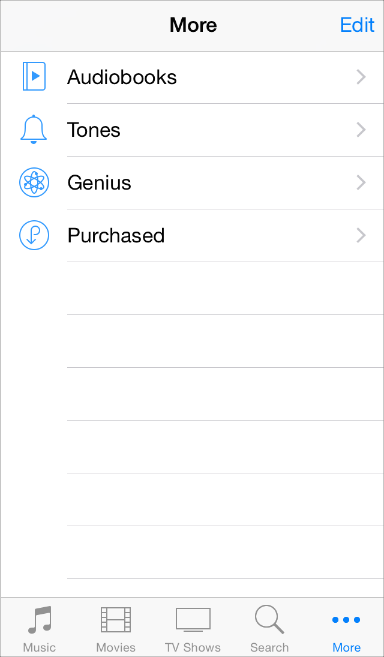
To learn more about defining a table view in your code, see [*Table View Programming Guide for iOS*](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TableView_iPhone/AboutTableViewsiPhone/AboutTableViewsiPhone.html#//apple_ref/doc/uid/TP40007451) and [UITableView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableView_Class/index.html#//apple_ref/occ/cl/UITableView).

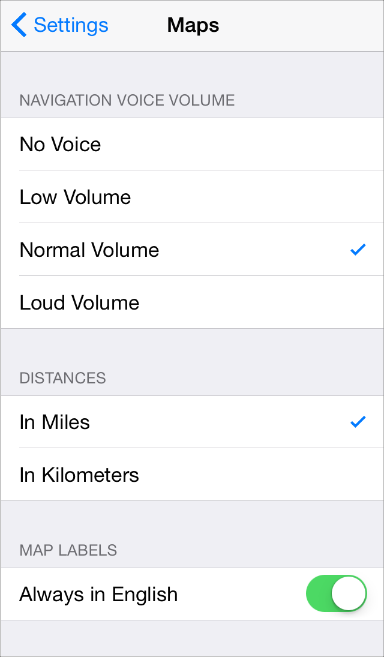
A table view:

* Displays data in rows that can be divided by section or separated into groups
* Provides controls that let users add or remove rows, select multiple rows, see more information about a row item, or reveal another table view

iOS defines two styles of table view:

**Plain.** In the plain style, rows can be separated into labeled sections and an optional index can appear vertically along the right edge of the view. A header can appear before the first item in a section, and a footer can appear after the last item.





**Grouped.** In the grouped style, rows are displayed in groups, which can be preceded by a header and followed by a footer. A grouped table view always contains at least one group of list items—one list item per row—and each group always contains at least one item. A grouped table view doesn’t include an index.

In both styles, a table row becomes highlighted briefly when a user taps a selectable item. If a row selection results in navigation to a new screen, the selected row becomes highlighted briefly as the new screen slides into place. When the user navigates back to the previous screen, the originally selected row again becomes highlighted briefly to remind the user of the earlier selection (it doesn’t remain highlighted).

iOS includes some *table view elements* that can extend the functionality of table views. Unless noted otherwise, these elements are suitable for use with table views only.

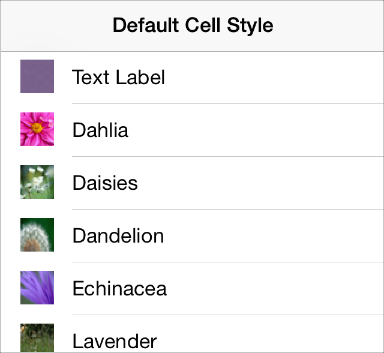
| Table view element | Name | Meaning |
| --- | --- | --- |
| mage: ../Art/check_2x.png | Checkmark | Indicates that the row is selected. |
| mage: ../Art/disclosure_indicator_2x.png | Disclosure indicator | Displays another table associated with the row. |
| mage: ../Art/detail_disclosure_2x.png | Detail Disclosure button | Displays additional details about the row in a new view (for information on how to use this element outside of a table, see [Popover](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ContentViews.html#//apple_ref/doc/uid/TP40006556-CH13-SW19)). |
| mage: ../Art/row_reorder_2x.png | Row reorder | Indicates that the row can be dragged to another location in the table. |
| mage: ../Art/row_insert_2x.png | Row insert | Adds a new row to the table. |
| mage: ../Art/delete_control_2x.png | Delete button control | In an editing context, reveals and hides the Delete button for a row. |
| mage: ../Art/delete_button_2x.png | Delete button | Deletes the row. |

In addition to the table-specific elements listed above, iOS defines the refresh control, which gives users the ability to refresh a table’s contents. To learn more about using a refresh control with a table in your app, see [Refresh Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW131).

iOS defines four table cell styles that implement the most common layouts for table rows in both plain and grouped tables. Each cell style is best suited to display a different type of information.

NOTE

Programmatically, these styles are applied to a table view’s cell, which is an object that tells the table how to draw its rows.

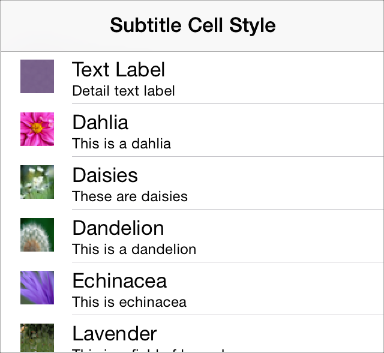


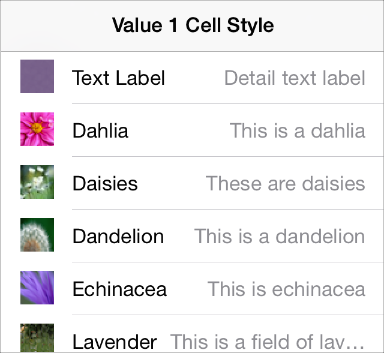
**Default** ([UITableViewCellStyleDefault](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/index.html#//apple_ref/c/econst/UITableViewCellStyleDefault)). The default cell style includes an optional image in the left end of the row, followed by a left-aligned title.

The default style is good for displaying a list of items that don’t need to be differentiated by supplementary information.

**Subtitle** ([UITableViewCellStyleSubtitle](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/index.html#//apple_ref/c/econst/UITableViewCellStyleSubtitle)). The subtitle style includes an optional image in the left end of the row, followed by a left-aligned title on one line and a left-aligned subtitle on the line below.

The left alignment of the text labels makes the list easy to scan. This table cell style works well when list items look similar, because users can use the additional information in the detail text labels to help distinguish items named in the text labels.

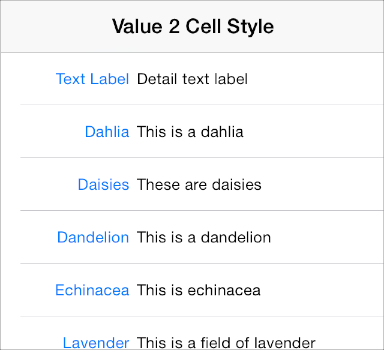




**Value 1** ([UITableViewCellStyleValue1](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/index.html#//apple_ref/c/econst/UITableViewCellStyleValue1)). The value 1 style displays a left-aligned title with, on the same line, a right-aligned subtitle in a lighter font.

**Value 2** ([UITableViewCellStyleValue2](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/index.html#//apple_ref/c/econst/UITableViewCellStyleValue2)). The value 2 style displays a right-aligned title in a blue font, followed on the same line by a left-aligned subtitle in a black font. Images don’t fit well in this style.

In the value 2 layout, the crisp vertical margin between the text and the detail text helps users focus on the first words of the detail text label.



NOTE

All four standard table-cell styles allow the addition of table view elements, such as a checkmark or disclosure indicator. Adding these elements decreases the width of the cell available for the title and subtitle.

Use a table view to display large or small amounts of information cleanly and efficiently. For example:

* **Provide a list of options from which users can select.** You can use the checkmark to show users the currently selected options in the list.

Use either a plain or a grouped table view to display a list of choices that appears when users tap an item in a table row. Use a plain table view to display a list of choices that appears when users tap a button or other UI element that is *not* in a table row.

* **Display hierarchical information.** The plain table style is well suited for displaying a hierarchy of information. Each list item can lead to a different subset of information displayed in another list. Users follow a path through the hierarchy by selecting one item in each successive list. The disclosure indicator tells users that tapping anywhere in the row reveals the subset of information in a new list.
* **Display conceptually grouped information.** Both table view styles allow you to provide context by supplying header and footer views between sections of information.

You can also use a header-footer view—that is, an instance ofUITableViewHeaderFooterView—to display text or a custom view in a header or footer. To learn how to use a header-footer view in your code, see [*UITableViewHeaderFooterView Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewHeaderFooterView_class/index.html#//apple_ref/doc/uid/TP40012241).

Follow these guidelines when you use table views:

**Always provide feedback when users select a list item.** Users expect a table row to become highlighted briefly when they tap a selectable item in it. After tapping, users expect a new view to appear (or the row to display a checkmark) to indicate that the item has been selected or enabled.

**If table content is extensive or complex, avoid waiting until all the data is available before displaying anything.** Instead, fill the onscreen rows with textual data immediately and display more complex data—such as images—as they become available. This technique gives users useful information right away and increases the perceived responsiveness of your app.

**Consider displaying “stale” data while waiting for new data to arrive.** Although this technique isn’t recommended for apps that handle frequently changing data, it can help more static apps give users something useful right away. Before you decide to do this, gauge how often the data changes and how much users depend on seeing fresh data quickly.

**If the data is slow loading or complex, show users that processing is continuing.** If a table contains only complex data, it may be difficult to display anything useful right away. In these rare cases, it's important to avoid displaying empty rows, because empty rows can imply that your app has stalled. Instead, the table should display a spinning activity indicator, along with an informative label (such as “Loading…”) centered in the screen. This behavior reassures users that processing is continuing.

**If appropriate, use a custom title for the Delete button.** If it helps users to better understand the way your app works, you can create a title to replace the system-provided Delete title.

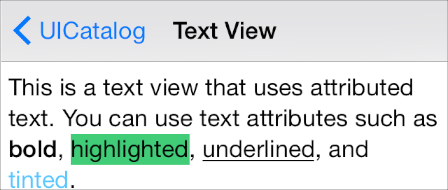
**As much as possible, use succinct text labels to avoid truncation.** Truncated words and phrases can be difficult for users to scan and understand. Text truncation is automatic in all table cell styles, but it can present more or less of a problem, depending on which cell style you use and on where truncation occurs.

**Avoid combining an index with table view elements that are displayed on the right edge of the table.** Table view elements that are displayed on the right edge of the table—such as the disclosure indicator—interfere with the index.

**Create a custom table cell style if you want to lay out your table rows in a nonstandard way.** It’s better to create a custom table cell style than to significantly alter a standard one. To learn how to create your own cells, see [Customizing Cells](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TableView_iPhone/TableViewCells/TableViewCells.html#//apple_ref/doc/uid/TP40007451-CH7-SW18).

Text View

A *text view* accepts and displays multiple lines of attributed text.



API NOTE

To learn more about defining a text view in your code, see [UITextView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextView_Class/index.html#//apple_ref/occ/cl/UITextView).

A text view:

* Is a rectangle of any height
* Supports scrolling when the content is too large to fit inside its bounds
* Supports custom fonts, colors, and alignments (by default, a text view displays left-aligned system font in black)
* Can support editing, in which case a keyboard appears when the user taps inside the text view (the keyboard’s input method and layout are determined by the user’s language settings)

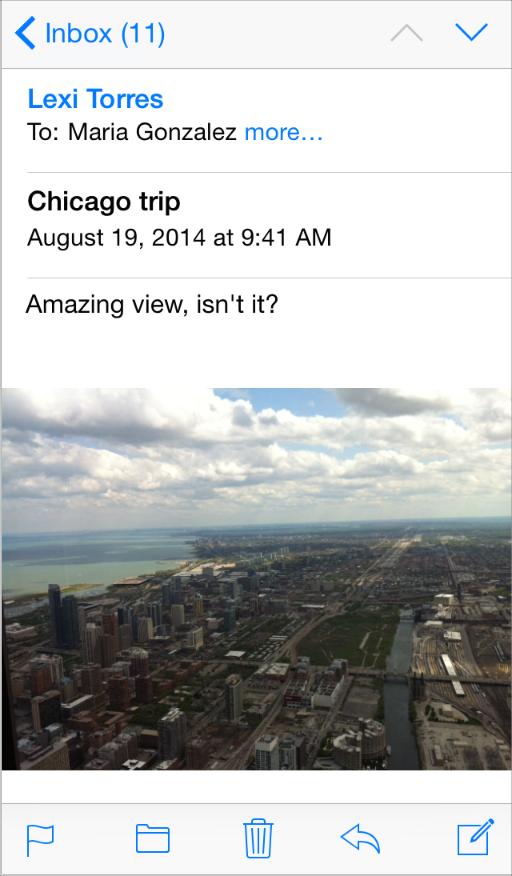
**Always make sure the text is easy to read.** Although you can use attributed strings to combine multiple fonts, colors, and alignments in creative ways, it’s essential to maintain the readability of the text. It’s a good idea to support Dynamic Type and use the [UIFont](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIFont_Class/index.html#//apple_ref/occ/cl/UIFont) methodpreferredFontForTextStyle to get the text for display in a text view. For some guidelines on supporting Dynamic Type, see [Text Should Always Be Legible](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ColorImagesText.html#//apple_ref/doc/uid/TP40006556-CH58-SW3); for programmatic information, see [Text Styles](https://developer.apple.com/library/ios/documentation/StringsTextFonts/Conceptual/TextAndWebiPhoneOS/CustomTextProcessing/CustomTextProcessing.html#//apple_ref/doc/uid/TP40009542-CH4-SW65).

**Specify different keyboard types to accommodate different types of content you expect users to enter.** For example, you might want to make it easy for users to enter a URL, a PIN, or a phone number. Note, however, that you have no control over the keyboard’s input method and layout, which are determined by the user’s language settings.

iOS provides several keyboard types, each designed to facilitate a different type of input. To learn about the keyboard types that are available, see the documentation for [UIKeyboardType](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextInputTraits_Protocol/index.html#//apple_ref/c/tdef/UIKeyboardType). To learn more about managing the keyboard in your app, read [Managing the Keyboard](https://developer.apple.com/library/ios/documentation/StringsTextFonts/Conceptual/TextAndWebiPhoneOS/KeyboardManagement/KeyboardManagement.html#//apple_ref/doc/uid/TP40009542-CH5).

Web View

A *web view* is a region that can display rich HTML content (shown here between the navigation bar and toolbar in Mail on iPhone).



API NOTE

To learn more about defining a web view in your code, see [UIWebView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/index.html#//apple_ref/occ/cl/UIWebView).

A web view:

* Displays web content
* Performs some automatic processing on web content, such as converting a phone number to a phone link

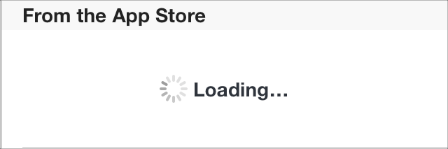
If you have a webpage or web app, you might decide to use a web view to implement a simple iOS app that provides a wrapper for your webpage or web app. If you plan to use a web view to access web content that you control, be sure to read [*Safari Web Content Guide*](https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/Introduction/Introduction.html#//apple_ref/doc/uid/TP40002051).

**Avoid using a web view to create an app that looks and behaves like a mini web browser.**People expect to use Safari on iOS to browse web content, so replicating this broad functionality within your app is not recommended.

Controls

Activity Indicator

An *activity indicator* shows that a task or process is progressing (shown here with text labels).



API NOTE

To learn how to define an activity indicator in your code, see[*UIActivityIndicatorView Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIActivityIndicatorView_Class/index.html#//apple_ref/doc/uid/TP40006830).

An activity indicator:

* Spins while a task is progressing and disappears when the task completes
* Doesn’t allow user interaction

Use an activity indicator in a toolbar or a main view to show that processing is occurring, without suggesting when it will finish.

**Don’t display a stationary activity indicator.** Users associate a stationary activity indicator with a stalled process.

**Use an activity indicator to reassure users that their task or process hasn’t stalled.**Sometimes it’s more important to simply reassure users than to suggest when processing will finish.

**Customize an activity indicator to harmonize with the view it’s in.** If appropriate, coordinate the size and color of an activity indicator with the background of the view it’s in.

Contact Add Button

A *Contact Add button* lets the user add an existing contact to a text field or other text-based view.



API NOTE

To learn how to define a Contact Add button in your code, see [UIButton](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html#//apple_ref/occ/cl/UIButton).

A Contact Add button:

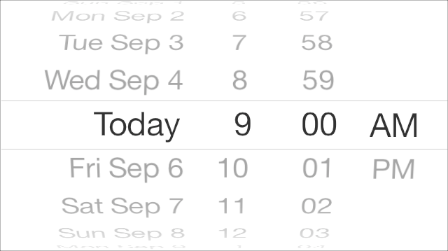
* Displays a list of the user’s contacts
* Helps users add a contact to the view that contains the Contact Add button

Use a Contact Add button to give users an easy way to access a contact without using the keyboard. For example, users can tap the Contact Add button in the To field of the Mail compose view instead of typing a recipient’s name.

Because the Contact Add button functions as an alternative to typing contact information, it’s not appropriate to use the button in a view that doesn’t accept keyboard input.

Date Picker

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



API NOTE

To learn how to define a date picker in your code, see [UIDatePicker](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDatePicker_Class/index.html#//apple_ref/occ/cl/UIDatePicker).

A date picker:

* Displays up to four independent wheels, each of which displays values in a single category, such as month or hour
* Uses dark text to display the current value in the middle of the view
* Can’t be resized (the size of a date picker is the same size as the iPhone keyboard)
* Has four modes, each of which displays wheels containing a set of different values:
  + **Date and time.** The date and time mode (the default mode) displays wheels for the calendar date, hour, and minute values, and an optional wheel for the AM/PM designation.
  + **Time.** The time mode displays wheels for the hour and minute values, and an optional wheel for the AM/PM designation.
  + **Date.** The date mode displays wheels for the month, day, and year values.
  + **Countdown timer.** The countdown timer mode displays wheels for the hour and minute. You can specify the total duration of a countdown, up to a maximum of 23 hours and 59 minutes.

Use a date picker to let users pick—instead of type—a date or time value that consists of multiple parts, such as the day, month, and year.

**As much as possible, display a date picker inline with the content.** It’s best when users can avoid navigating to a different view to use a date picker. In a horizontally regular environment, a date picker can appear within a popover or inline with content.

**If it makes sense in your app, change the interval in the minutes wheel.** By default, a minutes wheel displays 60 values (0 to 59). If you need to display a coarser granularity of choices, you can set a minutes wheel to display a larger minute interval, as long as the interval divides evenly into 60. For example, you might want to display the quarter-hour intervals 0, 15, 30, and 45.

Detail Disclosure Button

A *Detail Disclosure button* reveals additional details or functionality related to an item.

mage: ../Art/detail_disclosure_2x.png

API NOTE

To learn how to define a Detail Disclosure button in your code, see[*UITableViewCell Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/index.html#//apple_ref/doc/uid/TP40006938) and [UIButton](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html#//apple_ref/occ/cl/UIButton).

A Detail Disclosure button reveals a separate view that contains additional information or functionality related to a specific item.

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

Typically, you use a Detail Disclosure button in a table view to give users a way to see more details or functionality related to a list item. However, you can also use this element in other types of views to give users a way to see more information or functionality related to an item in that view.

Info Button

An *Info button* reveals configuration details about an app, sometimes on the back of the current view.

mage: ../Art/info_button_2x.png

API NOTE

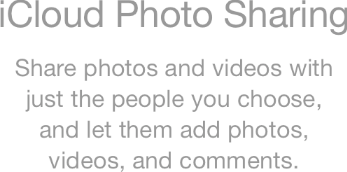
To learn more about defining an Info button in your code, see [UIButton](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html#//apple_ref/occ/cl/UIButton).

iOS includes two styles of Info button: a dark-colored button that looks good on light content and a light-colored button that looks good on dark content.

Use an Info button to reveal configuration details or options about an app. You can use the style of Info button that coordinates best with the UI of your app.

Label

A *label* displays static text.



API NOTE

To learn more about defining labels in your code, see [*UILabel Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UILabel_Class/index.html#//apple_ref/doc/uid/TP40006797).

A label:

* Displays any amount of static text
* Doesn’t allow user interaction except, potentially, to copy the text

Use a label to name or describe parts of your UI or to provide short messages to the user. A label is best suited for displaying a relatively small amount of text.

**Take care to make your labels legible.** It’s best to support Dynamic Type and use the [UIFont](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIFont_Class/index.html#//apple_ref/occ/cl/UIFont)method preferredFontForTextStyle to get the text for display in a label. If you choose to use custom fonts, don’t sacrifice clarity for fancy lettering or showy colors. (For guidelines about using text in an app, see [Color and Typography](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ColorImagesText.html#//apple_ref/doc/uid/TP40006556-CH58-SW1); to learn more about Dynamic Type, see [Text Styles](https://developer.apple.com/library/ios/documentation/StringsTextFonts/Conceptual/TextAndWebiPhoneOS/CustomTextProcessing/CustomTextProcessing.html#//apple_ref/doc/uid/TP40009542-CH4-SW65).)

Network Activity Indicator

A *network activity indicator* appears in the status bar and shows that network activity is occurring.

mage: ../Art/network_activity_indicator_7_2x.png

API NOTE

In your code, use the UIApplication method[networkActivityIndicatorVisible](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/occ/instm/UIApplication/isNetworkActivityIndicatorVisible) to control the indicator’s visibility.

The network activity indicator:

* Spins in the status bar while network activity proceeds and disappears when network activity stops
* Doesn’t allow user interaction

Display the network activity indicator to provide feedback when your app accesses the network for more than a couple of seconds. If the operation finishes sooner than that, you don’t have to show the network activity indicator, because the indicator is likely to disappear before users notice its presence.

Page Control

A *page control* indicates the number of open views and which one is currently visible (shown here in Weather).

mage: ../Art/page_control_weather_2x.png

API NOTE

To learn more about defining a page control in your code, see [UIPageControl](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPageControl_Class/index.html#//apple_ref/occ/cl/UIPageControl).

A page control:

* Displays an indicator dot for each open view in an app (from left to right, the dots represent the order in which the views were opened)
* By default, uses an opaque dot to represent the currently visible view and translucent dots to represent all other open views
* Doesn’t allow users to visit views nonsequentially
* Doesn’t shrink or squeeze together dots as more views are opened (if you try to display more dots than will fit in the view, the dots are clipped)
* Doesn’t enable navigation between views by default; you must implement view-to-view navigation and update the page control’s state appropriately

Use a page control when it’s more important to show users how many views are open than it is to help them choose a specific view. A page control is designed for apps in which each view is a peer of every other view.

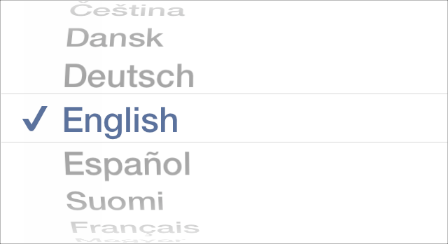
**Don’t use a page control to display views in a hierarchy or other complex arrangement.** A page control doesn’t show how views are related to each other and it doesn’t indicate which view corresponds to each dot, so it can’t help users navigate to a specific view.

**Avoid displaying too many dots.** More than about 10 dots are hard for users to count at a glance and more than about 20 open views are time consuming to visit in sequence. If users can open more than about 20 peer views in your app, consider displaying the views in a different arrangement that provides more information about the views and enables nonsequential navigation.

**Vertically center a page control between an open view’s bottom edge and the screen’s bottom edge.** In this position, a page control is always visible without getting in users’ way.

Picker

A *picker* displays a set of values from which a user picks one.



API NOTE

To learn more about defining a picker in your code, see [*UIPickerView Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerView_Class/index.html#//apple_ref/doc/uid/TP40006842).

A picker:

* Is a generic version of the date picker (for more information about the date picker, see[Date Picker](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW3))
* Displays one or more wheels, each of which contains a list of values
* Uses dark text to display the current value in the middle of the view
* Can’t be resized (the size of a picker is the same size as the iPhone keyboard)

Use a picker to make it easy for people to choose from a set of values.

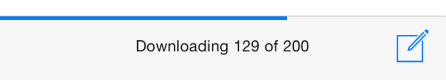
**In general, use a picker when users are familiar with the entire set of values.** Because many of the values are hidden when the wheel is stationary, it’s best when users can predict what the values are. If you need to provide a large set of choices that aren’t well known to your users, a picker might not be the appropriate control.

**As much as possible, display a picker inline with the content.** It’s best when users can avoid navigating to a different view to use a picker.

**Consider using a table view, instead of a picker, if you need to display a very large number of values.** This is because the greater height of a table view makes scrolling faster.

Progress View

A *progress view* shows the progress of a task or process that has a known duration (shown here with the Mail toolbar).



API NOTE

To learn more about defining a progress view in your code, see [*UIProgressView Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIProgressView_Class/index.html#//apple_ref/doc/uid/TP40006782).

A progress view:

* Consists of a track that fills from left to right as the task or process proceeds
* Doesn’t allow user interaction

iOS defines two styles of progress view:

* **Default.** The default style includes an unfilled track appearance so that it can stand alone in a content area.

mage: ../Art/progress_view_default_2x.png

* **Bar.** The bar style doesn’t include the unfilled track appearance because it’s intended to display with a bar, such as a navigation bar or a toolbar.

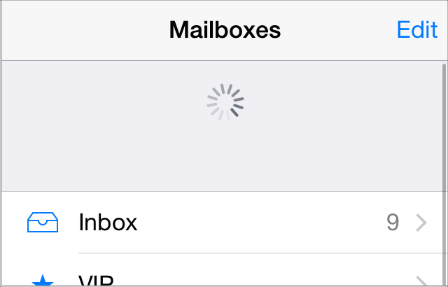
mage: ../Art/progress_view_bar_2x.png

Use a progress view to give feedback on a task that has a well-defined duration, especially when it’s important to indicate approximately how long the task will take.

**If appropriate, coordinate the appearance of a progress view with the style of your app.**By customizing a progress view, you can specify, for example, a custom tint or image for both the track and the fill of a progress view.

Refresh Control

A *refresh control* performs a user-initiated content refresh—typically in a table (shown here above the mailbox list).



API NOTE

To learn more about defining a refresh control in your code, see [*UIRefreshControl Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIRefreshControl_class/index.html#//apple_ref/doc/uid/TP40012250).

A refresh control:

* Looks similar to an activity indicator
* Can display a title
* Is hidden by default until the user initiates a refresh action by dragging down from the top edge of a table

Use a refresh control to give users a consistent way to tell a table or other view to update its contents immediately, without waiting for the next automatic update.

**Don’t stop performing automatic content updates just because you provide a refresh control.** Even though users appreciate being able to request that an update be performed *now*, they still appreciate content that refreshes itself automatically. If you rely on users to initiate all refreshes, users who are unaware of the refresh control are likely to wonder why your app displays stale data. In general, you want to give users the option to refresh contents immediately; you don’t want to make users responsible for every update.

**Supply a short title only if it adds value.** In particular, don’t use the title to describe how to use the refresh control.

Rounded Rectangle Button

The rounded rectangle button is deprecated in iOS 7 and later. Instead, use the system button—that is, a [UIButton](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html#//apple_ref/occ/cl/UIButton) of type UIButtonTypeSystem. For guidelines, see [System Button](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW10).

Segmented Control

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



API NOTE

To learn more about defining a segmented control in your code, see[UISegmentedControl](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISegmentedControl_Class/index.html#//apple_ref/occ/cl/UISegmentedControl).

A segmented control:

* Consists of two or more segments whose widths are proportional, based on the total number of segments
* Can display text or images

Use a segmented control to offer choices that are closely related but mutually exclusive.

**Make sure that each segment is easy to tap.** To maintain a comfortable hit region of 44 x 44 points for each segment, limit the number of segments. On iPhone, a segmented control should have five or fewer segments.

**As much as possible, make the size of each segment’s contents consistent.** Because all segments in a segmented control have equal width, it doesn’t look good if the content fills some segments, but not others.

**Avoid mixing text and images in a single segmented control.** A segmented control can contain text or images. An individual segment can contain either text or an image, but not both. In general, it’s best to avoid putting text in some segments and images in other segments of a single segmented control.

**If necessary, adjust the positioning of content in a customized segmented control.** If you customize the background appearance of a segmented control, make sure that the automatic centering of the control’s content still looks good. Use the bar metrics APIs to adjust the positioning of the content inside a segmented control (to learn more about specifying bar metrics, see the appearance-customization APIs described in [UISegmentedControl](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISegmentedControl_Class/index.html#//apple_ref/occ/cl/UISegmentedControl)).

Slider

A *slider* allows users to make adjustments to a value or process throughout a range of allowed values (shown here with custom images on the left and the right).

mage: ../Art/slider_2x.png

API NOTE

To learn more about defining a slider in your code, see [UISlider](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISlider_Class/index.html#//apple_ref/occ/cl/UISlider).

A slider:

* Consists of a horizontal track and a thumb (a circular control that users can slide)
* Can include optional images that convey the meaning of the left and right values
* Fills the portion of the track between the minimum value (typically on the left) and the thumb

Use a slider to give users fine-grained control over values they can choose or over the operation of the current process.

**If it adds value, create custom appearances for a slider.** For example, you can:

* Define the appearance of the thumb, so that users can see at a glance whether the slider is active
* Supply images to appear at both ends of the slider to help users understand what the slider does

Typically, these custom images correspond to the minimum and maximum values of the value range that the slider controls. A slider that controls image size, for example, could display a very small image at the minimum end and a very large image at the maximum end.

* Define a different appearance for the track, depending on which side of the thumb it is on and which state the control is in

**Don’t use a slider to display a volume control.** If you need to display a volume slider, use the system provided volume slider available when you use the [MPVolumeView](https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MPVolumeView_Class/index.html#//apple_ref/occ/cl/MPVolumeView) class. Note that when the currently active audio output device doesn’t support volume control, the volume slider is replaced by the appropriate device name.

Stepper

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

mage: ../Art/stepper.jpg

API NOTE

To learn more about defining a stepper in your code, see [UIStepper](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIStepper_Class/index.html#//apple_ref/occ/cl/UIStepper).

A stepper:

* Is a two-segment control in which one segment displays a plus symbol and the other segment displays a minus symbol by default
* Supports custom images
* Doesn’t display the value that the user changes

Use a stepper when users might need to make small adjustments to a value.

**Avoid using a stepper when users are likely to make large changes to a value.** It makes sense to use a stepper to set the number of copies in the Printer Options action sheet, because users rarely change this value by very much. On the other hand, it wouldn’t make sense to use a stepper to help users choose a page range, because even a reasonable page range would require a lot of taps.

**Make it obvious which value the stepper affects.** A stepper doesn’t display any values, so you need to make sure that users know which value they’re changing when they use a stepper.

Switch

A *switch* presents two mutually exclusive choices or states.

On

mage: ../Art/switch_on_2x.png

Off

mage: ../Art/switch_off_2x.png

API NOTE

To learn more about defining a switch in your code, see [UISwitch](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISwitch_Class/index.html#//apple_ref/occ/cl/UISwitch).

A switch:

* Indicates the binary state of an item
* Is used in table views only

Use a switch in a table row to give users a way to specify one of two options, such as yes/no or on/off, that govern the state of an item.

You can use a switch control to change the state of other UI elements in the view. Depending on the choice users make, new list items might appear or disappear, or list items might become active or inactive.

System Button

A system button performs an app-specific action.

mage: ../Art/system_button_2x.png

API NOTE

In iOS 7, UIButtonTypeRoundedRect was redefined asUIButtonTypeSystem. To learn more about defining a system button in your code, see [UIButton](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html#//apple_ref/occ/cl/UIButton).

A system button:

* Has no border or background appearance by default
* Can contain an icon or a text title
* Supports custom decoration, such as a border or background image (to add a custom appearance, use a button of type UIButtonTypeCustom and supply a custom background image)

Use a system button to initiate an action. When you supply a title for a system button, follow this approach:

* **Use a verb or verb phrase to describe the action the button performs.** An action-specific title shows users that the button is interactive and tells them what will happen when they tap it.
* **Use title-style capitalization.** Capitalize every word except articles, coordinating conjunctions, and prepositions of four or fewer letters.
* **Avoid creating a title that is too long.** Overly long text gets truncated, which can make it difficult for users to understand it.

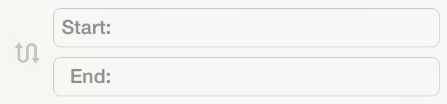


**If appropriate, add a border or background appearance to a system button in a content region.** Most of the time, you can avoid adding ornamentation to a button by crafting a clear call-to-action title, defining a tint, and providing contextual clues. In some content areas, however, it can be appropriate to focus attention on a button by adding a border or background appearance.

In Phone, for example, the bordered number keys reinforce the mental model of making a call and the background of the Call button gives users an eye-catching target that’s easy to hit.

Text Field

A *text field* accepts a single line of user input (shown here with a purpose description and placeholder text).



API NOTE

To learn more about defining a text field and customizing it to display images and buttons, see [UITextField](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextField_Class/index.html#//apple_ref/occ/cl/UITextField).

A text field:

* Is a fixed-height field with rounded corners
* Automatically displays a keyboard when users tap within it
* Can include system provided buttons, such as the Bookmarks button
* Can display text that uses multiple styles (to learn more about this, see [UITextView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextView_Class/index.html#//apple_ref/occ/cl/UITextView))

Use a text field to get a small amount of information from the user.

**Customize a text field if it helps users understand how they should use it.** For example, you can display custom images in the left or right sides of the text field, or you can add a system-provided button, such as the Bookmarks button. In general, you should use the left end of a text field to indicate its purpose and the right end to indicate the presence of additional features, such as bookmarks.

**Display the Clear button in the right end of a text field when appropriate.** When this element is present, tapping it clears the contents of the text field, regardless of any other image you might display over it.

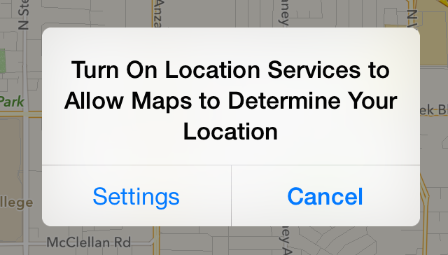
**Display a hint in the text field if it helps users understand its purpose.** A text field can display placeholder text—such as Name (or Address)—when there is no other text in the field.

**Specify a keyboard type that’s appropriate for the type of content you expect users to enter.** For example, you might want to make it easy for users to enter a URL, a PIN, or a phone number. iOS provides several different keyboard types, each designed to facilitate a different type of input. To learn about the keyboard types that are available, see [UIKeyboardType](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextInputTraits_Protocol/index.html#//apple_ref/c/tdef/UIKeyboardType) in[*UITextInputTraits Protocol Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextInputTraits_Protocol/index.html#//apple_ref/doc/uid/TP40006896). To learn more about managing the keyboard in your app, read [Managing the Keyboard](https://developer.apple.com/library/ios/documentation/StringsTextFonts/Conceptual/TextAndWebiPhoneOS/KeyboardManagement/KeyboardManagement.html#//apple_ref/doc/uid/TP40009542-CH5). Note that you have no control over the keyboard’s input method and layout, because these attributes are determined by the user’s language settings.

Temporary Views

Alert

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



API NOTE

To use an alert in your code, you create a [UIAlertController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIAlertController_class/index.html#//apple_ref/occ/cl/UIAlertController) and specify the[UIAlertControllerStyleAlert](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIAlertController_class/index.html#//apple_ref/c/econst/UIAlertControllerStyleAlert).

An alert:

* Displays a required title and an optional message
* Contains one or more buttons

The infrequency with which alerts appear helps users take them seriously. It’s best to minimize the number of alerts your app displays, and make sure each one offers critical information and useful choices.

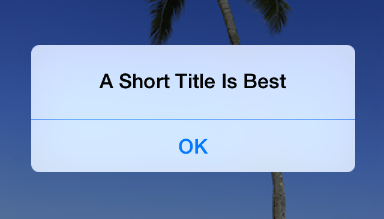
**Avoid creating unnecessary alerts.** In general, alerts are unnecessary in the following scenarios:

| If an alert does this... | Do this instead of using an alert... |
| --- | --- |
| Provides information related to the standard functioning of an app | Design an eye-catching way to display the information, one that harmonizes with the app’s style. |
| Updates users on tasks that are progressing normally | Use a progress view or activity indicator (described in [Progress View](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW7) and [Activity Indicator](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW2)) or integrate status information into the app UI. |
| Asks for confirmation of user-initiated tasks | Use an action sheet (described in [Action Sheet](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW36)). |
| Informs users of problems they can do nothing about | If the problem isn’t critical, integrate the information into the app’s UI; otherwise, use an alert. |

As you read the guidelines for designing alert text, it’s useful to know the following definitions:

* *Title-style capitalization* means that every word is capitalized, except articles, coordinating conjunctions, and prepositions of four or fewer letters when they aren’t the first word.
* *Sentence-style capitalization* means that the first word is capitalized, and the rest of the words are lowercase unless they are proper nouns or proper adjectives.

**Succinctly describe the situation and explain what people can do about it.** Ideally, the text you write gives people enough context to understand why the alert has appeared and to decide which button to tap.



**Keep the title short enough to display on a single line, if possible.** A long alert title is difficult for people to read quickly, and it might get truncated or force the alert message to scroll.

**Avoid single-word titles.** Single-word titles, such as Error or Warning, rarely provide any useful information.

**When possible, use a sentence fragment.** A short, informative statement tends to be easier to understand than a complete sentence.

**As much as possible, write a title that makes it unnecessary to add a message.** For example, you might be able to avoid adding a message if you use a question—or, less frequently, two sentences—for the alert title.

**Avoid sounding accusatory or judgmental when you need to deliver negative news.**People understand that many alerts tell them about problems or warn them about dangerous situations. As long as you use a friendly tone, it’s better to be negative and direct than it is to be positive but oblique.

**As much as possible, avoid “you,” “your,” “me,” and “my.”** Sometimes, text that identifies people directly can be ambiguous and can even be interpreted as insulting or patronizing.

**Use capitalization and punctuation appropriately.** Specifically:

| When the alert title... | Use... |
| --- | --- |
| Is a sentence fragment or a single sentence that is not a question | Title-style capitalization and no ending punctuation |
| Is a single sentence that is a question | Sentence-style capitalization and an ending question mark |
| Consists of two or more sentences | Sentence-style capitalization and appropriate ending punctuation for each sentence |

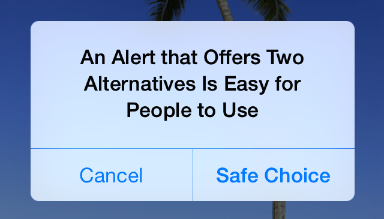
**If you must provide an optional alert message, write a short, complete sentence.** If possible, keep the message short enough to be displayed on one or two lines. If the message is too long, it will scroll, giving users a poor experience. Use sentence-style capitalization and appropriate ending punctuation in the message.



**Avoid lengthening alert text with descriptions of which button to tap.** Ideally, the combination of unambiguous alert text and logical button labels gives people enough information to understand the situation and their choices. If you must provide detailed guidance, follow these guidelines:

* Be sure to use the word “tap” (not “touch” or “click” or “choose”) to describe the selection action.
* Don’t enclose a button title in quotation marks, but do preserve its capitalization.

**Be sure to test the appearance of an alert in both orientations.** Because in landscape the height of an alert is constrained, the alert’s appearance may differ from its appearance in portrait. It’s recommended that you optimize the length of the alert text so that it can be read without scrolling no matter what the orientation.



**Generally, use a two-button alert.** A two-button alert is often the most useful, because it’s easiest for people to choose between two alternatives. A single button alert is less likely to be helpful because it informs people without giving them any control over the situation. An alert that contains three or more buttons is significantly more complex than a two-button alert and should be avoided as much as possible. If you add too many buttons to an alert, it can cause the alert to scroll, which is a bad user experience.

NOTE

If you find that you need to offer people more than two choices, consider using an action sheet instead (to learn how to use an action sheet, see [Action Sheet](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW36)).

**Place buttons appropriately.** Ideally, the button that's most natural to tap should meet two criteria: It should perform the action that users are most likely to want and it should be the least likely to cause problems if a user taps it inadvertently. Specifically:

* When the most likely button performs a nondestructive action, it should be on the right in a two-button alert. The button that cancels this action should be on the left.
* When the most likely button performs a destructive action, it should be on the left in a two-button alert. The button that cancels this action should be on the right.

NOTE

Pressing the Home button while an alert is visible should quit the app, as expected. Doing so should also be identical to tapping the Cancel button—that is, the alert is dismissed and the action isn’t performed.

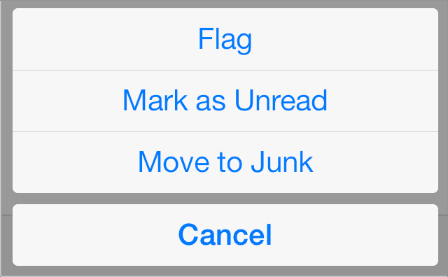
**Give alert buttons short, logical titles.** The best button titles consist of one or two words that describe the result of tapping the button. Follow these guidelines as you create titles for alert buttons:

* As with all button titles, use title-style capitalization and no ending punctuation.
* As much as possible, use verbs and verb phrases that relate directly to the alert text—for example, “Cancel,” “View All,” “Reply,” or “Ignore.”
* Use “OK” for a simple acceptance option if there is no better alternative. Avoid using “Yes” or “No.”
* Avoid “you,” “your,” “me,” and “my” as much as possible. Button titles that use these words are often ambiguous and can appear patronizing.

Action Sheet

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

In a horizontally compact environment, an action sheet emerges from the bottom of the screen



In a horizontally regular environment, an action sheet is always displayed in a popover



API NOTE

To use an action sheet in your code, you create a [UIAlertController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIAlertController_class/index.html#//apple_ref/occ/cl/UIAlertController) and specify the [UIAlertControllerStyleActionSheet](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIAlertController_class/index.html#//apple_ref/c/econst/UIAlertControllerStyleActionSheet).

An action sheet:

* Appears as the result of a user action
* Displays two or more buttons

Use an action sheet to:

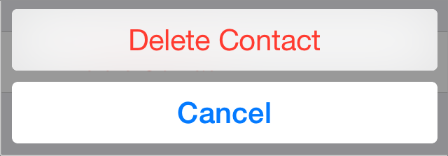
* **Provide alternative ways to complete a task.** An action sheet lets you to provide a range of choices that make sense in the context of the current task, without giving these choices a permanent place in the UI.
* **Get confirmation before completing a potentially dangerous task.** An action sheet prompts users to think about the potentially dangerous effects of the step they’re about to take and gives them some alternatives.

**In a horizontally compact environment, include a Cancel button so that users can easily and safely abandon the task.** Place the Cancel button at the bottom of the action sheet to encourage users to read through all the alternatives before making a choice.

**In a horizontally regular environment, base the way the action sheet is displayed on the way the user initiates the task.** Specifically:

| If the task is initiated from... | Display the action sheet... | Include a Cancel button? |
| --- | --- | --- |
| Outside of a popover | Without animation—that is, the action sheet and the popover appear simultaneously | No, because users can tap outside the popover to dismiss the action sheet |
| Inside a popover | With animation—that is, the action sheet slides up on top of the popover’s content | Yes, because users need to be able to dismiss the action sheet without closing the popover |

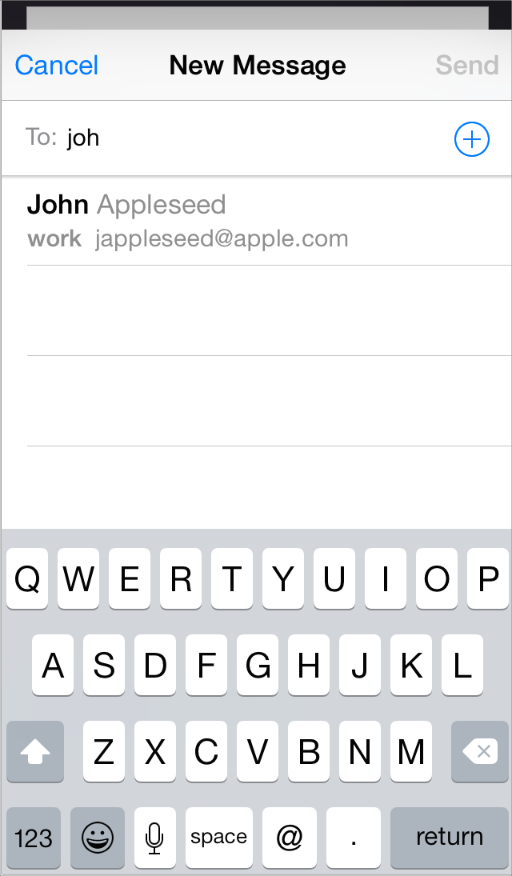
**In all environments, use red for the button that performs a potentially destructive action.**Display a red button at the top of the action sheet, because the closer to the top of the action sheet a button is, the more eye-catching it is.



**Avoid making users scroll an action sheet.** If you include too many buttons in an action sheet, users must scroll to see all their choices. This is a disconcerting experience for users, because they must spend extra time to distinguish the choices. Also, it can be very difficult for users to scroll without inadvertently tapping a button.

Modal View

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



API NOTE

To use a modal view in your code, you create a [UIPresentationController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPresentationController_class/index.html#//apple_ref/occ/cl/UIPresentationController)and specify an appropriate style (for a complete list of styles, see [Modal Presentation Styles](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/index.html#//apple_ref/c/tdef/UIModalPresentationStyle)).

A modal view:

* Can occupy the entire screen, the entire area of a parent view (such as a popover), or a portion of the screen
* Contains the text and controls that are necessary to complete the task
* Usually displays a button that completes the task and dismisses the view and a Cancel button that abandons the task and dismisses the view

Use a modal view when you need to offer the ability to accomplish a self-contained task related to your app’s primary function. A modal view is especially appropriate for a multistep subtask that requires UI elements that don’t belong in the main app UI all the time.

**Choose a modal view style that suits the current task, the current environment, and the visual style of your app.** You can use any of these styles, defined here:

| Modal view style | Appearance | Recommended for |
| --- | --- | --- |
| Full screen | Covers the entire screen. | Presenting a potentially complex task that people can complete within the context of the modal view. |
| Page sheet | In a horizontally regular environment, a style that partially covers the underlying content. All uncovered areas are dimmed to prevent the user from interacting with them.  (In a horizontally compact environment, this style behaves the same as the full screen style.) | Presenting a potentially complex task that people can complete within the context of the modal view. |
| Form sheet | In a horizontally regular environment, a style that displays the content centered in the screen. All uncovered areas are dimmed to prevent the user from interacting with them. In some cases, the position of the modal view is adjusted when a keyboard is present.  (In a horizontally compact environment, this style behaves the same as the full screen style.) | Gathering structured information from the user. |
| Current context | Uses the same size as its parent view. | Displaying modal content within a split view pane, popover, or other non–full–screen view. |

**Don’t display a modal view on top of a popover.** With the possible exception of an alert, nothing should display on top of a popover. In rare cases when you might need to display a modal view as a result of an action the user takes in a popover, close the popover before you open the modal view.

**Coordinate the overall look of a modal view with the appearance of your app.** For example, a modal view often includes a navigation bar that contains a title and buttons that cancel or complete the modal view’s task. When this is the case, the navigation bar should use the same appearance as the navigation bar in the app.

**Display a title that identifies the task, if appropriate.** You might also display text in other areas of the view that more fully describes the task or provides some guidance.

**Choose an appropriate transition style for revealing the modal view.** Use a style that coordinates with your app and enhances the user’s awareness of the temporary context shift that the modal view represents. To do this, you can specify one of the following transition styles:

* **Vertical.** In the vertical style, the modal view slides up from the bottom edge of the screen and slides back down when dismissed (this is the default transition style).
* **Flip.** In the flip style, the current view flips horizontally from right to left to reveal the modal view. Visually, the modal view looks as if it is the back of the current view. When the modal view is dismissed, it flips horizontally from left to right, revealing the previous view.

**If you vary the transition styles for modal views in an app, do so in a way that makes sense to users.** Users are quick to notice behavioral differences in an app and will assume that they mean something. It’s best to establish a logical, consistent pattern that users can easily detect and remember, and avoid changing transition styles without a good reason.