# UI Design Basics

Designing for iOS

iOS embodies the following themes:

* **Deference.** The UI helps people understand and interact with the content, but never competes with it.
* **Clarity.** Text is legible at every size, icons are precise and lucid, adornments are subtle and appropriate, and a sharpened focus on functionality motivates the design.
* **Depth.** Visual layers and realistic motion impart vitality and heighten people’s delight and understanding.



Whether you’re redesigning an older app or creating a new one, consider approaching the job in this way:

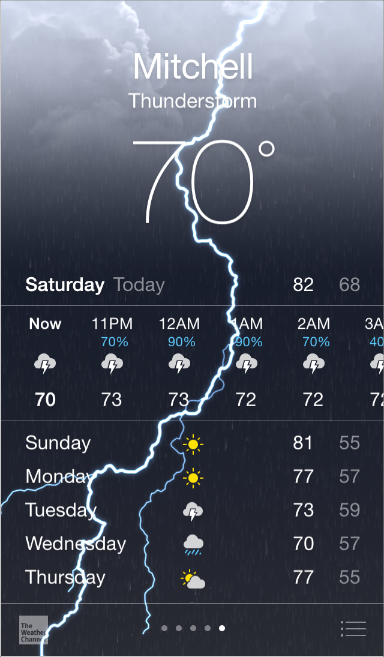
* First, look past the UI to the app’s core functionality and affirm its relevance.
* Next, use the themes of iOS to inform the design of the UI and the user experience. Add details and embellishments with care and never gratuitously.
* Finally, be sure to design your UI to adapt to various devices and modes so that users can enjoy your app in as many contexts as possible.

Throughout the process, be prepared to defy precedent, question assumptions, and let a focus on content and functionality motivate every design decision.

Defer to Content

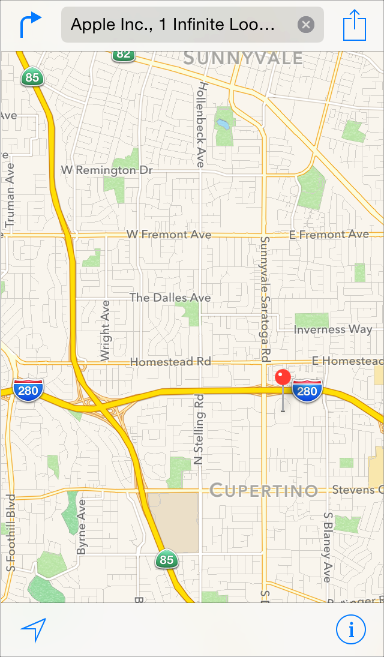
Although crisp, beautiful UI and fluid motion are highlights of the iOS experience, the user’s content is at its heart.

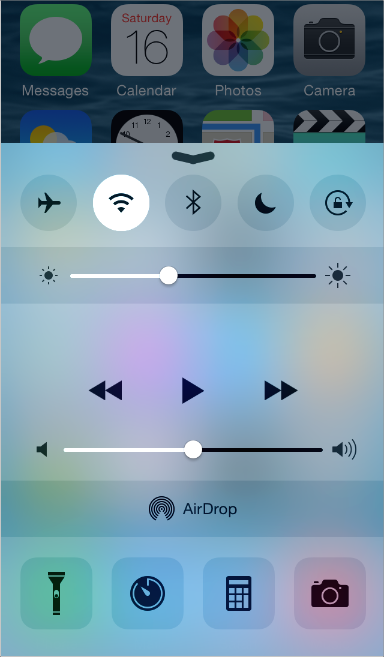
Here are some ways to make sure that your designs elevate functionality and defer to the user’s content.



**Take advantage of the whole screen.** Weather is a great example of this approach: The beautiful, full-screen depiction of a location’s current weather instantly conveys the most important information, with room to spare for hourly data.

**Reconsider visual indicators of physicality and realism.** Bezels, gradients, and drop shadows sometimes lead to heavier UI elements that can overpower or compete with the content. Instead, focus on the content and let the UI play a supporting role.





**Let translucent UI elements hint at the content behind them.** Translucent elements—such as Control Center—provide context, help users see that more content is available, and can signal transience. In iOS, a translucent element blurs only the content directly behind it—giving the impression of looking through rice paper—it doesn’t blur the rest of the screen.

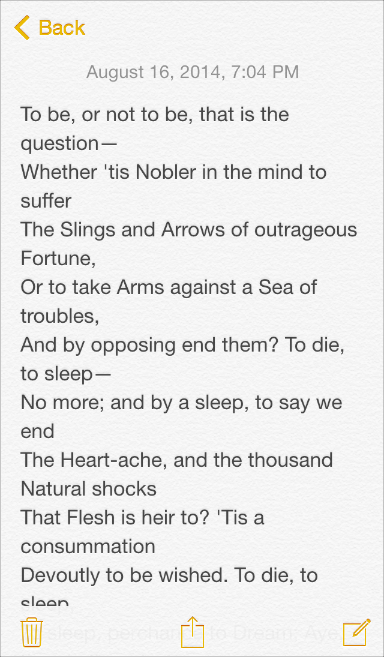
Provide Clarity

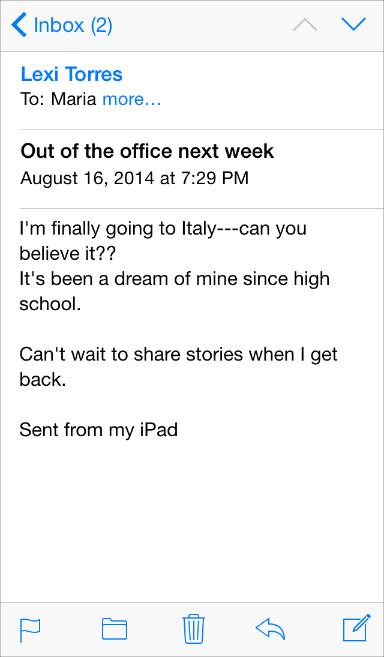
Providing clarity is another way to ensure that content is paramount in your app. Here are some ways to make the most important content and functionality clear and easy to interact with.



**Use plenty of negative space.** Negative space makes important content and functionality more noticeable and easier to understand. Negative space can also impart a sense of calm and tranquility, and it can make an app look more focused and efficient.

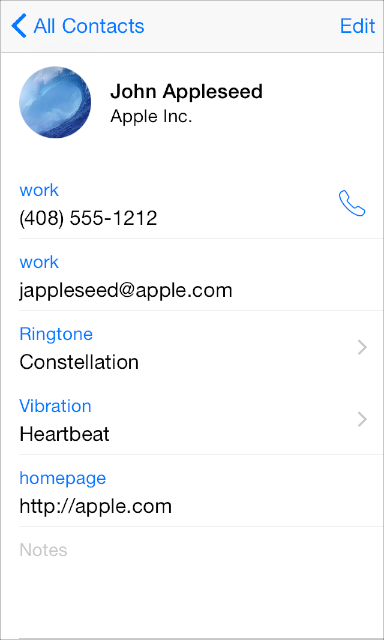
**Let color simplify the UI.** A key color—such as yellow in Notes—highlights important state and subtly indicates interactivity. It also gives an app a consistent visual theme. The built-in apps use a family of pure, clean system colors that look good at every tint and on both dark and light backgrounds.





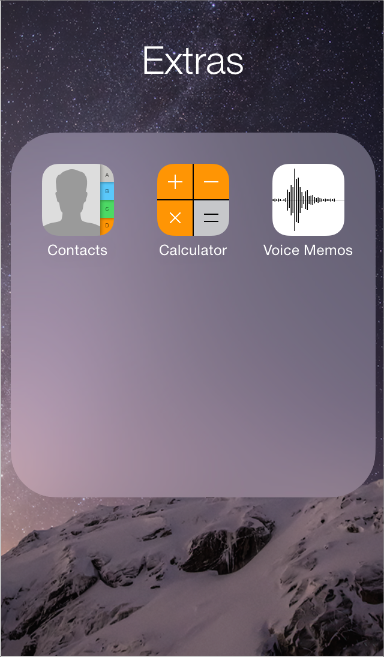
**Ensure legibility by using the system fonts.** iOS system fonts automatically adjust letter spacing and line height so that text is easy to read and looks great at every size. Whether you use system or custom fonts, be sure to adopt Dynamic Type so your app can respond when the user chooses a different text size.

**Embrace borderless buttons.** By default, all bar buttons are borderless. In content areas, a borderless button uses context, color, and a call-to-action title to indicate interactivity. And when it makes sense, a content-area button can display a thin border or tinted background that makes it distinctive.



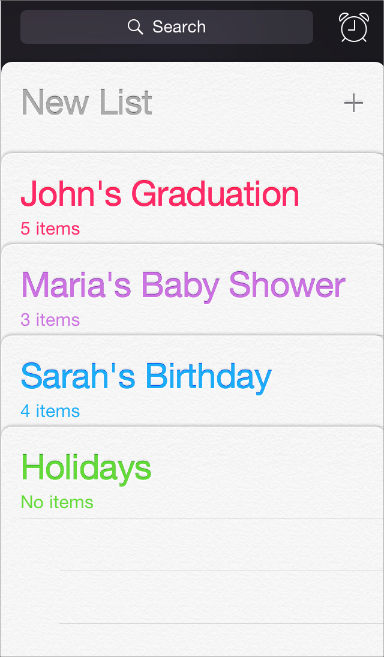
Use Depth to Communicate

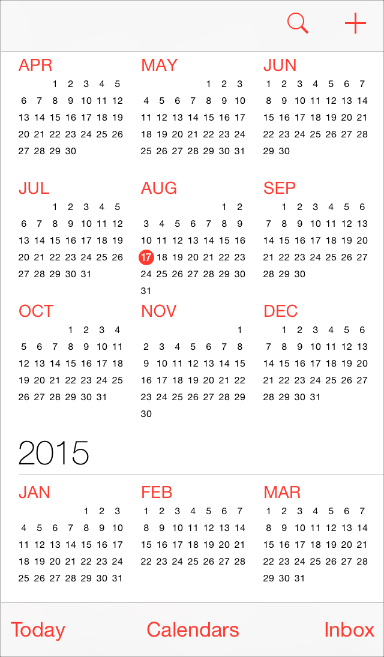
iOS often displays content in distinct layers that convey hierarchy and position, and that help users understand the relationships among onscreen objects.



By using a translucent background and appearing to float above the Home screen, folders separate their content from the rest of the screen.

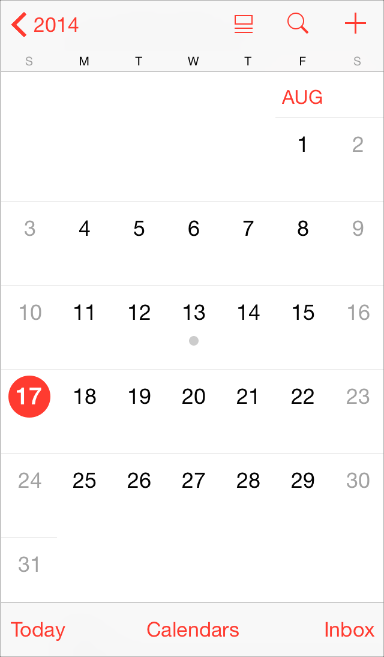
Reminders displays lists in layers, as shown here. When users work with one list, the other lists are collected together at the bottom of the screen.

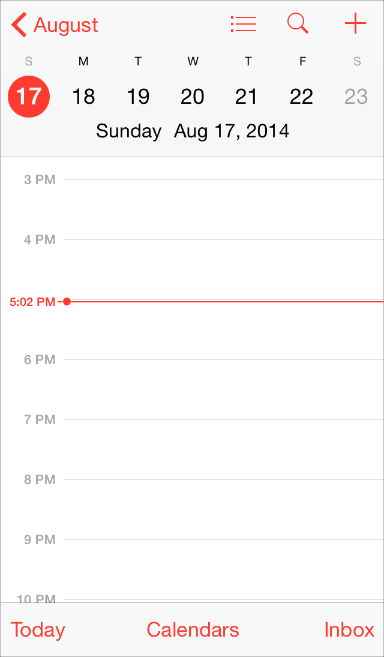




Calendar uses enhanced transitions to give users a sense of hierarchy and depth as they move between viewing years, months, and days. In the scrolling year view shown here, users can instantly see today’s date and perform other calendar tasks.

When users select a month, the year view zooms in and reveals the month view. Today’s date remains highlighted and the year appears in the back button, so users know exactly where they are, where they came from, and how to get back.

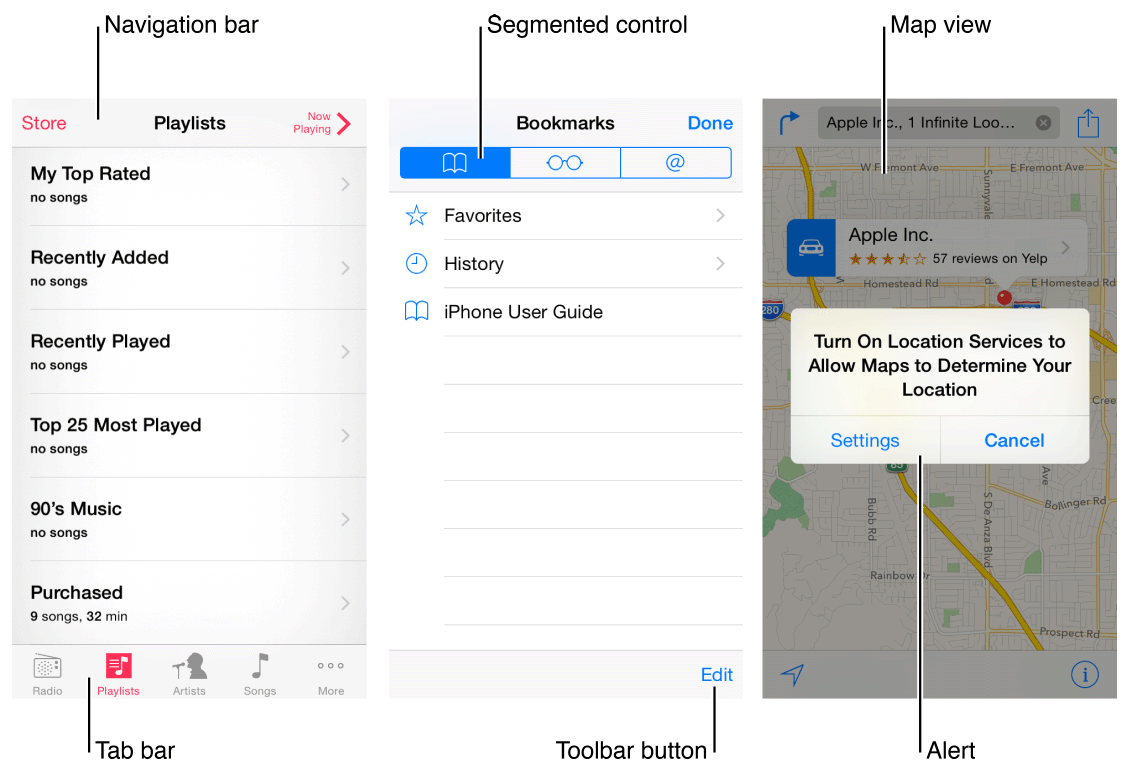




A similar transition happens when users select a day: The month view appears to split apart, pushing the current week to the top of the screen and revealing the hourly view of the selected day. With each transition, Calendar reinforces the hierarchical relationship between years, months, and days.

iOS App Anatomy

Almost all iOS apps use at least some of the UI components defined by the UIKit framework. Knowing the names, roles, and capabilities of these basic components helps you make informed decisions as you design the UI of your app.



The UI elements provided by UIKit fall into four broad categories:

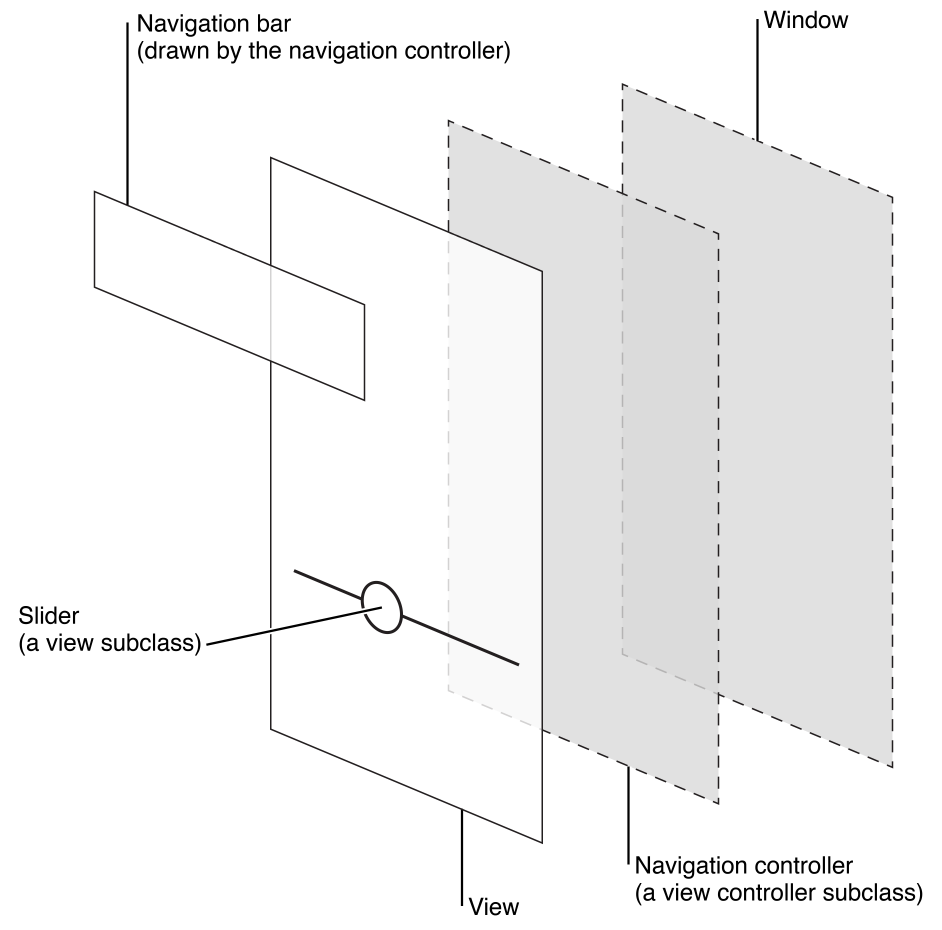
* **Bars.** Bars contain contextual information that tells users where they are and controls that help users navigate or initiate actions.
* **Content views.** Content views contain app-specific content and can enable behaviors such as scrolling, insertion, deletion, and rearrangement of items.
* **Controls.** Controls perform actions or display information.
* **Temporary views.** Temporary views appear briefly to give users important information or additional choices and functionality.

In addition to defining UI elements, UIKit defines objects that implement functionality, such as gesture recognition, drawing, accessibility, and printing support.

Programmatically, a UI element is a type of *view* because it inherits from UIView. A view knows how to draw itself onscreen, and it knows when a user touches within its bounds. Controls (such as buttons and sliders), content views (such as collection views and table views), and temporary views (such as alerts and action sheets) are all types of views.

To manage a set or hierarchy of views in your app, you typically use a *view controller*. A view controller coordinates the display of views, implements the functionality behind user interactions, and can manage transitions from one screen to another. For example, Settings uses a navigation controller to display its hierarchy of views.

Here’s an example of how views and view controllers can combine to present the UI of an iOS app.



Although developers think in terms of views and view controllers, users tend to experience an iOS app as a collection of screens. From this perspective, a *screen* generally corresponds to a distinct visual state or mode in an app.

NOTE

An iOS app includes a window. But—unlike a window in a computer app—an iOS window has no visible parts and it can’t be moved to another location on the display. Most iOS apps contain only one window; apps that support an external display can have more than one.

In *iOS Human Interface Guidelines*, the word *screen* is used as it’s understood by most users. As a developer, you might also read about screens in other contexts, where the term refers to the [UIScreen](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIScreen_Class/index.html#//apple_ref/occ/cl/UIScreen) object you can use to access an external display screen.

Adaptivity and Layout

Build In Adaptivity

People generally want to use their favorite apps on all their devices and in any orientation. In iOS 8 and later, you use size classes and Auto Layout to help you meet this expectation by defining how the layout of screens, view controllers, and views should adapt when the display environment changes. (The concept of *display environment* can refer to the entire device screen or only a portion of it, such as the area in a popover or the primary view in a split view controller.)

iOS defines two size classes: regular and compact. The *regular* size class is associated with expansive space and the *compact* size class is associated with constrained space. To characterize a display environment, you specify a horizontal size class and a vertical size class. As you might guess, an iOS device can use one set of size classes for portrait orientation and a different set of size classes for landscape.

iOS automatically makes various layout changes when the size classes of a display environment change. For example, when the vertical size class changes from compact to regular, navigation bars and toolbars automatically become taller.

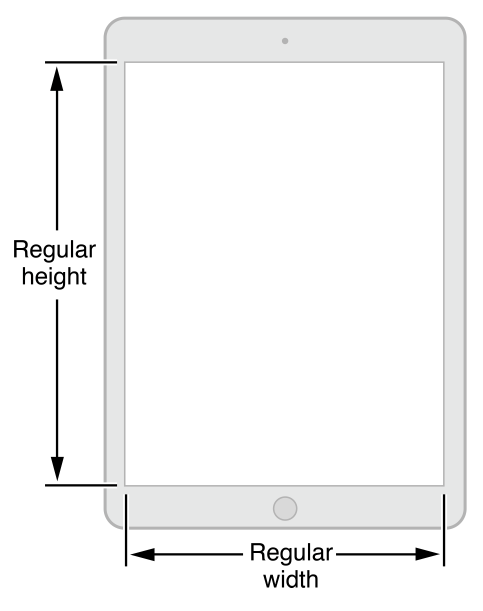
When you rely on size classes to drive changes in the layout, your app can look great in any display environment. To learn how to work with size classes in Interface Builder, see [*Size Classes Design Help*](https://developer.apple.com/library/ios/recipes/xcode_help-IB_adaptive_sizes/_index.html#//apple_ref/doc/uid/TP40014436).

NOTE

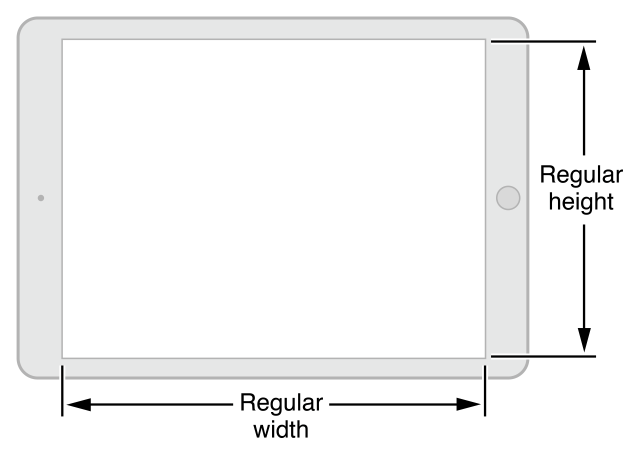
Within a size class, continue to use Auto Layout to make small layout adjustments, such as stretching or condensing content.

The following concrete examples can help you visualize how size classes characterize the display environments of various devices. For example, iPad uses the regular size class in both dimensions and in both orientations. In other words, the iPad display environment is always horizontally and vertically regular.

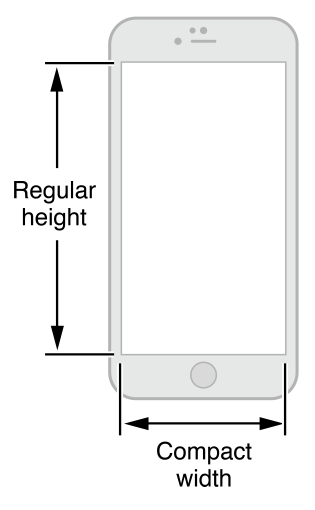
The size classes of iPad in portrait



The size classes of iPad in landscape

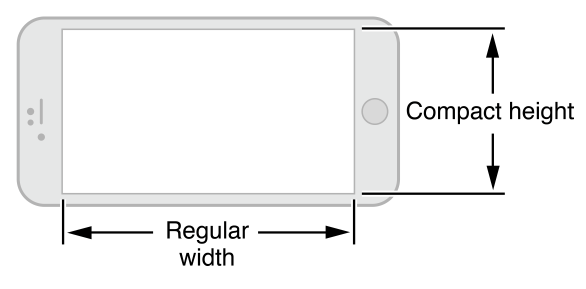


The display environment of iPhone can vary depending on the device and its orientation.

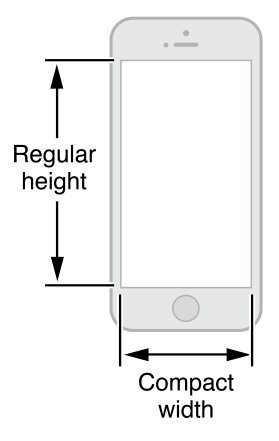


In portrait, iPhone 6 Plus uses the compact horizontal and regular vertical size classes.

In landscape, iPhone 6 Plus uses the regular horizontal and compact vertical size classes.

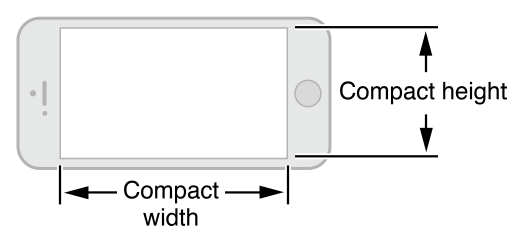


All other iPhone models, including iPhone 6, use the same sets of size classes.



In portrait, iPhone 6, iPhone 5, and iPhone 4s use the compact horizontal and regular vertical size classes.

In landscape, these devices use the compact size class in both the horizontal and vertical dimensions.



Provide a Great Experience in Each Environment

When you take advantage of adaptivity, you can ensure that your UI responds appropriately to changes in the display environment. Follow these guidelines to give people a great experience on all devices and orientations.

**Maintain focus on the primary content in all environments.** This is your highest priority. People use your app to view and interact with the content they care about. Changing the focus when the environment changes can disorient people and make them feel they’ve lost control over the app.

**Avoid gratuitous changes in layout.** A comparable experience in all environments lets people maintain their usage patterns when they rotate a device or run your app on a different device. For example, if you use a grid to display images in a horizontally regular environment, you don’t have to display the same information in a list in a horizontally compact environment, although you might adjust the dimensions of the grid.

**Be straightforward if your app runs in only one orientation.** People expect to use your app in different orientations, and it’s best when you can fulfill that expectation. But if it’s essential that your app run in only one orientation, you should:

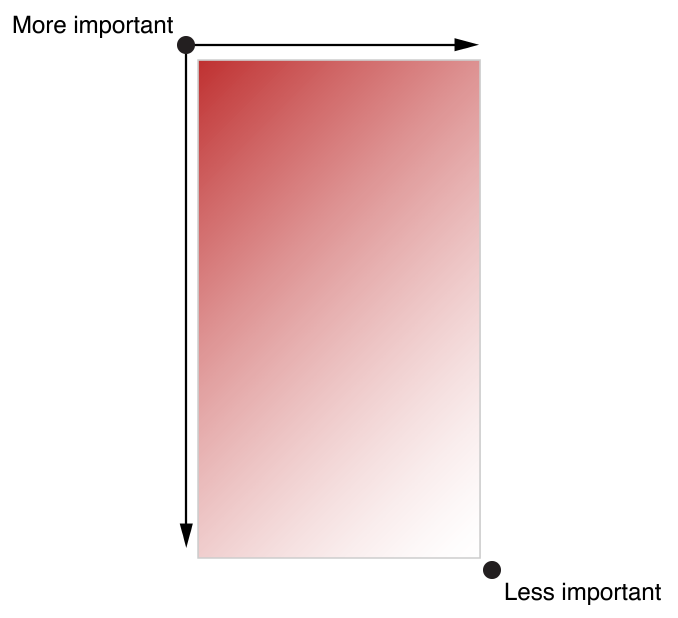
* **Avoid displaying a UI element that tells people to rotate the device.** Running in the supported orientation clearly tells people to rotate the device, if required, without adding unnecessary clutter to the UI.
* **Support both variants of an orientation.** For example, if an app runs only in landscape, people should be able to use it whether they’re holding the device with the Home button on the right or on the left. And if people rotate the device 180 degrees while using the app, it’s best if the app responds by rotating its content 180 degrees.

**If your app interprets changes in device orientation as user input, handle rotation in app-specific ways.** For example, a game that lets people move game pieces by rotating the device can’t respond to device rotation by rotating the screen. In a case like this, you should launch in both variants of the required orientation and allow people to switch between the variants until they start the main task of the app. As soon as people begin the main task, begin responding to device movement in app-specific ways.

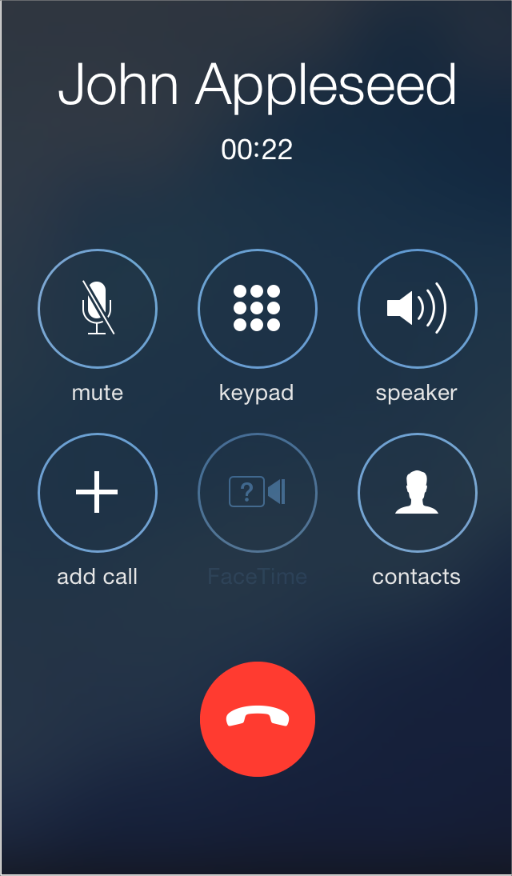
Use Layout to Communicate

Layout encompasses more than just how UI elements look on an app screen. With your layout, you show users what’s most important, what their choices are, and how things are related.

**Make it easy to focus on the main task by elevating important content or functionality.**Some good ways to do this are to place principal items in the upper half of the screen and—in left-to-right cultures—near the left side of the screen:



**Use visual weight and balance to show users the relative importance of onscreen elements.** Large items catch the eye and tend to appear more important than smaller ones. Larger items are also easier for users to tap, which makes them especially useful in apps—such as Phone and Clock—that users often use in distracting surroundings.



**Use alignment to ease scanning and communicate groupings or hierarchy.** Alignment tends to make an app look neat and organized and it gives users places to focus while they scroll through screenfuls of information. Indentation and alignment of different information groups indicate how the groups are related and make it easier for users to find specific items.

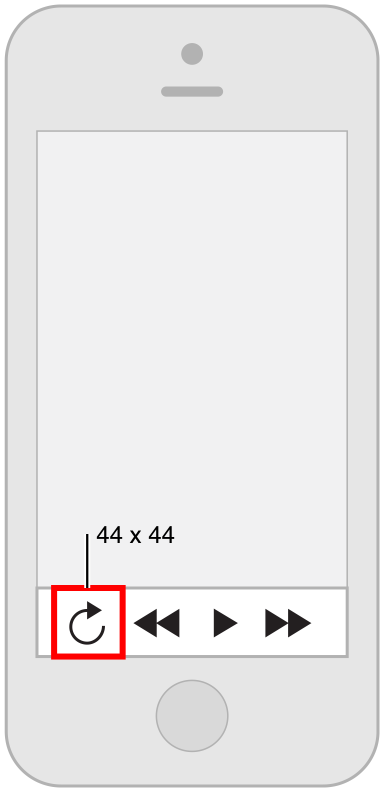
**Make sure that users can understand primary content at its default size.** For example, users shouldn’t have to scroll horizontally to read important text, or zoom to see primary images.

**Be prepared for changes in text size.** Users expect most apps to respond appropriately when they choose a different text size in Settings. To accommodate some text-size changes, you might need to adjust the layout; for more information about displaying text in your app, 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).

**As much as possible, avoid inconsistent appearances in your UI.** In general, elements that have similar functions should also look similar. People often assume that there must be a reason for the inconsistencies they notice, and they’re apt to spend time trying to figure it out.

**Make it easy for people to interact with content and controls by giving each interactive element ample spacing.** Give tappable controls a hit target of about 44 x 44 points.

Recommended



Not Recommended



Starting and Stopping

Start Instantly

It’s often said that people spend no more than a minute or two evaluating a new app. When you make the most of this brief period by presenting useful content immediately, you pique the interest of new users and give all users a superior experience.

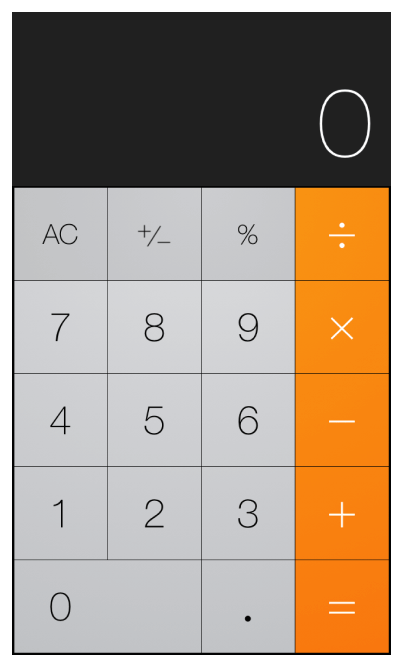
IMPORTANT

Don’t tell people to reboot or restart their devices after installing your app. Restarting takes time and can make your app seem unreliable and hard to use.

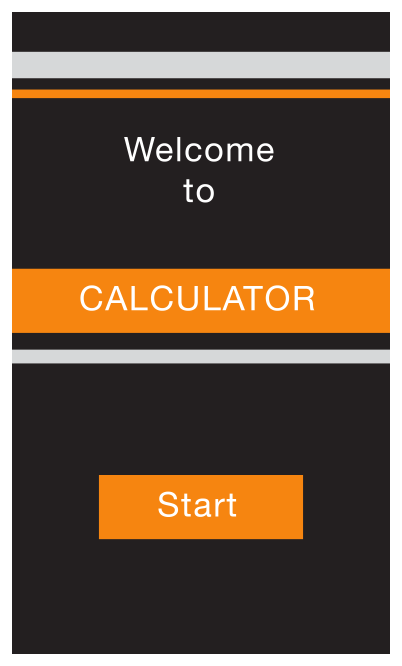
If your app has memory-usage or other issues that make it difficult to run unless the system has just booted, you need to address those issues. For some guidance on developing a well-tuned app, see [Use Memory Efficiently](https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/PerformanceTips/PerformanceTips.html#//apple_ref/doc/uid/TP40007072-CH7-SW8).

**As much as possible, avoid displaying a splash screen or other startup experience.** It’s best when users can begin using your app immediately.

Recommended



Not Recommended



**Avoid asking people to supply setup information.** Instead:

* **Focus on the needs of 80 percent of your users.** When you do this, most people won’t have to supply any settings, because the app is already set up to behave the way they expect. If there is functionality that only a few users might want—or that most users might want only once—leave it out.
* **Get as much information as possible from other sources.** If you can use any of the information people supply in built-in app or device settings, query the system for these values; don’t ask people to enter them again.
* **If you must ask for setup information, prompt people to enter it within your app.**Then, store this information as soon as possible (potentially, in your app’s settings). This way, people aren’t forced to switch to Settings before they get the chance to enjoy your app. If people need to make changes to this information later, they can go to the app’s settings at any time.

**Delay a login requirement for as long as possible.** It’s best when users can navigate through much of your app and use some of its functionality without logging in. For example, App Store doesn’t ask users to log in until they decide to buy something. Users often abandon apps that force them to log in before they can do anything useful.

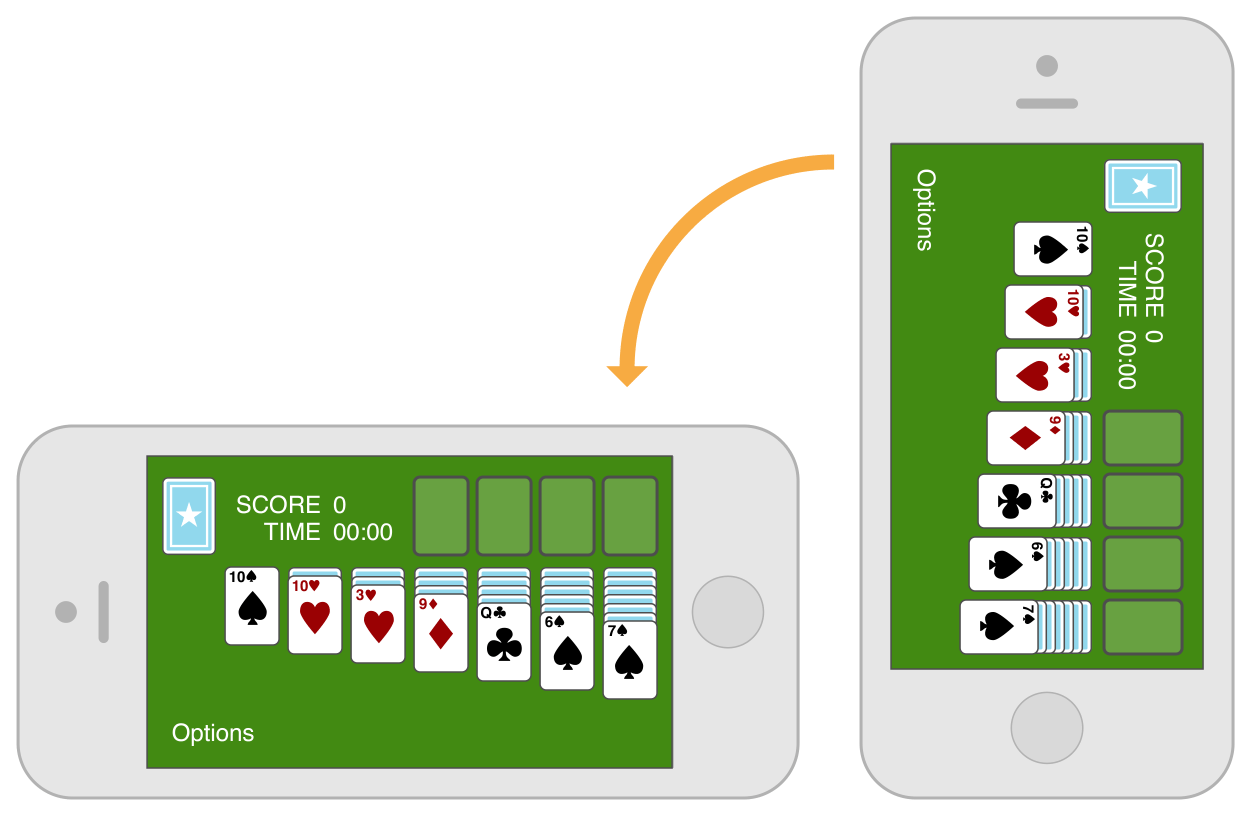
If users must log in, display in the login view a brief, friendly explanation that describes the reasons for the requirement and how it benefits users.

**Think carefully before providing an onboarding experience.** (Onboarding introduces an app’s features and explains how to perform common tasks.) Before you consider onboarding, make every effort to design your app so that all its features and tasks are intuitive and easily discoverable. *Onboarding is not a substitute for good app design*. If you still feel that onboarding is necessary, follow these guidelines to create a brief, targeted experience that doesn’t get in the user’s way.

* **Give users only the information they need to get started.** A good onboarding experience shows users what to do first or briefly demonstrates a few of the features that most users are interested in. If you give too much information to users before they have a chance to explore your app, you make users responsible for remembering details they don't need right away and you may send the message that your app is hard to use. If additional help is needed for specific tasks, provide that help only when the user is performing those tasks.
* **Use animation and interactivity to engage users and help them learn by doing.** Add text sparingly and only if it enriches the experience; don’t expect users to read long passages. For example, don’t describe how to perform a simple task when you can use animation to show users what to do. To lead users through a more complex task, you might add transient overlay views that briefly describe each step as the user is about to do it. As much as possible, avoid displaying screenshots of your app because they’re not interactive and users can confuse them with app UI.
* **Make it easy to dismiss or skip the onboarding experience.** After users have viewed the onboarding experience, they probably don’t want to view it again; other users may not want to view it at all. Be sure to remember the choice users make and don’t force them to make it every time they open your app.

**Avoid asking users to rate your app too soon.** Asking for a rating too soon tends to annoy users and may decrease the amount of useful feedback you receive. To encourage well-considered feedback, be sure to give users a chance to form an opinion about your app before you ask them to rate it. For example, you might wait until users have visited a minimum number of different screens or performed a minimum number of tasks.

**In general, launch in the device’s current orientation.** However, if your app runs in *only* one orientation, you should always launch in that orientation and let users rotate the device if necessary. For example, if a game or media-viewing app runs in landscape only, it’s appropriate to launch the app in landscape, even when the device is currently in portrait. This way, if people start the app while the device is in portrait, they know to rotate the device to landscape to view the content.



NOTE

It’s best when a landscape-only app supports both variants of landscape orientation—that is, with the Home button on the right or on the left. If the device is already in landscape, a landscape-only app should launch in that variant, unless there’s a very good reason not to. Otherwise, launch a landscape-only app in the variant with the Home button on the right. (To learn more about supporting different device orientations, see [Adaptivity and Layout](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/LayoutandAppearance.html#//apple_ref/doc/uid/TP40006556-CH54-SW1).)

**Provide a launch file or image.** iOS displays a launch image the moment your app starts—giving users the impression that your app is fast and giving your app enough time to load content. Learn how to create a launch file or image in [Launch Images](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/LaunchImages.html#//apple_ref/doc/uid/TP40006556-CH22-SW1).

**If possible, avoid requiring users to read a disclaimer or agree to an end-user license agreement before they can do anything else.** Instead, you can let the App Store display your disclaimer or end-user license agreement (EULA) so that people can access it before they get your app. If you must provide these items within your app, be sure to integrate them in a way that harmonizes with your UI and balances business requirements with user experience needs.

**When your app restarts, restore its state so users can continue where they left off.** People shouldn’t have to remember the steps they took to reach their previous location in your app. To learn more about efficient ways to preserve and restore your app’s state, see [Preserving Your App’s Visual Appearance Across Launches](https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/StrategiesforImplementingYourApp/StrategiesforImplementingYourApp.html#//apple_ref/doc/uid/TP40007072-CH5-SW2).

Always Be Prepared to Stop

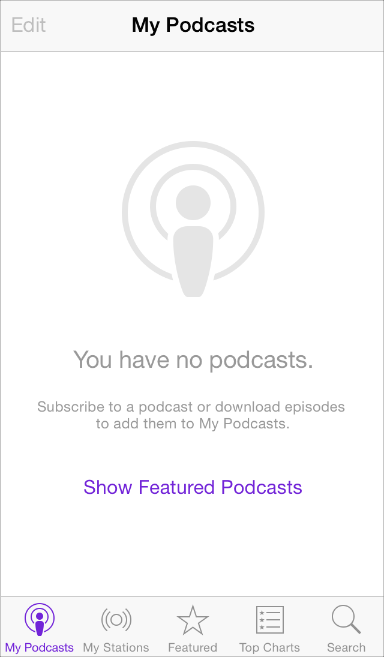
**An iOS app never displays a Close or Quit option.** People stop using an app when they switch to another app, return to the Home screen, or put their devices in sleep mode.

When people switch away from your app, iOS multitasking transitions it to the background and replaces its UI with the UI of the new app. To prepare for this situation, your app should:

* **Save user data as soon as possible and as often as reasonable.** Do this because an app in the background can be told to exit or terminate at any time.
* **Save the current state when stopping at the finest level of detail possible.** In this way, people don’t lose their context when they switch back to your app. For example, if your app displays scrolling data, save the current scroll position. You can learn more about efficient ways to preserve and restore your app’s state in [Preserving Your App’s Visual Appearance Across Launches](https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/StrategiesforImplementingYourApp/StrategiesforImplementingYourApp.html#//apple_ref/doc/uid/TP40007072-CH5-SW2).

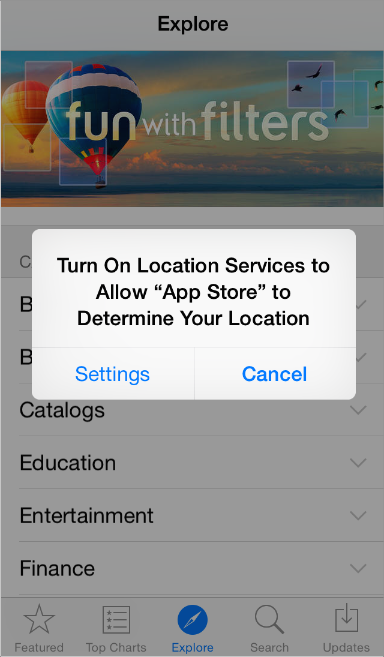
Some apps may need to keep running in the background while users run another app in the foreground. For example, users might want to keep listening to the song that’s playing in one app while they’re using a different app to check their to-do list or play a game. Learn how to handle multitasking correctly and gracefully in [Multitasking](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Multitasking.html#//apple_ref/doc/uid/TP40006556-CH38-SW1).

**Never quit an iOS app programmatically.** People tend to interpret this as a crash. If something prevents your app from functioning as intended, you need to tell users about the situation and explain what they can do about it. Here are two good ways to do this:



**If all app features are unavailable, display a screen that describes the situation and suggests a correction.** The information gives feedback to users and reassures them that there’s nothing wrong with your app. It also puts users in control, letting them decide whether they want to take corrective action and continue using your app or switch to another app.

**If only some app features are unavailable, display either a screen or an alert when people try to use the feature.** Otherwise, people should be able to use the rest of the app. If you decide to use an alert, be sure to display it *only* when people try to access the feature that isn’t functioning.



## Navigation

People tend to be unaware of the navigation experience in an app unless it doesn’t meet their expectations. Your job is to implement navigation in a way that supports the structure and purpose of your app without calling attention to itself.

Broadly speaking, there are three main styles of navigation, each of which is well suited to a specific app structure:

* Hierarchical
* Flat
* Content- or experience-driven

In a hierarchical app, users navigate by making one choice per screen until they reach their destination. To navigate to another destination, users must retrace some of their steps—or start over from the beginning—and make different choices. Settings and Mail are good examples of apps that use a hierarchical structure.

In an app with a flat information structure, users can navigate directly from one primary category to another because all primary categories are accessible from the main screen. Music and App Store are good examples of apps that use a flat structure.

It’s no surprise that in an app that uses a content- or experience-driven information structure, navigation is also defined by the content or experience. For example, users navigate through a book by moving from one page to the next or by choosing a page in the table of contents; in a game, navigation is often an important part of the experience.

In some cases, it works well to combine more than one navigation style in an app. For example, the items in one category of a flat information structure might best be displayed in a hierarchy.

**Users should always know where they are in your app and how to get to their next destination.** Regardless of the navigation style that suits the structure of your app, the most important thing is that a user’s path through the content is logical, predictable, and easy to follow.

UIKit defines some standard UI elements that make it easy to implement hierarchical and flat navigation styles, in addition to some elements that help you enable content-centric navigation, such as in a book-style or media-viewing app. A game or other app that provides an experience-driven navigation style typically relies on custom elements and behaviors.

**Use a navigation bar to give users an easy way to traverse a hierarchy of data.** The navigation bar’s title can show users their current position in the hierarchy; the back button makes it easy to return to the previous level. To learn more, see [Navigation Bar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW3).

**Use a tab bar to display several peer categories of content or functionality.** A tab bar is a good way to support a flat information architecture and its persistence lets people switch between categories regardless of their current location. To learn more, see [Tab Bar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW52).

**Use a page control when each app screen represents an individual instance of the same type of item or page.** A page control is good for showing users how many items or pages are available and which one is currently displayed. For example, Weather uses a page control to show how many location-specific weather pages the user has opened. To learn more about the page control, see [Page Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW6).

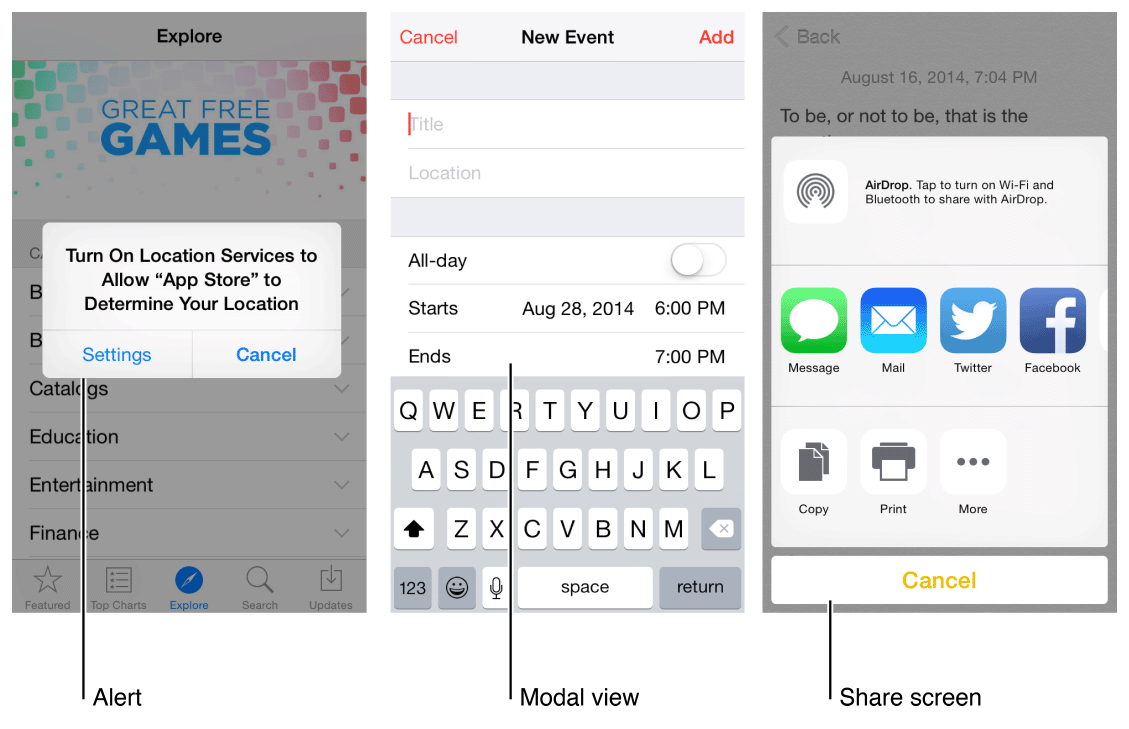
**In general, it’s best to give users one path to each screen.** If there’s one screen that users need to see in more than one context, consider using a temporary view, such as a modal view, action sheet, or alert. To learn more, see [Modal View](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW3), [Action Sheet](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW36), and [Alert](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW2).

UIKit also provides the following related controls:

* [Segmented Control](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW27). A segmented control can give users a way to see different categories or aspects of the content on the screen; it doesn’t enable navigation to a new screen.
* [Toolbar](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW4). Although a toolbar looks similar to a navigation bar or a tab bar, it doesn’t enable navigation. Instead, a toolbar gives users controls that act on the contents of the current screen.

Modal Contexts

Modality—that is, a mode in which something exists or is experienced—has advantages and disadvantages. It can give users a way to complete a task or get information without distractions, but it does so by temporarily preventing them from interacting with the rest of the app.



Ideally, people can interact with iOS apps in nonlinear ways, so it’s best when you can minimize the number of modal experiences in your app. In general, consider creating a modal context only when:

* It’s critical to get the user’s attention
* A self-contained task must be completed—or explicitly abandoned—to avoid leaving the user’s data in an ambiguous state

**Keep modal tasks simple, short, and narrowly focused.** You don’t want your users to experience a modal view as a mini app within your app. If a subtask is too complex, people can lose sight of the main task they suspended when they entered the modal context. Be especially wary of creating a modal task that involves a hierarchy of views, because people can get lost and forget how to retrace their steps. If a modal task must contain subtasks in separate views, be sure to give users a single, clear path through the hierarchy, and avoid circularities. For guidelines on using modal views, see [Modal View](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW3).

**Always provide an obvious and safe way to exit a modal task.** People should always be able to predict the fate of their work when they dismiss a modal view.

**If the task requires a hierarchy of modal views,** make sure your users understand what happens if they tap a Done button in a view that’s below the top level. Examine the task to decide whether a Done button in a lower-level view should complete only the part of the task in that view, or the entire task. Because of this potential for confusion, avoid adding a Done button to a subordinate view as much as possible.

**Reserve alerts for delivering essential—and ideally actionable—information.** An alert interrupts the user’s experience and requires a tap to dismiss, so it’s important for users to feel that the alert’s message warrants the intrusion. To learn more, see [Alert](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW2).

**Respect users’ preferences for receiving notifications.** In Settings, users indicate how they want to receive notifications from your app. Be sure to abide by these preferences so that users aren’t tempted to turn off all notifications from your app.

Interactivity and Feedback

Users Know the Standard Gestures

People use gestures—such as tap, drag, and pinch—to interact with apps and their iOS devices. Using gestures gives people a close personal connection to their devices and enhances their sense of direct manipulation of onscreen objects. People generally expect gestures to work the same in all the apps they use.

**Tap** To press or select a control or item.

**Drag** To scroll or pan—that is, move side to side.

To drag an element.

**Flick** To scroll or pan quickly.

**Swipe** With one finger, to return to the previous screen, to reveal the hidden view in a split view controller, or the Delete button in a table-view row.

With four fingers, to switch between apps on iPad.

**Double tap** To zoom in and center a block of content or an image.

To zoom out (if already zoomed in).

**Pinch** Pinch open to zoom in; pinch close to zoom out.

**Touch and hold** In editable or selectable text, to display a magnified view for cursor positioning.

**Shake** To initiate an undo or redo action.

In addition to the standard gestures users know, iOS defines a few gestures that invoke systemwide actions, such as revealing Control Center or Notification Center. Users rely on these gestures to work regardless of the app they’re using.

**Avoid associating different actions with the standard gestures.** Unless your app is a game, redefining the meaning of a standard gesture may confuse people and make your app harder to use.

**Avoid creating custom gestures that invoke the same actions as the standard gestures.**People are used to the behavior of the standard gestures, and they don’t appreciate being expected to learn different ways to do the same thing.

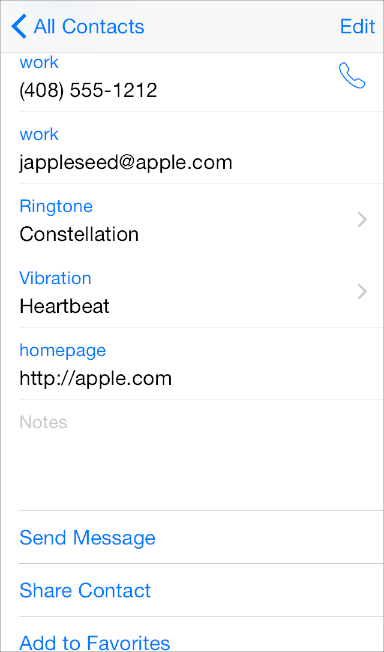
**Use complex gestures as shortcuts to expedite a task, not as the only way to perform it.**As much as possible, always give users a simple, straightforward way to perform an action, even if it means an extra tap or two. Simple gestures let users focus on the experience and the content, not the interaction.

**In general, avoid defining new gestures unless your app is a game.** In games and other immersive apps, custom gestures can be a fun part of the experience. But in apps that help people do things that are important to them, it’s best to use standard gestures because people don’t have to make an effort to discover them or remember them.

**In a regular environment, consider using multifinger gestures.** Although complex gestures aren’t appropriate for every app, they can enrich the experience in apps that people spend a lot of time in, such as games or content-creation environments. Always bear in mind that nonstandard gestures aren’t discoverable and should rarely, if ever, be the only way to perform an action.

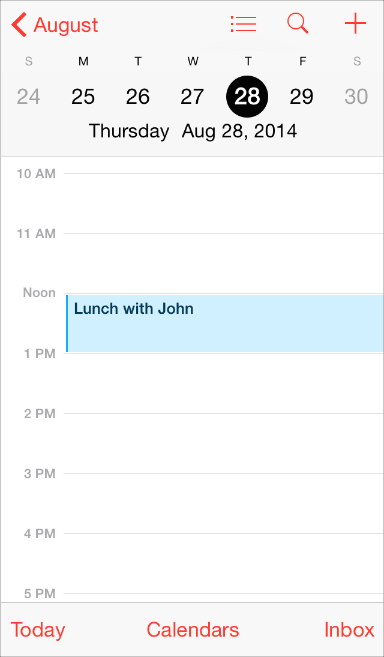
Interactive Elements Invite Touch

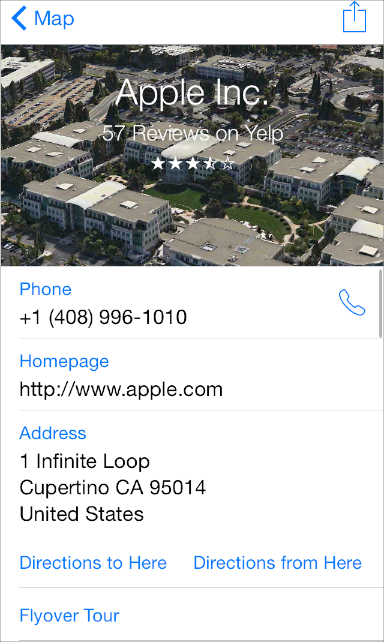
To signal interactivity, the built-in apps use a variety of cues, including color, location, context, and meaningful icons and labels. Users rarely need additional decorations to show them that an onscreen element is interactive or to suggest what it does.



A key color gives users a strong visual indicator of interactivity, especially in apps that don’t use an abundance of other colors. In Contacts, blue marks the interactive elements and gives the app a unified and recognizable visual theme.

The back button uses several cues to indicate its interactivity and convey its function: It appears in response to navigation, it displays a back-pointing chevron, it typically uses a key color, and it can display a title that describes the previous screen.





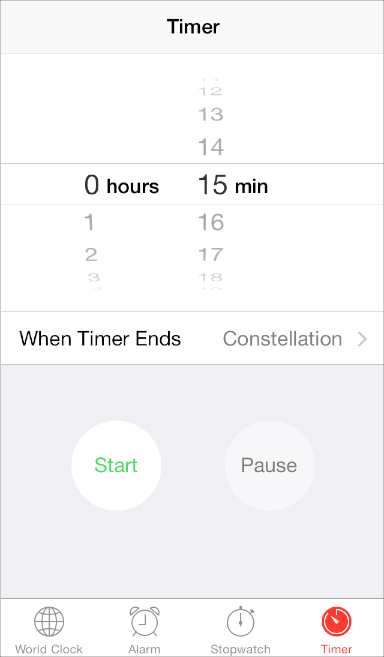
An icon or a title that provides a clear call to action invites users to tap it. For example, the titles in Maps, such as “Flyover Tour” and “Directions to Here,” clearly describe actions that users can take. Combined with a key color, actionable titles tend to make button borders or other embellishments superfluous.

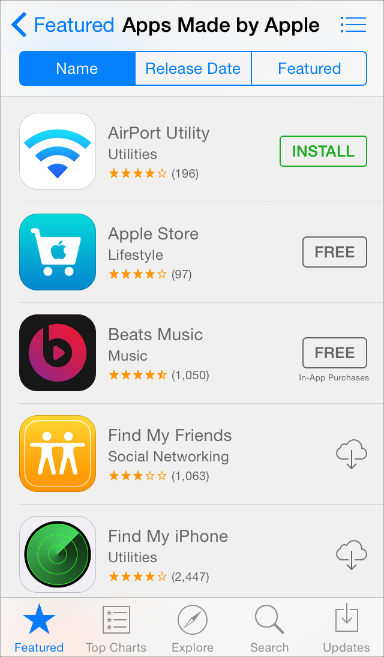
**In a content area, add a button border or background only if necessary.** Buttons in bars, action sheets, and alerts don’t need borders because users know that most of the items in these areas are interactive. In a content area, on the other hand, a button might need a border or a background to distinguish it from the rest of the content. For example, Music, Clock, Photos, and App Store use such buttons in a few specific contexts.



Photos uses a button border to differentiate the Start Sharing button from the explanatory text that appears above it.

Clock uses button backgrounds in the Stopwatch and Timer screens to draw attention to the Start and Pause buttons and to make them easy to tap even when the user's surroundings are distracting.





App Store uses a button border in a table row to emphasize the distinction between tapping the row to get more information and tapping the button to initiate (or install) a purchase.

Feedback Aids Understanding

Feedback helps users know what an app is doing, discover what they can do next, and understand the results of their actions. UIKit controls and views provide many kinds of feedback.

**As much as possible, integrate status and other relevant feedback information into your UI.** It’s best when users can get this type of information without taking action or being distracted from their content. For example, Mail displays the current mailbox status in the toolbar where it doesn’t compete with the user’s content.

mage: ../Art/mail_update_2x.png

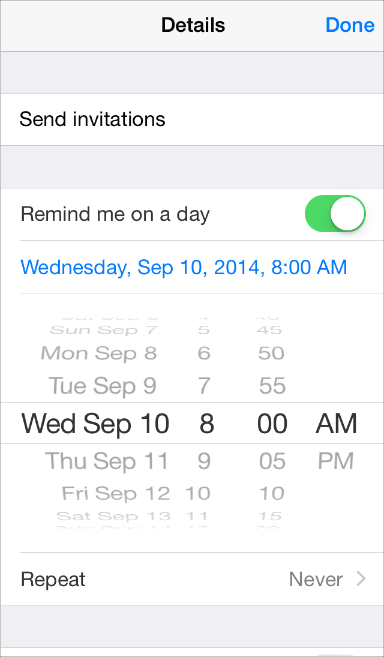
**Avoid unnecessary alerts.** An alert is a powerful feedback mechanism, but it should be used only to deliver important—and ideally actionable—information. If users see too many alerts that don’t contain essential information, they quickly learn to ignore all alerts. To learn more about using an alert, see [Alert](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW2).

Inputting Information Should Be Easy

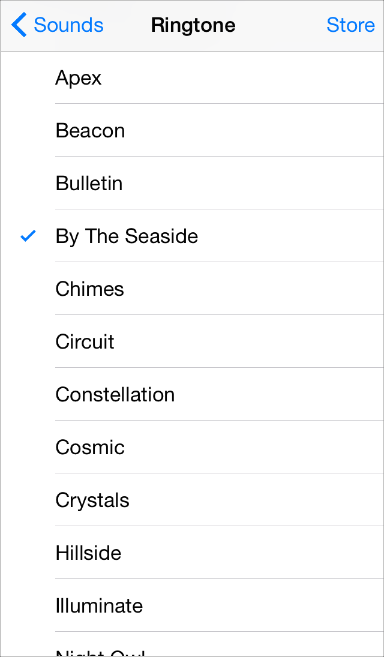
Inputting information takes time and attention, whether people tap controls or use the keyboard. When an app slows people down by asking for a lot of user input before anything useful happens, people can feel discouraged from using it.

**Make it easy for users to make choices.** For example, you can use a picker or a table view instead of a text field, because most people find it easier to select an item from a list than to type words.

A date picker in Reminders



A list of options in Settings



**Get information from iOS, when appropriate.** People store lots of information on their devices. When it makes sense, don’t force people to give you information that you can easily find for yourself, such as their contacts or calendar information.

**Balance a request for input by giving users something useful in return.** A sense of give and take helps people feel they’re making progress as they move through your app.

## Animation

Beautiful, subtle animation pervades the iOS UI and makes the app experience more engaging and dynamic. Appropriate animation can:

* Communicate status and provide feedback
* Enhance the sense of direct manipulation
* Help people visualize the results of their actions

**Add animation cautiously, especially in apps that don’t provide an immersive experience.**Animation that seems excessive or gratuitous can obstruct app flow, decrease performance, and distract users from their task.

In particular, use motion effects and UIKit dynamic behaviors with purpose and restraint, and be sure to test the results. Used appropriately, these effects can increase users’ understanding and enjoyment; overusing them can make an app seem disorienting and difficult to control.

**When appropriate, make custom animation consistent with built-in animation.** People are accustomed to the subtle animation used in the built-in iOS apps. In fact, people tend to regard the smooth transitions between views, the fluid response to changes in device orientation, and the physics-based scrolling as an expected part of the iOS experience. Unless you’re creating an app that enables an immersive experience—such as a game—custom animation should be comparable to the built-in animations.

**Use animation consistently throughout your app.** As with other types of customization, it’s important to use custom animation consistently so that users can build on the experience they gain as they use your app.

**In general, strive for realism and credibility in custom animation.** People tend to be willing to accept artistic license in appearance, but they can feel disoriented when they experience movement that doesn’t make sense or appears to defy physical laws. For example, if you reveal a view by sliding it down from the top of the screen, you should dismiss it by sliding it back up because doing so helps users remember where the view comes from. If you dismiss the same view by sliding it down beyond the bottom of the screen, you break the user’s mental model of a view that’s available above the top of the screen.

Branding

Successful branding involves more than adding brand assets to an app. The best apps integrate existing assets with a unique look and feel to give users a delightful, memorable experience.

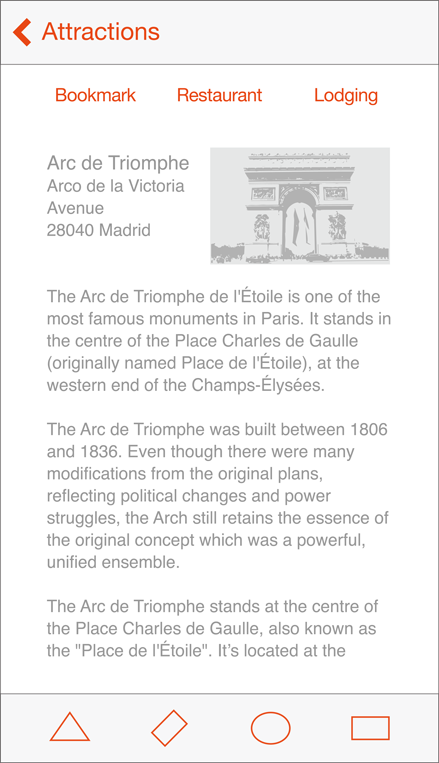
iOS makes it easy to use custom icons, colors, and fonts to create a distinctive UI that sets your app apart from the rest. As you design these elements, keep two things in mind:

* Each custom element should look good and function well by itself, but it should also look like it belongs with the other elements in the app, whether they’re custom or standard.
* To feel at home in iOS, an app doesn’t have to look like the built-in apps but it does need to integrate deference, clarity, and depth (to learn more about these themes, see[Designing for iOS](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html#//apple_ref/doc/uid/TP40006556-CH66-SW1)). Take the time to figure out what deference, clarity, and depth mean in your app and then express this meaning in your custom elements.

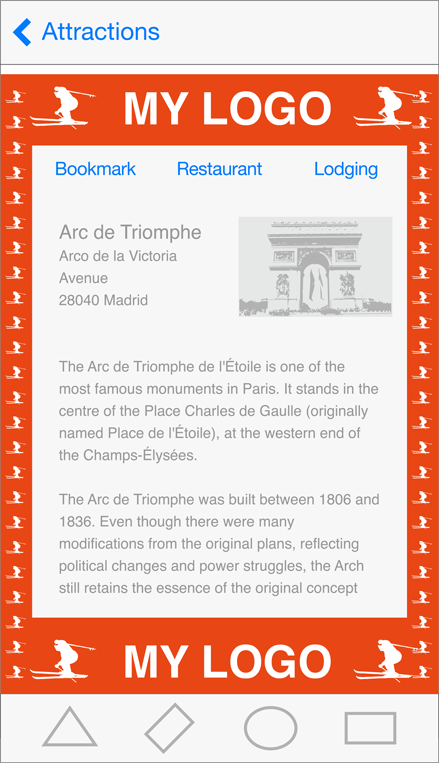
When you need to remind users of an existing brand within your app, follow these guidelines.

**Incorporate a brand’s assets in a refined, unobtrusive way.** People use your app to get things done or be entertained; they don’t want to feel as if they’re being forced to watch an advertisement. For the best user experience, you want to quietly remind users of the brand identity through your choice of font, color, and imagery.

Recommended



Not Recommended



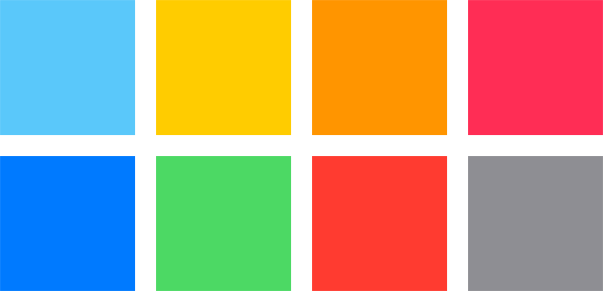
**Don’t take space away from the content people care about.** For example, displaying a second, persistent bar at the top of the screen that does nothing but display brand assets means that there’s less room for content. Instead, defer to the user’s content and consider less intrusive ways to display pervasive branding, such as using a custom tint or font, or subtly customizing the background of a screen.

**Resist the temptation to display your logo throughout the app.** Mobile device screens are relatively small, and every occurrence of a logo takes space away from the content that users want to see. What’s more, displaying a logo in an app doesn’t serve the same purpose as displaying it in a webpage: It’s common for users to arrive on a webpage without knowing its owner, but it’s much less likely that users will open an iOS app without seeing its app icon.

Color and Typography

Color Enhances Communication

In iOS, color helps indicate interactivity, impart vitality, and provide visual continuity. The built-in apps use a family of pure, clean colors that look great individually and in combination, and on both light and dark backgrounds.



**If you create multiple custom colors, make sure they work well together.** For example, if pastels are essential to your app’s style, you should create a family of coordinating pastels that can be used throughout the app.

**Pay attention to color contrast in different contexts.** For example, if there’s not enough contrast between the navigation bar background and the bar-button titles, the buttons will be hard for users to see. A quick but unscientific way to find out if your colors have sufficient contrast is to view your app on a device in different lighting conditions, including outdoors on a sunny day.

Although viewing your app on a device can help you find some of the areas you need to work on, it's no substitute for a more objective approach that yields reliable results. This approach involves determining the ratio between the luminance values of the foreground and background colors. To get this ratio, use an online contrast ratio calculator or you can perform the calculation yourself using the formula established in the WCAG 2.0 standard. Ideally the color contrast ratio in your app is 4.5:1 or higher.

**Take bar translucency and app content into account when you use a custom bar tint.** If you need to create a bar tint that matches a specific color, such as a color in an existing brand, you may have to experiment with various colors before you get the results you want. A bar’s appearance is affected by both the translucency that iOS provides and the appearance of the app content that can be behind the bar.

API NOTE

To tint bar button items, use the tintColor property; to tint the bar itself, use thebarTintColor property. To learn more about these bar properties, see[*UINavigationBar Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UINavigationBar_Class/index.html#//apple_ref/doc/uid/TP40006887), [*UITabBar Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITabBar_Class/index.html#//apple_ref/doc/uid/TP40007521), [*UIToolbar Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIToolbar_Class/index.html#//apple_ref/doc/uid/TP40006927), and [*UISearchBar Class Reference*](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISearchBar_Class/index.html#//apple_ref/doc/uid/TP40007529).

**Be aware of color blindness.** Most color blind people have difficulty distinguishing red from green. Test your app to make sure that there are no places where you use red and green as the only way to distinguish between two states or values (some image-editing software includes tools that can help you proof for color-blindness). In general, it’s a good idea to use more than one way to indicate an element’s interactivity (to learn more about indicating interactivity in iOS, see [Interactive Elements Invite Touch](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/InteractivityInput.html#//apple_ref/doc/uid/TP40006556-CH55-SW4)).

**Consider choosing a key color to indicate interactivity and state.** Key colors in the built-in apps include yellow in Notes and red in Calendar. If you define a key color to indicate interactivity and state, make sure that the other colors in your app don’t compete with it.

**Avoid using the same color in both interactive and noninteractive elements.** Color is one of the ways that a UI element indicates its interactivity. If interactive and noninteractive elements have the same color, it’s harder for users to know where to tap.

**Color communicates, but not always in the way you intend.** Everyone sees color differently, and many cultures differ in how they assign meanings to colors. Spend time to research how your use of color might be perceived in other countries and cultures. As much as possible, you want to be sure that the colors in your app send the appropriate message.

**In most cases, don’t let color distract users.** Unless color is essential to your app’s purpose, it usually works well to use color as a subtle enhancement.

Text Should Always Be Legible

Above all, text must be legible. If users can’t read the words in your app, it doesn’t matter how beautiful the typography is. When you adopt Dynamic Type in your app, you get:

* Automatic adjustments to letter spacing and line height for every font size
* The ability to specify different text styles for semantically distinct blocks of text, such asBody, Footnote, or Headline
* Text that responds appropriately to changes the user makes to text-size settings (including accessibility text sizes)

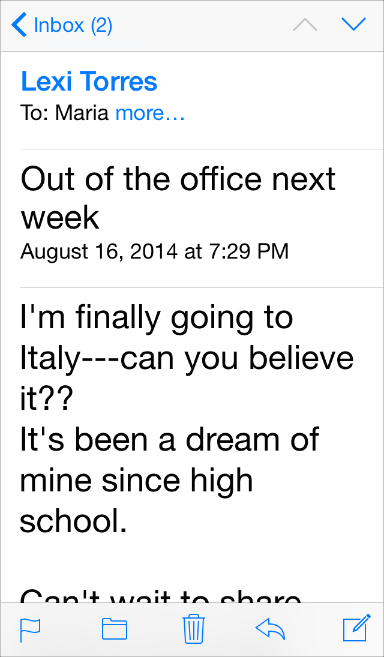
NOTE

If you use a custom font, you can still scale type according to the system setting for text size. Your app is responsible for responding appropriately when the user changes the setting.

Adopting Dynamic Type requires some work on your part. To learn how to use text styles and ensure that your app gets notified when the user changes the text size setting, see [Text Styles](https://developer.apple.com/library/ios/documentation/StringsTextFonts/Conceptual/TextAndWebiPhoneOS/CustomTextProcessing/CustomTextProcessing.html#//apple_ref/doc/uid/TP40009542-CH4-SW65).

**Prioritize content when responding to text-size changes.** Not all content is equally important to users. When users choose a larger text size, they want to make the content they care about easier to read; they don’t always want every word on the screen to be larger.

For example, when users choose a large accessibility text size, Mail displays the subject and body of the message in the large size but leaves the less important text—such as the date and the recipient—in a smaller size.



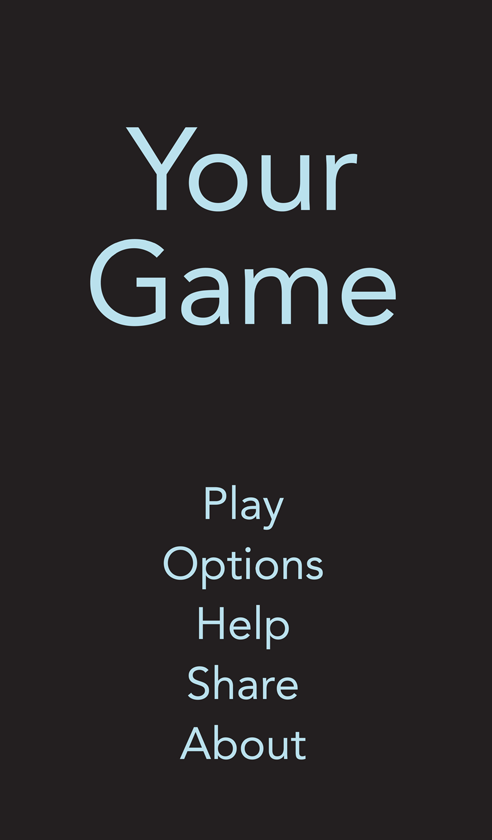
**When appropriate, adjust the layout when the user chooses a different text size.** For example, you might want to change a one-column layout of body text to a two-column layout when the user chooses a small text size. If you decide to adjust the layout for different text sizes, you might choose to do so for subsets of sizes—such as small, medium, and large—rather than change the layout for every possible size.

**Make sure all styles of a custom font are legible at different sizes.** One way to do this is to emulate some of the ways iOS displays font styles at different text sizes. For example:

* Text should never be smaller than 11 points, even when the user chooses the extra-small text size. For comparison, the body style uses a font size of 17 points at the large size, which is the default text-size setting.
* In general, font size and leading values differ by one point per text-size setting. Exceptions to this are the two caption styles, which use the same font size, leading, and tracking at the extra-small, small, and medium settings.
* At the smallest three text sizes, tracking values are relatively large; at the largest three text sizes, tracking values are relatively tight.
* The headline and body styles use the same font size. To distinguish it from the body style, the headline style uses a heavier weight.
* Text in a navigation controller uses the same font size that body style text uses for the large setting (specifically, 17 points).
* Text always uses either regular or medium weight; it doesn’t use light or bold.

**In general, use a single font throughout your app.** Mixing several different fonts can make your app seem fragmented and sloppy. Instead, use one font and just a few styles and sizes. Use the [UIFont](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIFont_Class/index.html#//apple_ref/occ/cl/UIFont) text styles API to define different areas of text according to semantic usage, such as body or headline.

Recommended



Not Recommended



Icons and Graphics

The App Icon

Every app needs a beautiful app icon. It’s not unusual for people to base their initial opinions about your app’s quality, purpose, and reliability solely on the look of your app icon.



Here are a few of the things you should keep in mind as you think about your app icon. When you’re ready to start creating it, see [App Icon](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/AppIcons.html#//apple_ref/doc/uid/TP40006556-CH19-SW1) for detailed guidance and specifications.

* The app icon is an important part of your app’s brand. Approach the design as an opportunity to tell your app’s story and build an emotional connection with users.
* The best app icons are unique, uncluttered, engaging, and memorable.
* An app icon needs to look good at many different sizes and on different backgrounds. Details that might enrich an icon at large sizes can make it look muddy at small sizes.

Bar Icons

iOS provides a lot of small icons—representing common tasks and types of content—for use in tab bars, toolbars, and navigation bars. It’s a good idea to use the built-in icons as much as possible because users already know what they mean.

mage: ../Art/bar-icons_2x.png

If you need to represent custom actions or content types you can create custom bar icons. Designing these small, streamlined icons is very different from designing an app icon. If you need to create 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) to learn how.



Note that you can use text instead of icons to represent items in a navigation bar or toolbar. For example, Calendar uses “Today,” “Calendars,” and “Inbox” instead of icons in the toolbar.

To help you decide whether to use text or icons in the navigation bar or toolbar in your app, consider how many icons are visible onscreen at one time. Too many icons on a screen can make an app seem difficult to decode. Also, note that this decision might be different depending on the horizontal environment, because a horizontally regular environment tends to have more room for text in bars.

Graphics

iOS apps tend to be graphically rich. Whether you’re displaying users’ photos or creating custom artwork, here are a few guidelines you should follow.

**Support the Retina display.** Make sure that you supply high resolution assets for all artwork and graphics in your app. In particular, supply @3x assets for iPhone 6 Plus and @2x assets for all other high-resolution iOS devices.

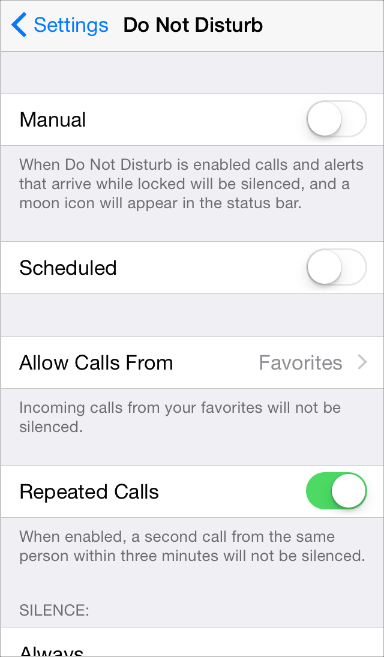
**Display photos and graphics in their original aspect ratio, and don’t scale them greater than 100%.** You don’t want the artwork or graphics in your app to look skewed or too large. Let users choose whether they want to zoom images in or out.

**Don’t use images that replicate Apple products in your designs.** These symbols are copyrighted, and product designs can change frequently.

**Don't use Apple app icons, images, or screenshots in your designs.** Apple designs are copyrighted and can’t appear in your UI unless they are provided by the system.

Terminology and Wording

Every word you display in an app is part of a conversation you have with users. Use this conversation as an opportunity to provide clarity and to help people feel comfortable in your app.



Settings is an essential app for all users, so it uses simple, direct language to describe what users can do. For example, Settings > Do Not Disturb explains the effects of various options without using technical jargon that might be difficult for unsophisticated users to understand.

**Use terminology that you’re sure your users understand.** Use what you know about your users to determine whether the words and phrases you plan to use are appropriate. For example, technical jargon is rarely helpful in an app aimed at unsophisticated users, but in an app designed for technically savvy users, it might be appreciated.

**Use a tone that’s informal and friendly, but not too familiar.** You want to avoid being stilted or too formal, but you don’t want to risk sounding falsely jovial or patronizing. Remember that users are likely to read the text in your UI many times, and what might seem clever at first can become irritating when repeated.

**Think like a newspaper editor, and watch out for redundant or unnecessary words.** When your UI text is short and direct, users can absorb it quickly and easily. Identify the most important information, express it concisely, and display it prominently so that people don’t have to read too many words to find what they’re looking for or to figure out what to do next.

**Give controls short labels or use well-understood icons.** People should be able to tell at a glance what a control does.

**Take care to be accurate when describing dates.** It’s often appropriate to use friendly terms such as *today* and *tomorrow* when you display date information in your UI. But it can be confusing if you don’t account for the user’s current locale. For example, consider an event that starts just before midnight. To users in the same time zone, the event starts today, but to users in an earlier time zone, the same event may have started yesterday.

**Make the most of the opportunity to communicate with potential users by writing a great App Store description.** In addition to describing your app accurately and highlighting the qualities you think people are most likely to appreciate, be sure to:

* **Correct all spelling, grammatical, and punctuation errors.** Although such errors don’t bother everyone, in some people they can create a negative impression of your app’s quality.
* **Keep all-capital words to a minimum.** Occasional all-capital words help draw people’s attention, but when an entire passage is capitalized, it’s difficult to read and it can be interpreted as shouting.
* **Consider describing specific bug fixes.** If a new version of your app contains bug fixes that customers have been waiting for, it can be a good idea to mention this in your description.

Integrating with iOS

Integrating with iOS means giving users a compelling, delightful experience that feels at home on the platform; it doesn’t mean creating an app that looks like a copy of a built-in app.

The best way to integrate your unique app with the platform is to understand the themes that motivate iOS—these are described in [Designing for iOS](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html#//apple_ref/doc/uid/TP40006556-CH66-SW1)—and figure out how your app should express them. As you do this, follow the guidelines in this section to help you give users the experience they expect.

Use Standard UI Elements Correctly

As much as possible, it’s a good idea to use the standard UI elements that UIKit provides. When you use standard elements instead of creating custom ones, both you and your users benefit:

* Standard UI elements automatically receive updates if iOS introduces a redesigned appearance—custom elements don’t get updated.
* Standard UI elements tend to offer various ways to customize their appearance or behavior. For example, all views (that is, objects that inherit from [UIView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/index.html#//apple_ref/occ/cl/UIView)) 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, which makes it easy to add color to your app.
* People are comfortable with the standard UI elements, so they instantly understand how to use them in your app.

To take advantage of the benefits of using standard UI elements, it’s crucial that you:

**Follow the guidelines for every UI element.** When a UI element looks and works the way people expect it to, they can depend on their prior experience to help them use it in your app. You can find UI element guidelines in [Bars](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW1), [Content Views](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ContentViews.html#//apple_ref/doc/uid/TP40006556-CH13-SW1), [Controls](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW1), and [Temporary Views](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Alerts.html#//apple_ref/doc/uid/TP40006556-CH14-SW1).

**Don’t mix UI element styles from different versions of iOS.** You don’t want to confuse users by displaying UI elements that look like they belong in a different version of iOS than the version that’s currently running on the device.

**In general, avoid creating a custom UI element that performs a standard action.** First, ask yourself why you’re creating a custom UI element that behaves exactly like a standard one. If you just want a custom look, consider changing the look of a standard element by using the UIKit appearance customization APIs or tint color. If you want a slightly different behavior, be sure to find out whether a standard element might do what you want when you adjust its properties and attributes. If you need completely custom behavior, it’s best to design a custom element that doesn’t look too similar to the standard ones.

TIP

Interface Builder makes it easy to get the standard UI elements, use the appearance customization APIs, access properties and attributes, and apply custom and system provided icons to your controls. To learn more about Interface Builder, see *Xcode Overview*.

**Don’t use system-defined buttons and icons to mean something else.** iOS provides many buttons and icons that you can use in your app. Be sure you understand the documented, semantic meaning of these buttons and icons; don’t rely on your interpretation of their appearance. (You can find the meaning of each icon in [Toolbar and Navigation Bar Buttons](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW33) and[Tab Bar Icons](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW34).)

If you can’t find a system-provided button or icon that has the appropriate meaning for a function in your app, you can create your own. For some guidelines to help you design custom icons, see[Bar Button Icons](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/BarIcons.html#//apple_ref/doc/uid/TP40006556-CH21-SW1).

**If your app enables an immersive task or experience, it may be reasonable to create completely custom controls.** This is because you’re creating a unique environment, and discovering how to control that environment is an experience users expect in such apps.

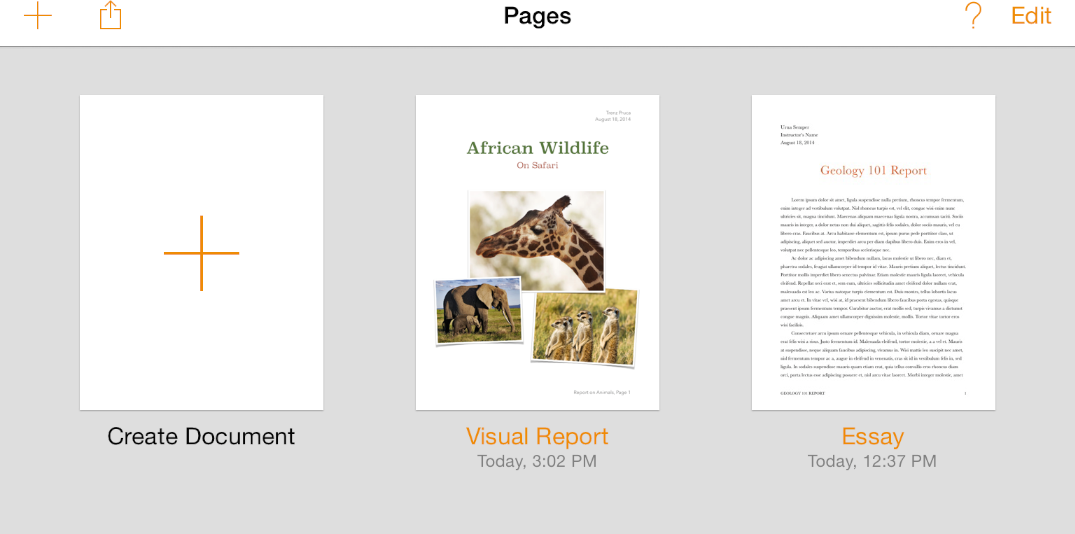
Downplay File and Document Handling

iOS apps can help people create and manipulate files, but this doesn’t mean that people should have to think about the file system on an iOS device.

If your app helps people create and edit documents, it works well to provide some sort of app-specific document library view that lets them open an existing document or create a new one. Ideally, such a library view:

* **Is highly graphical.** People should be able to easily identify the document they want by looking at visual representations of the documents onscreen.
* **Lets people make the fewest possible gestures to do what they want.** For example, people might scroll horizontally through a carousel or grid of existing documents and open the desired one with a tap.
* **Includes a new document function.** Instead of making people go somewhere else to create a new document, a document library might let them tap a placeholder image to create a new document.

For example, Pages displays the user’s documents, along with an easy way to create new documents, in a graphical library view.



TIP

You can use the Quick Look Preview feature to let people preview documents within your app, even if your app can’t open them. To learn how to provide this feature in your app, see [Quick Look](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/QuickLook.html#//apple_ref/doc/uid/TP40006556-CH43-SW1).

If your app lets people use documents that they created in other apps, you can display a modal document picker view controller to help them access these documents. The document picker view controller can display documents in the user’s iCloud Drive in addition to Document Provider extensions, which are associated with other document-creation or document-storage apps. To learn more about Document Provider extensions, see [Document Provider Extensions](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/AppExtensions.html#//apple_ref/doc/uid/TP40006556-CH67-SW5); to learn more about the document picker view controller, see [*Document Picker Programming Guide*](https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/DocumentPickerProgrammingGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40014451).

**Give people confidence that their work is always preserved unless they explicitly cancel or delete it.** If your app helps people create and edit documents, don’t require them to take an explicit save action. iOS apps should take responsibility for saving people’s input, both periodically and when they open a different document or switch away from the app.

If the main function of your app isn’t content creation—but you allow people to switch between viewing information and editing it—it can make sense to ask them to save their changes. In this scenario, it often works well to provide an Edit button in the view that displays the information. When people tap the Edit button, replace it with a Save button and add a Cancel button. The transformation of the Edit button helps remind people that they’re in an editing mode and might need to save changes, and the Cancel button gives them the opportunity to exit without saving their changes.

Be Configurable If Necessary

Some apps might need to give users a way to make setup or configuration choices, but most apps can avoid or delay doing this. Successful apps work well for most people right away, while also offering some convenient ways to adjust the user experience.

When you design your app to function the way most of your users expect, you decrease the need for settings. If you need information about the user, query the system for it instead of asking users to provide it. If you decide you must provide app settings that users rarely need to change, see [The Settings Bundle](https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html#//apple_ref/doc/uid/TP40007072-CH6-SW7) to learn how to support them in your code.

**As much as possible, offer configuration options in the main UI.** Putting options in the main UI can make sense if the options represent a primary task and if people might want to change them frequently. If people are likely to change an app’s configuration only occasionally, it can make sense to put them in a separate view.

**If necessary, help users go directly to your app’s settings in Settings.** In particular, if you display a message that describes where to find your settings, such as “Go to Settings > MyApp > Privacy > Location Services,” replace the description with a button that opens that location in Settings. To learn how to enable this behavior, see [Settings Launch URL](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/constant_group/Settings_Launch_URL).

Take Advantage of iOS Technologies

iOS provides a wealth of technologies that support common tasks and scenarios in ways that users expect. This expectation means that it’s almost always better to integrate system-supported technologies into your app than it is to design a custom approach.

Some iOS technologies—such as [Multitasking](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Multitasking.html#//apple_ref/doc/uid/TP40006556-CH38-SW1) and [VoiceOver](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/VoiceOverAccessibility.html#//apple_ref/doc/uid/TP40006556-CH45-SW1)—are system features that all apps should incorporate. Others enable specific app functionality, such as handling tickets and gift cards ([Passbook](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Passbook.html#//apple_ref/doc/uid/TP40006556-CH33-SW1)), enabling user purchases within an app ([In-App Purchase](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/InAppPurchase.html#//apple_ref/doc/uid/TP40006556-CH36-SW1)), displaying in-app advertising ([iAd Rich Media Ads](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/iAdRichMediaAds.html#//apple_ref/doc/uid/TP40006556-CH41-SW1)), integrating with [Game Center](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/GameCenter.html#//apple_ref/doc/uid/TP40006556-CH37-SW1), and supporting [iCloud](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/iCloud.html#//apple_ref/doc/uid/TP40006556-CH35-SW1).