[iOS Human Interface Guidelines](https://developer.apple.com/ios/human-interface-guidelines/)

The world’s most advanced mobile OS offers everything you need to design beautiful, engaging apps that radiate power and simplicity.

As an app designer, you have the opportunity to deliver an extraordinary product that rises to the top of the App Store charts. To do so, you'll need to meet high expectations for quality and functionality.

Three primary themes differentiate iOS from other platforms:

Clarity. Throughout the system, 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. Negative space, color, fonts, graphics, and interface elements subtly highlight important content and convey interactivity.

Deference. Fluid motion and a crisp, beautiful interface help people understand and interact with content while never competing with it. Content typically fills the entire screen, while translucency and blurring often hint at more. Minimal use of bezels, gradients, and drop shadows keep the interface light and airy, while ensuring that content is paramount.

Depth. Distinct visual layers and realistic motion convey hierarchy, impart vitality, and facilitate understanding. Touch and discoverability heighten delight and enable access to functionality and additional content without losing context. Transitions provide a sense of depth as you navigate through content.

To maximize impact and reach, keep the following principles in mind as you imagine your app’s identity.

Aesthetic Integrity

Aesthetic integrity represents how well an app’s appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

Consistency

A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation

The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback

Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors

People learn more quickly when an app’s virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control

Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it’s usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they’re in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they’re already underway.

What's New in iOS 10

With iOS 10, you can build more powerful apps than ever.

Widgets on the Search screen and Home screen. A widget provides timely, useful information or app-specific functionality without the need to open an app. In the past, people added widgets to Notification Center for quick access. Now, people add widgets to the Search screen, which is accessed by swiping to the right on the Home screen and the Lock screen. You can also show a widget above the quick action list that appears when people use 3D Touch to press your app icon on the Home screen. The design and behavior of widgets has also changed. Be sure to review and update your existing designs accordingly.

Integration with Messages. Apps can integrate with Messages by implementing a messaging extension that appears below a conversation in Messages and lets people share app-specific content with friends. Apps can share text, photos, videos, stickers, and even interactive content, such as an in-message game.

Integration with Siri. Apps can integrate with Siri and let people use their voice to perform specific types of app-specific actions, such as making calls, sending messages, and starting workouts.

Expanded Notifications. You can enhance notifications with an expanded detail view that opens when people use 3D Touch to press your notification or swipe your notification down on an unlocked device. Use this view to give people quick access to more information about a notification and the ability to take immediate action without leaving their current context.

Interface Essentials

Most iOS apps are built using components from UIKit, a programming framework that defines common interface elements. This framework lets apps achieve a consistent appearance across the system, while at the same time offering a high level of customization. UIKit elements are flexible and familiar. They’re adaptable, enabling you to design a single app that looks great on any iOS device, and they automatically update when the system introduces appearance changes. The interface elements provided by UIKit fit into three main categories:

Bars. Tell people where they are in your app, provide navigation, and may contain buttons or other elements for initiating actions and communicating information.

Views. Contain the primary content people see in your app, such as text, graphics, animations, and interactive elements. Views can enable behaviors such as scrolling, insertion, deletion, and arrangement.

Controls. Initiate actions and convey information. Buttons, switches, text fields, and progress indicators are examples of controls.

In addition to defining the interface of iOS, UIKit defines functionality your app can adopt. Through this framework, for example, your app can respond to gestures on the touchscreen and enable features such as drawing, accessibility, and printing.

iOS tightly integrates with other programming frameworks and technologies too, such as Apple Pay, HealthKit, and ResearchKit, enabling you to design amazingly powerful apps.

3D Touch

3D Touch adds an additional dimension to touch-based interaction. On supported devices, people can access additional functionality by applying varying levels of pressure to the touchscreen. Apps can respond by displaying a menu, showing additional content, or playing an animation. People don’t need to learn new gestures to interact with 3D Touch. They quickly discover the additional interactive dimension when they press lightly on the screen and get a response.

Home Screen Interaction

On the Home screen, pressing the icon of an app that supports 3D Touch displays an action view. This view lets you quickly perform common app-specific tasks and see interesting information. Calendar, for example, provides a shortcut for creating an event. It also shows the next event on your schedule. For design guidance, see Home Screen Actions and Widgets.

Peek and Pop

A peek lets people use 3D Touch to preview an item, such as a page, link, or file, in a view that appears temporarily above the current context. To peek at an item that supports this functionality, apply a little pressure to the item with your finger. Simply lift your finger to exit the peek. To open the item and see more detail, press a little harder until the item pops and fills the screen. In some peek views, you can swipe up to reveal related action buttons. For example, while peeking at a link in Safari, you can swipe up to reveal buttons for opening the link in the background, adding the link to your reading list, and copying the link.

Use peeking to provide live, content-rich previews. Ideally, peeking gives enough information about an item to augment the current task, or helps you decide whether to fully engage the item. For example, preview a link in a Mail message before deciding to open it in Safari or share it with friends. Peeking is often used in tables to view detailed row information before the row is selected.

Design big-enough peek views. Design a peek view that's large enough so that fingers don’t obscure its content. Make the peek detailed enough for people to decide whether to press a little deeper to fully open (pop) the item.

Adopt Peek and Pop consistently. If you support Peek and Pop in some places but not others, people won’t know where they can use the feature and may think there’s a problem with your app or their device.

Allow every peek to be popped. Even though peeking should give people most of the information they need, always let them transition to the pop if they decide to switch away from the current task and focus on the item. Popping should show the same thing as tapping the item.

Avoid displaying button-like elements in a peek view. If a user lifts a finger to tap an element that looks like a button, the peek disappears.

Don’t enable peeking and an edit menu for the same item. It can be confusing to users and hard for the system to detect intent when both features are enabled for one item. For additional guidance, see Edit Menus.

Provide action buttons when appropriate. Not every peek needs action buttons, but they’re a great way to offer shortcuts for common tasks. If your app already provides custom touch-and-hold actions for items, it’s good practice to include the same actions during peeks.

Avoid providing an action button that opens a peeked item. People generally press deeper to open an item they’re peeking. As a result, there’s typically no need to provide an explicit Open button.

Don’t make peek the only way to perform item actions. Not every device supports peek and pop, and some people may turn off 3D Touch. Your app should provide other ways to trigger item actions in situations like these. For example, your app could mirror a peek’s quick actions in a view that appears while touching and holding an item.

Live Photos

Apps can incorporate pressure into the photo viewing experience by supporting Live Photos. Live Photos come to life when you press them, using movement and sound to show the moments just before and after the photo was taken. For design guidance, see Live Photos.

Accessibility

iOS offers extensive accessibility features for users with vision loss, hearing loss, and other disabilities. Most UIKit-based apps can be made accessible with very little effort, allowing more people to use your app while providing an equally engaging experience for all.

Provide alternative text labels for images, icons, and interface elements. Alternative text labels aren’t visible onscreen, but they let VoiceOver audibly describe what's onscreen, making navigation easier for people with visual impairments.

Respond to accessibility preferences. If your app uses UIKit to implement its user interface, text and interface elements automatically adapt to certain accessibility preferences, such as bold and larger text. Your app should also check and respond to accessibility preferences when appropriate, such as when the option to reduce motion is enabled. Apps implementing custom fonts should attempt to match the accessibility behavior of the system fonts.

Test your app with accessibility features. In addition to text and motion changes, accessibility options can change contrast, invert colors, reduce transparency, and more. Enable these settings and observe how your app will look and behave for people who enable these features.

Include closed captions and audio descriptions. Closed captions allow the deaf and hard-of-hearing to perceive spoken dialogue and other audible content in videos. Audio descriptions provide spoken narration of important video content for the visually impaired.

For more information, see iOS Accessibility and Accessibility Programming Guide for iOS.

Audio

Whether sound is a primary aspect of your app’s experience or an embellishment, you need to know how people expect sound to behave and meet those expectations.

Users manipulate sound through the volume buttons, silence switch, headphone controls, and the onscreen volume slider. Many third-party accessories include sound controls too. Audio can be output through internal or external speakers, headphones, and even wirelessly through an AirPlay-enabled or Bluetooth device.

Silence. People switch their device to silent to avoid being interrupted by unexpected sounds, such as ringtones and incoming message sounds. They also want nonessential sounds disabled, including keyboard sounds, sound effects, game soundtracks, and other audible feedback. When the device is set to silent, only explicitly initiated sounds should occur, such as audio during media playback, alarms, and audio/video messaging.

Volume. Whether using physical device buttons or an onscreen slider, people expect changes in volume to affect all sound systemwide, including music and in-app sound effects. The only exception is the ringer volume, which is always adjusted separately when audio isn’t actively playing.

Headphones. People use headphones to hear sound privately and to free their hands. When plugging in headphones, users expect sound to reroute automatically without interruption. When unplugging headphones, they expect playback to pause immediately.

Adjust levels automatically when necessary, but not the overall volume. Your app can adjust relative, independent volume levels to achieve a great mix of audio. However, the final output should always be governed by the system volume.

Permit rerouting of audio when appropriate. People often want to select a different audio output device. For example, they may want to listen to music through their living room stereo, car radio, or Apple TV. Support this capability unless there’s a compelling reason not to.

Use the system-provided volume view to allow audio adjustments. The best way to provide interface controls for adjusting audio is to use a volume view. This view is customizable, includes a volume-level slider, and even includes a control for rerouting audio output. For developer guidance, see MPVolumeView.

Use the system’s sound services for short sounds and vibrations. For developer guidance, see System Sound Services.

Categorize your audio if sound is essential to your app. Different audio categories allow sounds to be silenced by the silence switch, to mix with other audio, or to play while your app is in the background. Pick a category based on its meaning and the current audio state of the device, and assign it to your audio sessions. For example, don’t make people stop listening to music from another app if you don’t need to. In general, it’s best to avoid changing the category while your app is running, with the exception of apps that record and playback audio at different times. For developer guidance, see Audio Session Programming Guide.

Category Meaning Behavior

Solo ambient Sound isn’t essential, but it silences other audio. For example, a game with a soundtrack. Responds to the silence switch.

Doesn’t mix with other sounds.

Doesn’t play in the background.

Ambient Sound isn’t essential, and it doesn’t silence other audio. For example, a game that lets people play music from another app during gameplay in place of the game’s soundtrack. Responds to the silence switch.

Mixes with other sounds.

Doesn’t play in the background.

Playback Sound is essential and might mix with other audio. For example, an audiobook or educational app that teaches a foreign language, which people might want to listen to after leaving the app. Doesn’t respond to the silence switch.

May or may not mix with other sounds.

Can play in the background.

Record Sound is recorded. For example, a note-taking app that offers an audio recording mode. An app of this nature might switch its category to playback if it lets people play the recorded notes. Does’t respond to the silence switch.

Doesn’t mix with other sounds.

Can record in the background.

Play and record Sound is recorded and played, potentially simultaneously. For example, an audio messaging or video calling app. Doesn’t respond to the silence switch.

May or may not mix with other sounds.

Can record and play in the background.

Resume audio playback when appropriate after an interruption occurs. Sometimes, currently playing audio is interrupted by audio from a different app. Temporary interruptions such as incoming phone calls are considered resumable. Permanent interruptions, such as a music playlist initiated by Siri, are considered nonresumable. When a resumable interruption occurs, your app should resume playback when the interruption ends if audio was actively playing when the interruption started. For example, a game playing a soundtrack and a media app in the process of playing audio should both resume.

Let other apps know when your app finishes playing temporary audio. If your app might temporarily interrupt the audio of other apps, it should flag audio sessions appropriately so other apps get notified when it’s safe to resume. For developer guidance, see the AVAudioSessionSetActiveOptionNotifyOthersOnDeactivation constant in AVFoundation.

Respond to audio controls only when it makes sense. People can control audio playback from outside of your app’s interface, such as in Control Center or with controls on their headphones, regardless of whether your app is in the foreground or background. If your app is actively playing audio, in a clear audio-related context, or connected to an AirPlay-enabeld device, it’s fine to respond to audio controls. Otherwise, your app shouldn't halt another app’s audio that may be playing when a control is activated.

Don’t repurpose audio controls. People expect audio controls to behave consistently in all apps. Never redefine the meaning of an audio control. If your app doesn’t support certain controls, then it simply shouldn’t respond to them.

Authentication

Ask users to authenticate only in exchange for value, such as personalizing the experience, accessing additional features, purchasing content, or synchronizing data. If your app requires authentication, keep the sign-in process quick, easy, and unobtrusive, so it doesn’t detract from the enjoyment of your app.

Delay sign-in as long as possible. People often abandon apps when they are forced to sign in before doing anything useful. Give them a chance to fall in love with your app before making a commitment to it. In a shopping app, let people browse your merchandise immediately upon launch and require sign-in only when they're ready to make a purchase. In a media streaming app, let people explore your content and see what you have to offer before signing in to play something.

Explain the benefits of authentication and how to sign up for your service. If your app requires authentication, display a brief, friendly explanation on the login screen that describes the reasons for the requirement and its benefits. Also, remember that not everyone using your app has an account from the start. Make sure you explain how to get one, or provide a simple in-app way to sign up.

Minimize data entry by showing appropriate keyboards. When asking for an email address, for example, show the email keyboard screen, which includes helpful data entry shortcuts.

Data Entry

Whether tapping interface elements or using the keyboard, inputting information can be a tedious process. When an app slows down the process by asking for lots of input before doing anything useful, people can get discouraged quickly, and may even abandon the app entirely.

When possible, present choices. Make data entry as efficient as possible. Consider using a picker or table instead of a text field, for example, because it’s easier to choose from a list of predefined options than to type a response.

Get information from the system whenever possible. Don’t force people to provide information that can be gathered automatically or with the user's permission, such as contact or calendar information.

Provide reasonable default values. To the extent possible, prefill fields with the most likely values. Providing good defaults minimizes decision making and speeds up the process.

Enable advancement only after collecting required values. Before enabling a Next or Continue button, make sure all required fields have values. Use the enablement of the button as a visual cue that it’s time to proceed.

Dynamically validate field values. It’s frustrating when you have to go back and correct mistakes after filling out a lengthy form. Whenever possible, check field values immediately after entry so users can correct them right away.

Require field values only when necessary. Use required fields only for information that is truly necessary to proceed.

Ease navigation through value lists. Especially in tables and pickers, it should be easy to pick a value. Consider sorting value lists alphabetically or in another logical manner that facilities speedy scanning and selection.

Show a hint in a text field to help communicate purpose. A text field can contain placeholder text—such as "Email" or "Password"—when there’s no other text in the field. Don’t use a separate label to describe a text field when placeholder text is sufficient.

Feedback

Feedback helps people know what an app is doing, discover what they can do next, and understand the results of actions.

Unobtrusively integrate status and other types of feedback into your interface. Ideally, users can get important information without taking action or being interrupted. Mail, for example, subtly displays status information in the toolbar while navigating through mailboxes of messages. This information doesn’t compete with the primary content onscreen, but can be checked at any time with a quick glance.

Avoid unnecessary alerts. An alert is a powerful feedback mechanism, but should be used only to deliver important—and ideally actionable—information. If people see too many alerts that don’t contain essential information, they quickly learn to ignore future alerts. For additional guidance, see Alerts.

Haptic Feedback

On supported devices, haptics provide a way to physically engage users with tactile feedback that gets attention and reinforces actions. Some system-provided interface elements, such as pickers, switches, and sliders, automatically provide haptic feedback as users interact with them. Your app can also ask the system to generate different types of haptic feedback. iOS manages the strength and behavior of this feedback.

Notification

Success

Warning

Failure

Impact

Light

Medium

Heavy

Selection

Selection

Success. Indicates that a task or action, such as depositing a check or unlocking a vehicle, has completed.

Use haptics judiciously. Overuse can cause confusion and diminish the significance of feedback.

In general, provide haptic feedback in response to user-initiated actions. It’s easy for people to correlate haptics with actions they initiated. Arbitrary feedback can feel disconnected and be misinterpreted.

Don’t redefine feedback types. To ensure a consistent experience, use feedback types as intended. Don’t, for example, use "impact" feedback to notify the user that a task has succeeded. Instead, use the "success" variation of "notification" feedback.

Fine tune your visual experience for haptics. Provide visual and haptic feedback together to create a deeper connection between actions and results. Make sure animations are sharp and precise, to visually match what the user feels.

Don’t rely on a single mode of communication. Not all devices support the full range of haptic feedback, and people can disable the feature entirely in Settings if they choose. In addition, haptic feedback occurs only when the device is active and your app is frontmost. Supplement haptics with visual and audible cues to ensure that important information isn’t missed.

Use haptics when visual feedback may be occluded. Some interactions, such as dragging an object to a location onscreen, are hidden by the user’s finger. Consider generating feedback that lets the user know when they’ve reached a particular location or value.

Prepare the system before initiating feedback. Because there may be some latency involved when providing haptic feedback, it’s best to get the system ready shortly before requesting the feedback. Otherwise, the haptics might come too late and feel disconnected from the user's actions or what they’re seeing on the screen.

Synchronize haptics with accompanying sound. Haptics don’t automatically synchronize with sounds. If you want an accompanying sound, you’re responsible for synchronizing it.

File Handling

People shouldn’t need to think about the file system while creating, viewing, and manipulating files. If your app works with files, downplay file handling as much as possible.

Instill confidence that work is always preserved unless canceled or deleted. In general, don’t make people explicitly save files. Instead, save changes automatically at regular intervals, when opening and closing files, and when switching to another app. In some cases, such as while editing an existing file, save and cancel options may still make sense for the sake of confirming when the edits are actually captured.

Don't provide an option to create local-only files. Users often expect all of their files to be available on all of their devices. Whenever possible, your app should support cloud-based file storage through a service such as iCloud.

Implement an intuitive and graphical file browsing interface. Ideally, use the system’s familiar document picker for browsing. If you implement a custom file browser, make sure it’s intuitive and efficient. File browsers work best when they’re highly graphical, providing visual representations of files. To speed navigation, minimize gestures, and consider providing a new document button, so people don’t need to go elsewhere to create a new document.

Let users preview files without leaving your app. You can use Quick Look to let people view the contents of Keynote, Numbers, and Pages documents, PDFs, images, and certain other types of files, even if your app doesn’t actually open them. See Quick Look.

When appropriate, share files with other apps. If it makes sense, your app can share its files with other apps through a document provider extension. Your app can also let people browse and open files from other apps. For developer guidance, see Document Picker Programming Guide.

First Launch Experience

Launch time is your first opportunity to onboard new users and reconnect with returning ones. Design a launch experience that’s fast, fun, and educational.

Provide a launch screen. A launch screen appears the moment your app starts, giving the impression that your app is fast and responsive, while allowing initial content to load. This screen is quickly replaced by the first screen of your app, so it is should closely resemble this screen with the exception of localizable text and interactive elements. To learn more, see Launch Screen.

Launch in the appropriate orientation. If your app supports both portrait and landscape modes, it should launch using the device’s current orientation. If your app only runs in one orientation, it should always launch in that orientation and let people rotate the device if necessary. Unless there’s a compelling reason not to, an app in landscape mode should orient itself correctly, regardless of whether the Home button is on the left or right. For additional guidance, see Layout.

Get to the action quickly. Avoid showing a splash screen, menus, and instructions that make it take longer to reach content and start using your app. Instead, let people dive right in. If your app needs tutorials or intro sequences, provide a way to skip them and don't show them to returning users.

Anticipate the need for help. Proactively look for times when people might be stuck. A game, for example, could casually show useful tips when paused or when a character isn’t advancing. Let users replay tutorials in case they miss something the first time.

Stick to the essentials in tutorials. It’s fine to provide guidance for beginners, but education isn’t a substitute for great app design. First and foremost, make your app intuitive. If too much guidance is needed, revisit the design of your app.

Make learning fun and discoverable. Learning by doing is a lot more fun and effective than reading a list of instructions. Use animation and interactivity to teach gradually and in context. Avoid displaying screenshots that appear interactive.

Avoid asking for setup information up front. People expect apps to just work. Design your app for the majority and let the few that want a different configuration adjust settings to meet their needs. To the extent possible, derive setup information from device settings and defaults, or through a synchronization service, such as iCloud. If you must ask for setup information, prompt for it in-app the first time, and let users modify it later in your app’s settings.

Avoid showing in-app licensing agreements and disclaimers. Let the App Store display agreements and disclaimers before your app is downloaded. If you must include these items within your app, integrate them in a balanced way that doesn’t disrupt the user experience.

Restore the previous state when your app restarts. Don't make people retrace steps to reach their previous location in your app. Preserve and restore your app’s state so they can continue where they left off.

Don’t ask people to rate your app too quickly or too often. Asking for a rating too soon or too frequently is annoying and decreases the amount of useful feedback you receive. To encourage well-considered feedback, give people time to form an opinion about your app before asking for a rating. Always provide a way to opt out of rating prompts and never force users to rate your app.

Don’t encourage rebooting. Restarting takes time and makes your app seem unreliable and hard to use. If your app has memory or other issues that make it difficult to run unless the system has just booted, you need to address those issues.

Gestures

People interact with an iOS device by performing gestures on the touchscreen. These gestures elicit a close personal connection with content and enhance the sense of direct manipulation of onscreen objects. People generally expect the following standard gestures to work the same across the system and in every app.

Tap. Activates a control or selects an item.

Drag. Moves an element from side-to-side or drags an element across the screen.

Flick. Scrolls or pans quickly.

Swipe. When performed with one finger, returns to the previous screen, reveals the hidden view in a split view controller, reveals the Delete button in a table-view row, or reveals actions in a peek. When performed with four fingers on an iPad, switches between apps.

Double tap. Zooms in and centers content or an image, or zooms out if already zoomed in.

Pinch. Zooms in when pinching outward, zooms out when pinching inward.

Touch and hold. When performed in editable or selectable text, displays a magnified view for cursor positioning. When performed in certain views, such as a collection view, enters a mode that allows items to be rearranged.

Shake. Initiates undo or redo.

As a general rule, use standard gestures. People are familiar with the standard gestures and don’t appreciate being forced to learn different ways to do the same thing. In games and other immersive apps, custom gestures can be a fun part of the experience. In other apps, it’s best to use standard gestures so extra effort isn’t needed to discover or remember them.

Don’t block systemwide gestures. In addition to the standard gestures, some additional gestures invoke systemwide actions, such as revealing Control Center or Notification Center. People rely on these gestures to work in every app.

Avoid using standard gestures to perform nonstandard actions. Unless your app is a game in active gameplay, redefining the meaning of standard gestures leads to confusion and complexity.

Offer shortcut gestures to supplement, not replace, interface-based navigation and actions. Whenever possible, offer a simple, visible way to navigate or perform an action, even if it means an extra tap or two. Many system apps include a navigation bar that provides a clear, tappable button back to the previous screen. But users can also navigate back by swiping from the side of the screen. On iPad, people can exit to the Home screen by pressing the Home button, or by using a four-finger pinching gesture.

Use multifinger gestures to enhance the experience of some apps. Although gestures involving multiple fingers at once aren’t appropriate for every app, they can enrich the experience in some apps, such as games and drawing apps. For example, a game might include multiple onscreen controls, such as a joystick and firing buttons, that can be operated simultaneously.

Loading

When content is loading, a blank or static screen can make it seem like your app is frozen, resulting in confusion and frustration, and potentially causing people to leave your app.

Make it clear when loading is occurring. At minimum, show an activity spinner that denotes something is happening. Even better, display explicit progress so the user can gauge how long they’ll be waiting.

Educate or entertain people to mask loading time. Consider showing hints about gameplay, entertaining video sequences, or interesting placeholder graphics.

Customize loading screens. Although standard progress indicators are usually OK, they can sometimes feel out of context. Consider designing a more immersive experience through custom animations and elements that match the style of your app or game.

Show content as soon as possible. Don’t make people wait for content to load before seeing the screen they're expecting. Show the screen immediately, and use placeholder text, graphics, or animations to identify where content isn't available yet. Replace these placeholder elements as the content loads. Whenever possible, preload upcoming content in the background, such as while an animation is playing or the user is navigating a level or menu.

For additional guidance, see Progress Indicators.

Modality

Modality creates focus by preventing people from doing other things until they complete a task or dismiss a message or view. Action sheets, alerts, and activity views provide modal experiences. When a modal view appears onscreen, the user must make a choice by tapping a button or otherwise exiting the modal experience. Some apps implement modal views, such as while editing an event in Calendar or choosing a bookmark in Safari. A modal view can occupy the entire screen, an entire parent view, such as a popover, or a portion of the screen. A modal view typically includes completion and cancel buttons that exit the view.

Minimize the use of modality. Generally, people prefer to interact with apps in nonlinear ways. Consider creating a modal context only when it’s critical to get someone’s attention, when a task must be completed or abandoned to continue using the app, or to save important data.

Provide an obvious and safe way to exit a modal task. Make sure people always know the outcome of their action when they dismiss a modal view.

Keep modal tasks simple, short, and narrowly focused. Don’t create an app within your app. If a modal task is too complex, people can lose sight of the task they suspended when they entered the modal context. Be especially wary of creating modal tasks that involve a hierarchy of views because users can get lost and forget how to retrace their steps. If a modal task must contain subviews, provide a single path through the hierarchy and a clear path to completion. Avoid using Done buttons for things other than completing the task.

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

Reserve alerts for delivering essential—and ideally actionable—information. An alert interrupts the experience and requires a tap to dismiss, so it’s important for people to feel that the intrusion is warranted. To learn more, see Alerts.

Respect notification preferences. In Settings, people specify how they want to receive notifications from your app. Abide by these preferences so they aren’t tempted to turn off your app’s notifications entirely.

Don’t display a modal view above a popover. With the possible exception of an alert, nothing should appear over a popover. In rare cases when you need to present a modal view after action is taken in a popover, close the popover before displaying the modal view.

Coordinate modal view appearance with your app. A modal view may include a navigation bar, for example. When this is the case, use the same appearance as the navigation bar in your app.

Choose an appropriate modal view style. You can use any of these styles:

Full screen. Covers the entire screen. Use for a potentially complex task that can be completed within the context of the modal view.

Page sheet. Partially covers the underlying content on larger devices held in landscape orientation. All uncovered areas are dimmed to prevent interaction with them. Covers the entire screen on smaller devices and in portrait orientation. Use for a potentially complex task that can be completed within the context of the modal view.

Form sheet. Appears centered on screen, but may be repositioned if a keyboard is visible. All uncovered areas are dimmed to prevent interaction with them. May cover the entire screen on smaller devices. Use for gathering information.

Current context. Appears at the same size as its parent view. Use for displaying modal content within a split view pane, popover, or other view that is not full-screen.

Choose an appropriate transition style for revealing a modal view. Use a transition style that coordinates with your app and enhances the awareness of the temporary context shift. The default transition vertically slides the modal view up from the bottom of the screen and back down once dismissed. The flip-style transition appears to flip a view horizontally to reveal the modal view. Visually, the modal view looks like the back of the current view. It flips back once dismissed. Use consistent modal transition styles throughout your app.

For modal view developer guidance, see UIViewController and UIPresentationController.

Navigation

People tend to be unaware of an app’s navigation until 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. Navigation should feel natural and familiar, and shouldn’t dominate the interface or draw focus away from content. In iOS, there are three main styles of navigation.

Hierarchical navigation. Make one choice per screen until you reach a destination. To go to another destination, you must retrace your steps or start over from the beginning and make different choices. Settings and Mail use this navigation style.

Flat navigation. Switch between multiple content categories. Music and App Store use this navigation style.

Content-driven or experience-driven navigation. Move freely through content, or the content itself defines the navigation. Games, books, and other immersive apps generally use this navigation style.

Some apps combine multiple navigation styles. For example, an app that uses flat navigation may implement hierarchical navigation within each category.

Always provide a clear path. People should always know where they are in your app and how to get to their next destination. Regardless of navigation style, it’s essential that the path through content is logical, predictable, and easy to follow. In general, give people one path to each screen. If they need to see a screen in multiple contexts, consider using an action sheet, alert, popover, or modal view. To learn more, see Action Sheets, Alerts, Popovers, and Modality.

Design an information structure that makes it fast and easy to get to content. Organize your information structure in a way that requires a minimum number of taps, swipes, and screens.

Use touch gestures to create fluidity. Make it easy to move through your interface with minimum friction. For example, you could let people swipe from the side of the screen to return to the previous screen.

Use standard navigation components. Whenever possible, use standard navigation controls such as page controls, tab bars, segmented controls, table views, collection views, and split views. Users are already familiar with these controls, and will intuitively know how to get around your app.

Use a navigation bar to traverse a hierarchy of data. The navigation bar’s title can show the current position in the hierarchy, and the back button makes it easy to return to the previous location. For specific guidance, see Navigation Bars.

Use a tab bar to present peer categories of content or functionality. A tab bar lets people quickly and easily switch between categories, regardless of the current location. For specific guidance, see Tab Bars.

Use a page control when you have multiple pages of the same type of content. A page control clearly communicates the number of pages available and which one is currently active. The Weather app uses a page control to show location-specific weather pages. For specific guidance, see Page Controls.

TIP

Segmented controls and toolbars don’t enable navigation. Use a segmented control to organize information into different categories. Use a toolbar to provide controls for interacting with the current context. For additional information on these types of elements, see Segmented Controls and Toolbars.

Ratings and Reviews

Ratings and reviews help people make informed decisions when considering whether to try out your app. Positive ratings and reviews can mean more downloads of your app, and customer feedback gives you insight into real world usage that helps direct future development efforts.

Delivering a great overall experience is the best way to encourage positive ratings and reviews, but it’s also important to ask for feedback at appropriate times. Keep these considerations in mind when asking people to rate your app.

Ask for a rating only after the user has demonstrated engagement with your app. For example, prompt the user upon the completion of a game level or productivity task. Never ask for a rating on first launch or during onboarding. Allow ample time to form an opinion.

Don’t interrupt the user, especially when they’re performing a time-sensitive or stressful task. Look for logical pauses or stopping points, where a rating request makes the most sense.

Don’t be a pest. Repeated rating prompts can be irritating, and may even negatively influence the user’s opinion of your app. Allow at least a week or two between rating requests and only prompt again after the user has demonstrated additional engagement with your app.

System Rating and Review Prompts

The system offers a consistent, nonintrusive way for apps to request ratings and reviews. To use this feature, you simply identify places in your app's user experience where it makes sense to ask for feedback. If the user hasn't already given feedback and a request hasn't been made too recently, the system displays an in-app prompt that asks for a rating and an optional written review. The user can supply feedback or dismiss the prompt with a single tap. (In Settings, the user can also opt out of receiving these rating prompts for all apps they have installed.) The system automatically limits the display of the prompt to three occurrences per app within a 365-day period.

Prefer the system-provided prompt. The system's rating prompt offers a familiar, efficient experience that's designed to engage the user with minimal impact.

Don't use buttons or other controls to request feedback. Since the system limits how often rating prompts occur, attempting to request feedback in response to a control may result in no rating prompt being displayed.

For developer guidance, see SKStoreReviewController in StoreKit.

TIP

Responding to reviews is a great way to communicate with users, address concerns, and potentially improve your app’s rating. For best practices, see Responding to Reviews on the App Store.

Ratings and Reviews

Ratings and reviews help people make informed decisions when considering whether to try out your app. Positive ratings and reviews can mean more downloads of your app, and customer feedback gives you insight into real world usage that helps direct future development efforts.

Delivering a great overall experience is the best way to encourage positive ratings and reviews, but it’s also important to ask for feedback at appropriate times. Keep these considerations in mind when asking people to rate your app.

Ask for a rating only after the user has demonstrated engagement with your app. For example, prompt the user upon the completion of a game level or productivity task. Never ask for a rating on first launch or during onboarding. Allow ample time to form an opinion.

Don’t interrupt the user, especially when they’re performing a time-sensitive or stressful task. Look for logical pauses or stopping points, where a rating request makes the most sense.

Don’t be a pest. Repeated rating prompts can be irritating, and may even negatively influence the user’s opinion of your app. Allow at least a week or two between rating requests and only prompt again after the user has demonstrated additional engagement with your app.

System Rating and Review Prompts

The system offers a consistent, nonintrusive way for apps to request ratings and reviews. To use this feature, you simply identify places in your app's user experience where it makes sense to ask for feedback. If the user hasn't already given feedback and a request hasn't been made too recently, the system displays an in-app prompt that asks for a rating and an optional written review. The user can supply feedback or dismiss the prompt with a single tap. (In Settings, the user can also opt out of receiving these rating prompts for all apps they have installed.) The system automatically limits the display of the prompt to three occurrences per app within a 365-day period.

Prefer the system-provided prompt. The system's rating prompt offers a familiar, efficient experience that's designed to engage the user with minimal impact.

Don't use buttons or other controls to request feedback. Since the system limits how often rating prompts occur, attempting to request feedback in response to a control may result in no rating prompt being displayed.

For developer guidance, see SKStoreReviewController in StoreKit.

TIP

Responding to reviews is a great way to communicate with users, address concerns, and potentially improve your app’s rating. For best practices, see Responding to Reviews on the App Store.

Requesting Permission

Users must grant permission for an app to access personal information, including the current location, calendar, contact information, reminders, and photos. Although people appreciate the convenience of using an app that has access to this information, they also expect to have control over their private data. For example, people like being able to automatically tag photos with their physical location or find nearby friends, but they also want the option to disable such features.

Request personal data only when your app clearly needs it. It’s natural to be suspicious of a request for personal information, especially if there’s no obvious need for it. Make sure permission requests occur only when people are using features that clearly need personal data. For example, an app might only request access to the current location when activating a location tracking feature.

Explain why your app needs the information if it’s not obvious. You can add custom text to the system-provided permission request alert. Make the text specific and polite, so people don’t feel pressured. Keep the text short, and use sentence case. There’s no need to include your app name. The system already identifies your app as the one making the request.

Request permission at launch only when necessary for your app to function. Users won’t be bothered by this request if it’s obvious that your app depends on their personal information to operate.

Don’t request location information unnecessarily. Before accessing location information, check the system to see whether Location Services is enabled. With this knowledge, you can delay the alert until a feature truly requires it, or perhaps avoid the alert altogether.

To learn how to implement location features, see Location and Maps Programming Guide.

Settings

Some apps may need to provide a way to make setup or configuration choices, but most apps can avoid or delay doing so. Successful apps work well for most people right away, while also offering some convenient ways to adjust the experience. When you design your app to function the way most people expect, you decrease the need for settings.

Infer what you can from the system. If you need information about the user, device, or environment, query the system for it whenever possible instead of asking the user. For example, instead of asking someone to enter their zip code so you can present local options, ask permission to use their current location.

Thoughtfully prioritize configuration options within your app. Your app’s main screen is a good place for options that are essential or that change frequently. Secondary screens are better for options that change only occasionally.

Provide shortcuts to Settings when appropriate. If your app includes text that directs users to Settings, such as “Go to Settings > MyApp > Privacy > Location Services,” provide a button that opens that location automatically. To learn how to implement this behavior, see Settings Launch URL in UIApplication.

Expose infrequently changed configuration options in Settings. The Settings app is a central location for making configuration changes throughout the system, but people must leave your app to get there. It’s far more convenient to adjust settings directly within your app. If you must provide settings that rarely require change, see Implementing an iOS Settings Bundle in Preferences and Settings Programming Guide for developer guidance.

Terminology

Every word in your app is part of a conversation with your users. Use this conversation to help them feel comfortable in your app.

Use familiar, understandable words and phrases. Technology can be intimidating. Avoid acronyms and technical jargon that people might not understand. Use what you know about your audience to determine whether certain words or phrases are appropriate. In general, apps that appeal to everyone should steer clear of highly technical language. Such language may be appropriate in apps that target a more advanced or technical crowd.

Keep interface text clear and concise. People absorb short, direct text quickly and easily and don’t appreciate being forced to read long passages to accomplish a task. Identify the most important information, express it succinctly, and present it prominently so people don’t have to read too much to find what they’re looking for or figure out what to do next.

Avoid language that might sound patronizing. Avoid we, our, me, and my (for example "our tutorial" and "my workouts"). They're sometimes interpreted as insulting or patronizing.

Strive for an informal, friendly tone. An informal, approachable style echoes the way you speak with people over lunch. Use contractions occasionally, and you and your to address the user directly.

Be careful when using humor. Remember that people are likely to read the text in your interface many times, and what might seem clever at first can become irritating over time. Also remember that humor in one culture doesn't necessarily translate well to other cultures.

Use relevant and consistent language and imagery. Always make sure guidance is appropriate for the current context. If someone’s using an iPad, for example, don’t show iPhone tips or graphics. Use language that’s consistent with the platform. You tap, flick, swipe, pinch, and drag content on the touchscreen. You press physical buttons and content that responds to 3D Touch. You rotate and shake the device.

Refer to dates accurately. It’s appropriate to use friendly terms like today and tomorrow, but these terms can be confusing or inaccurate if you don’t account for the current locale. Consider an event that starts just before midnight. In one time zone, the event may start today. In another time zone, the same event may have started yesterday. Generally, dates should reflect the time zone of the person viewing the event. However, in some cases, such as in a flight tracking app, it may be clearer to explicitly show the start date and time zone where the flight originates.

Identify interactive elements appropriately. People should be able to tell at a glance what an element does. When labeling buttons and other interactive elements, use action verbs, such as Connect, Send, and Add.

Undo and Redo

Many apps allow people to shake the device to undo and redo certain operations, such as typing or deleting. When initiated in this manner, an alert asks the user to confirm or cancel the undo or redo operation.

Briefly and precisely describe the operation to be undone or redone. The undo and redo alert title automatically includes a prefix of “Undo ” or “Redo ” (including the trailing space). You need to provide an additional word or two that describe what’s being undone or redone, to appear after this prefix. For example, you might create alert titles such as “Undo Name” or “Redo Address Change.”

If you use the shake gesture for undo and redo, don’t use it for other actions. Although you can programmatically give multiple meanings to the shake gesture, you run the risk of confusing people and making your app unpredictable.

Provide undo and redo buttons sparingly. It’s confusing when apps provide multiple ways to perform the same task. If your app truly warrants dedicated undo and redo buttons, use the standard system-provided icons and put them in an expected location, such as a navigation bar.

Perform undo and redo operations in the current context only. Undo and redo should have a clear and immediate effect on the current context, not an earlier one.

For developer guidance, see Undo Architecture.

Multitasking

Multitasking lets you quickly switch from one app to another at any time through a multitasking interface on an iOS device, or by using a multifinger gesture on an iPad. On iPad, multitasking also lets you use two apps at once in Slide Over, Split View, or Picture in Picture mode. Slide Over is accessed by swiping from the right side of the screen to temporarily use a second app without leaving the context of the current app, such as to quickly view your Mail inbox while you’re using Safari. Split View lets you use two side-by-side apps at the same time, and Picture in Picture lets you watch a video while working in another app.

Designing an app that thrives in a multitasking environment hinges on your app harmoniously coexisting with other apps on the device. This means your app shouldn’t use too much CPU, memory, screen space, or other system resources. It should respond well to sudden interruptions and audio from other apps, transition to and from the background quickly and smoothly, and behave responsibly when operating in the background.

Be prepared for interruptions, and be ready to resume. Your app can be interrupted at any time. When an interruption occurs, your app should save the current state quickly and precisely so people can seamlessly continue where they left off when they return. For developer guidance, see Preserving Your App’s Visual Appearance Across Launches in App Programming Guide for iOS.

Make sure your interface works with a double-high status bar. Certain features, such as in-progress phone calls, audio recording, and tethering display an additional status bar at the top of the screen. In unprepared apps, this added height can cause layout problems by covering or pushing down other interface elements. Test your app with these features enabled to ensure that your interface responds appropriately and still looks great.

Pause activity that requires attention or active participation. If your app is a game or a media-viewing app, for example, make sure people don’t miss anything when they switch to another app. When they switch back, let them continue as if they’d never left.

Respond appropriately to external audio. Occasionally, your app’s audio may be interrupted by audio from another app or the system itself. For example, an incoming phone call or a music playlist initiated by Siri may interrupt your app’s audio. When situations like these occur, your app’s response should meet people’s expectations. For primary audio interruptions, such as playing music, podcasts, or audiobooks, your app should pause its audio indefinitely. For shorter interruptions, such as GPS directional notifications, your app should temporarily lower its audio volume or pause its audio and resume when the interruption ends. For additional guidance, see Audio.

Finish user-initiated tasks in the background. When someone starts a task, they expect it to finish even if they switch away from your app. If your app is in the middle of performing a task that doesn’t need additional input, complete it in the background before suspending.

Use notifications sparingly. Your app can arrange for notifications to be sent at specific times, whether your app is suspended, running in the background, or not running at all. Notifications are fine for communicating important information, but avoid pestering people with too many. For example, don’t show a notification every time your app finishes a task in the background. Instead, let people check on tasks by returning to your app. For additional guidance, see Notifications.

For iPad-specific developer guidance, see Adopting Multitasking Enhancements on iPad.

Notifications

Apps can use notifications to provide timely and important information anytime, whether the device is locked or in use. For example, notifications may occur when a message has arrived, an event is about to occur, new data is available, or the status of something has changed. People see notifications on the lock screen, at the top of the screen while using the device, and in Notification Center, which is opened by swiping down from the top edge of the screen. Each notification includes the app name, a small app icon, and a message. Notifications may also be accompanied by a sound, and may cause a badge to appear or update on the corresponding app’s icon.

The behavior of notifications is managed in Settings on an app-by-app basis. For any app that supports notifications, you can enable or disable the feature entirely. You can also enable visibility in Notification Center and on the lock screen, enable app icon badging, and choose one of these notification styles:

Banner. Appears at the top of the screen for a few seconds while the device is in use, then disappears.

Alert. Appears at the top of the screen while the device is in use and stays there until manually dismissed.

Tapping a notification when the device is unlocked, or swiping it to the side when the device is locked, dismisses the notification, removes it from Notification Center, opens the corresponding app, and shows related information. Tapping a new email notification on an unlocked device, for example, opens Mail and shows the new message.

On an unlocked device, swiping a notification up or letting it disappear dismisses the notification and may remove it from Notification Center.

Applying pressure to a notification using 3D Touch or, on an unlocked device, swiping a notification down, opens an expanded detail view. This view is customizable and can include up to four buttons for taking action. A to-do list app, for example, might deliver a task notification with a detail view that includes actions for deferring the task and marking it as complete. An Calendar event notification provides a Snooze action that briefly delays the event's alarm.

TIP

Notifications can be local or remote. Local notifications originate and are delivered on the same device. A to-do list app might use local notifications to alert someone about an upcoming meeting or due date. Remote notifications, also called push notifications, come from a server. A multiplayer game might use remote notifications to let players know when it’s their move.

People must explicitly opt into receiving notifications from every app that supports them—they’re asked to do so the first time they use the app. If someone opts out, they can always visit Settings to opt in.

Provide useful, informative notifications. People enable notifications to get quick updates, so focus on providing information of value. Use complete sentences, sentence case, proper punctuation, and don’t truncate your message—the system does this automatically, if necessary. Avoid telling people to open your app, navigate to specific screens, tap specific buttons, and perform other tasks that are hard to remember once the notification is dismissed.

Don’t send multiple notifications for the same thing, even if the user hasn't responded. People attend to notifications at their convenience. If you send multiple notifications for the same thing, you fill up Notification Center, and users may turn off notifications from your app.

Don’t include your app name or icon. The system automatically shows this information at the top of each notification.

Use badging to supplement notifications, not to denote critical information. Keep in mind that badging of your app can be turned off. If your app relies on badging to communicate important information, you run the risk of people missing it.

Keep badges up to date. Update your app’s badge as soon as the corresponding information is read. You don’t want people to think there’s new information available, only to find that they’ve already seen it. Note that reducing a badge’s count to zero removes all related notifications from Notification Center.

Provide a sound to supplement your notifications. Sound is a great way to get someone's attention when they’re not looking at the screen. A to-do list app might play an alert sound, for example, when it’s time to perform an important task. Your app can use a custom sound or a built-in alert sound for this. If you use a custom sound, make sure it’s short, distinctive, and professionally produced. See Preparing Custom Alert Sounds in Local and Remote Notification Programming Guide. Keep in mind that people can optionally disable notification alert sounds. They can also enable a vibration that accompanies the sound—this must be manually enabled and can’t be enabled programmatically by your app.

Consider providing a detail view. A notification detail view provides more information about a notification, as well as the ability to take immediate action without leaving the current context to open your app. This view should include useful information, be recognizable, and feel like a natural extension of your app. It can contain photos, videos, and other content, and can be dynamically updated while displayed. For example, a ridesharing app could display a map here that shows a car approaching your current location.

Provide intuitive, beneficial actions. A notification detail view can include up to four action buttons. These buttons should be for performing common, time-saving tasks that eliminate the need to open your app. Use short, title-case names that clearly describe the action results. A notification detail view can also display an onscreen keyboard for collecting information needed to take action. For example, a messaging app might let people respond directly from a new message notification.

Avoid providing destructive actions. Think carefully before providing destructive actions in a notification detail view. If you must provide them, make sure people have enough context to avoid unintended consequences. Actions identified as destructive appear in red.

For developer guidance, see Local and Remote Notification Programming Guide.

Printing

Your app can take advantage of the system’s built-in AirPrint technology to enable wireless printing of images, PDFs, and other content to compatible printers. While viewing printable content in an AirPrint-enabled app, people typically tap an action button in a navigation bar or toolbar and then tap a Print action to display a printer view. This view provides a list of available printers and any customizable options, such as the number of copies and page range, and provides a button to initiate printing.

Make printing discoverable. Enable printing through the system-provided Action button if your app has a toolbar or navigation bar. Users are familiar with this button and use it to print in other apps. If your app doesn’t have a toolbar or navigation bar, design a custom print button instead.

Enable printing only when it's possible to print. If there’s nothing onscreen to print or no printers are available, don’t show a Print action when someone taps the Action button. If your app implements a custom print button, disable or hide it when printing isn’t possible.

Provide printing options that add value. Think about the options people might want to specify when they print your content. Consider options for selecting a page range and requesting multiple copies. Enable additional options, such as front and back printing, if it makes sense and the printer supports them.

For developer guidance, see Drawing and Printing Guide for iOS and UIPrintInteractionController.

Quick Look

Within your app, Quick Look lets people preview Keynote, Numbers, Pages, and PDF documents, as well as images and other types of files, even if your app doesn't support those file formats. Mail uses Quick Look for viewing attachments. After downloading an attachment, Mail shows the attachment’s icon and file name within the message. Tapping the icon displays a preview of the attachment.

Present previews appropriately for the current context. On iPhone, if your app has a navigation bar, slide the preview into place like any other view in your app’s hierarchy. On iPads, or if your app doesn’t have its own navigation bar, open the preview in a full-screen modal view that includes a navigation bar. With both methods, the navigation bar includes a button for exiting Quick Look, as well as preview-specific buttons for performing actions such as sharing and markup. If your app includes a toolbar, any preview-specific buttons appear here instead of in the navigation bar.

For developer guidance, see Document Interaction Programming Topics for iOS and Quick Look.

Siri

Your app can integrate with Siri so that the user can perform certain tasks in response to spoken commands and questions. Siri handles the language processing and semantic analysis to turn spoken requests into instructions your app can handle. Your app defines the tasks it supports, validates information received, provides information for Siri to present, and takes action. Information from your app’s response is spoken by Siri and appears in the Siri interface. If appropriate, your app can provide a custom user interface for Siri to display. A fitness app, for example, might provide custom workout information.

The following types of iOS apps can integrate with Siri.

Type of app Supported Siri interactions

Audio and video calling Initiate calls.

Search the call history.

Messaging Send messages.

Read received messages.

Search for messages.

Payment services Send payments.

Request payments.

Pay bills.

Search for bills.

Photo management Search for photos and show them in the app.

Fitness activities Start, pause, resume, end, and cancel workouts.

Transportation services Book rides.

Provide ride status information.

Automotive apps that integrate with vehicles Activate hazard lights or honk the horn.

Lock and unlock the doors.

Check the current fuel or power level.

Automotive apps that integrate with CarPlay Change the car’s audio source.

Change the car’s climate.

Change the car’s defroster settings.

Change the car’s seat temperature.

Change the car’s radio station.

Strive for a voice-driven experience that doesn’t require touching or looking at the screen. People often interact with Siri through a headset, through their car, or from across the room. To the extent possible, let users complete tasks without unlocking their phone.

Respond quickly and minimize interaction. People use Siri for convenience and expect a fast response. Present efficient, focused choices that reduce the possibility of additional prompting.

Take people directly to content. Transitions from Siri to your app should go directly to the expected destination. Don’t show intermediary screens or messages that slow down the experience.

Be relevant and accurate. Make your app’s response relevant to the user’s current request and expectations. For example, if the user asks Siri to send a message with your app, send a message. Don't perform a different action.

Be appropriate. Never include content that could be offensive or demeaning.

When a request has a financial impact, default to the safest and least expensive option. Never deceive the user or misrepresent information. For a purchase with multiple pricing levels, don’t default to the most expensive. At the point where a user is making a payment, don’t charge extra fees without informing them.

Make sure your custom interface integrates well with Siri. It’s fine to use your app’s colors, imagery, and other design elements to communicate your brand, but any custom interface elements should still feel like they belong in Siri.

Don’t include your app name or icon in your custom interface. The system automatically shows this information.

Increase accuracy with custom vocabulary. Help Siri learn more about the actions your app performs by defining specific terms people might actually use in requests, such as contact names, photo tags, photo album names, ride options, and workout names. These terms should be nongeneric and unique to your app. Never include other app names, terms that are obviously connected with other apps, inappropriate language, or reserved phrases, such as “Hey Siri.” Note that any terms you define are used by Siri to help resolve requests, but aren’t guaranteed to be recognized.

Provide example requests. Provide Siri with example phrases to show in the guide that appears when you tap the Help button in the Siri interface. Use these phrases to teach people the easiest and most efficient way to use Siri with your app.

Don’t advertise. Your app’s Siri experience should never include advertisements, marketing, or in-app purchase sales pitches.

Don’t attempt to mimic or manipulate Siri. Your app should never impersonate Siri, attempt to reproduce the functionality Siri provides, or provide a response that appears to come from Apple.

For developer guidance, see SiriKit Programming Guide.

TV Providers

TV App Integration

The TV app provides global access to favorite, recently played, and recommended movies and television shows from across the system. Use this guidance to offer users a highly convenient and consistent entertainment experience.

Beginning and Resuming Playback

The TV app automatically opens your app and sends it a notification when the user initiates content playback within your app.

Ensure a smooth transition to your app. The TV app fades to black when transitioning to your app and doesn’t show your app’s launch screen. Your app should maintain visual continuity with this transition by immediately presenting its own black screen before starting to play or resume content.

Show the expected content immediately. The user expects the chosen content to begin playing as soon as the transition to your app is complete. Jump right from your app’s black screen into content. Avoid presenting splash screens, detail screens, intro animations, or any other barriers that make it take longer to reach content. This is especially important when resuming playback.

Don’t ask whether the user wants to resume playback. If playback can be resumed, it should automatically do so without prompting for confirmation.

Make sure content plays for the correct user. If your app supports multiple user profiles, the TV app can specify a profile when issuing a playback request. Your app should automatically switch to this profile before starting playback. If a playback request doesn’t specify a profile, ask the user to choose one before playback begins so this information can be provided in the future.

Loading Content

If loading takes more than two seconds, consider showing a single, black loading screen with a centered activity spinner and no surrounding content.

Avoid loading screens whenever possible. A loading screen is unnecessary if your content loads quickly.

Start playback as soon as possible. If a loading screen is needed, it should be visible only until enough content has loaded for playback to begin. Continue loading remaining content in the background.

Design black loading screens. Loading screens may appear during the transition to playing content. They should appear black so they blend with the fade-to-black functionality of the TV app.

Minimize content on loading screens. If you must include branding or imagery on a loading screen, do so minimally and in a way that maintains the black background and moves seamlessly to playback.

Exiting Playback

The user remains in your app after exiting playback, rather than returning to the TV app. This experience shouldn’t cause the user to become disoriented.

Show a contextually relevant screen. When exiting playback, display a detail screen for the content the user was just watching and include an option to resume playback. If a detail screen isn’t available, show a menu that includes the content the user was just watching or your app’s main menu.

Be prepared for an immediate exit. Prepare an exit screen as soon as possible after receiving a playback notification so the user sees it even if they exit immediately after playback begins.

Single Sign-On

Many popular TV providers allow users to sign in to their accounts at the system level, eliminating the need to authenticate on an app-by-app basis. If your app requires TV provider authentication, use this capability to provide the most efficient onboarding experience.

Avoid displaying a sign-out option when the user is signed in at the system level. If your app must include a sign-out option, invoking it should direct the user to Settings > TV Provider to sign out of their account.

Never instruct users to sign out by adjusting privacy controls. The TV provider controls in Settings > Privacy aren’t a sign out mechanism. These settings let users manage the apps that can access their TV provider account.

Animation

Beautiful, subtle animation throughout iOS builds a visual sense of connection between people and content onscreen. When used appropriately, animation can convey status, provide feedback, enhance the sense of direct manipulation, and help users visualize the results of their actions.

Use animation and motion effects judiciously. Don’t use animation for the sake of using animation. Excessive or gratuitous animation can make people feel disconnected or distracted, especially in apps that don’t provide an immersive experience. iOS uses motion effects, such as a parallax effect, to create the perception of depth on the Home screen and in other areas. These effects can increase understanding and enjoyment, but overusing them can make an app feel disorienting and difficult to control. If you implement motion effects, always test the results to make sure they work well.

Strive for realism and credibility. People tend to accept artistic license, but they can feel disoriented when movement doesn’t make sense or appears to defy physical laws. If someone reveals a view by sliding it down from the top of the screen, for example, they should be able dismiss the view by sliding it back up.

Use consistent animation. A familiar, flowing experience keeps users engaged. They're accustomed to the subtle animation used throughout iOS, such as smooth transitions, fluid changes in device orientation, and physics-based scrolling. Unless you’re creating an immersive experience, such as a game, custom animation should be comparable to the built-in animations.

Make animations optional. When the option to reduce motion is enabled in accessibility preferences, your app should minimize or eliminate application animations.

Branding

Successful branding involves more than just adding brand assets to your app. Great apps express unique brand identity through smart font, color, and image decisions. Provide enough branding to give people context in your app, but not so much that it becomes a distraction.

Incorporate refined, unobtrusive branding. People use your app to be entertained, get information, or get things done, not to watch an advertisement. For the best experience, subtly incorporate your brand through your app’s design. Using colors from your app icon throughout your interface is one great way to provide context in your app.

Don’t let branding get in the way of great app design. Above all, make your app feel like an iOS app. Ensure that it's intuitive, easy to navigate, easy to use, and focuses on content. Even if your app is available on other platforms, avoid diluting your design by focusing too much on consistent branding.

Defer to content over branding. Showing a persistent bar at the top of the screen that does nothing but display brand assets means there’s less room for viewing content. Instead, consider less intrusive ways to implement branding, such as using a custom color scheme or font, or subtly customizing the background.

Resist the temptation to display your logo throughout your app. Avoid displaying a logo throughout your app unless it’s necessary for providing context. This is especially important in navigation bars, where a title is more helpful.

Adhere to Apple's trademark guidelines. Apple trademarks should not appear in your app name or imagery. See Apple Trademark List and Guidelines for Using Apple Trademarks.

TIP

The App Store offers more opportunities to highlight your brand. For guidance, see App Store Marketing Guidelines.

Color

In iOS, color can indicate interactivity, impart vitality, and provide visual continuity. Look to the system’s color scheme for guidance when picking app tint colors that look great individually and in combination, on both light and dark backgrounds.

Use complementary colors throughout your app. The colors in your app should work well together, not conflict or distract. If pastels are essential to your app’s style, for example, use a coordinating set of pastels.

In general, choose a limited color palette that coordinates with your app logo. Subtle use of color is a great way to communicate your brand.

Consider choosing a key color to indicate interactivity throughout your app. In Notes, interactive elements are yellow. In Calendar, interactive elements are red. If you define a key color that denotes interactivity, make sure other colors don’t compete with it.

Avoid using the same color for interactive and noninteractive elements. If interactive and noninteractive elements have the same color, it’s hard for people to know where to tap.

Consider how artwork and translucency affect nearby colors. Variations in artwork sometimes warrant changes to nearby colors in order to maintain visual continuity and prevent interface elements from becoming overpowering or underwhelming. Maps, for example, displays a light color scheme when using map mode but switches to a dark color scheme when satellite mode is activated. Colors can also appear different when placed behind a translucent element, or when applied to a translucent element, such as a toolbar.

Apply color profiles to your images. The default color space on iOS is Standard RGB (sRGB). To ensure that colors are correctly matched to this color space, make sure your images include embedded color profiles.

Use wide color to enhance the visual experience on compatible devices. Wide color displays support a P3 color space that can produce richer, more saturated colors than sRGB. When the experience calls for vivid colors, use the Display P3 color profile at 16 bits per pixel (per channel) and export in .png format.

TIP

On Macs with wide color displays, you can use the system color picker to select and preview P3 colors, and to compare them with sRGB colors.

Test your app’s color scheme under a variety of lighting conditions. Lighting varies significantly both indoors and outdoors, based on room ambiance, time of day, the weather, and more. Colors you see on your computer won’t always look the same when your app is used in the real world. Always preview your app under multiple lighting conditions, including outdoors on a sunny day, to see how colors appear. If necessary, adjust colors to provide the best possible viewing experience in the majority of use cases.

Consider how the True Tone display affects color. The True Tone display uses ambient light sensors to automatically adjust the white point of the display to adapt to the lighting conditions of the current environment. Apps that focus primarily on reading, photos, video, and gaming can strengthen or weaken this effect by specifying a white point adaptivity style. For developer guidance, see Information Property List Key Reference.

Be aware of colorblindness and how different cultures perceive color. People see colors differently. Many colorblind people, for example, find it difficult to distinguish red from green (and either color from gray), or blue from orange. Avoid using these color combinations as the only way to distinguish between two states or values. For example, instead of using red and green circles to indicate offline and online, use a red square and a green circle. Some image-editing software includes tools that can help you proof for colorblindness. Also consider how your use of color might be perceived in other countries and cultures. In some cultures, for example, red is used to communicate danger. In others, red has positive connotations. Make sure the colors in your app send the appropriate message.

Use sufficient color contrast ratios. Insufficient contrast in your app makes content hard to read for everyone. Icons and text might blend with the background, for example. An online color contrast calculator can help you accurately analyze the color contrast in your app, to ensure that it meets optimal standards. Strive for a minimum contrast ratio of 4.5:1, although 7:1 is preferred because it meets more stringent accessibility standards.

Layout

People generally want the ability to use their favorite apps on all of their devices and in any orientation. In iOS, interface elements and layouts can be configured to automatically change shape and size on different devices, during multitasking on iPad, in split view, and when the screen is rotated. It’s essential that you plan ahead and design an app that provides a great experience in any context.

Maintain focus on the current content during context changes. Content is your highest priority. Changing focus when the environment changes can be disorienting, frustrating, and make people feel like they’ve lost control of the app.

Ensure primary content is clear at its default size. People shouldn’t have to scroll horizontally to read important text, or zoom to see primary images, unless they choose to change the size.

Maintain an overall consistent appearance throughout your app. In general, elements with similar functions should look similar.

Use visual weight and balance to convey importance. Large items catch the eye and appear more important than smaller ones. Larger items are also easier to tap, which is especially important when an app is used in distracting surroundings, such as in the kitchen or a gym. In general, place principal items in the upper half of the screen and—in a left-to-right reading context—near the left side of the screen.

Use alignment to ease scanning and to communicate organization and hierarchy. Alignment makes an app look neat and organized, helps people focus while scrolling, and makes it easier to find information. Indentation and alignment can also indicate how groups of content are related.

Avoid gratuitous layout changes. Just because someone rotates a device doesn’t mean the entire layout needs to change. For example, if your app shows a grid of images in portrait mode, it doesn’t have to present the same images as a list in landscape mode. Instead, it might simply adjust the dimensions of the grid. Try to maintain a comparable experience in all contexts.

If possible, support both portrait and landscape orientations. People prefer to use apps in different orientations, so it’s best when you can fulfill that expectation.

If your app supports only one orientation, support both variants. If it’s essential that your app run in a single orientation, make sure it supports both orientation variants. For example, if your app only runs in landscape mode, it should be usable regardless of whether the Home button is on the left or the right. If the device is rotated 180 degrees while using your app, your app should respond by rotating its content 180 degrees. If your app doesn't rotate automatically when someone holds the device wrong, they'll know instinctively to rotate it. You don't need to tell them.

Customize your app’s response to rotation according to context. A game that lets people move a character by rotating the device, for example, probably shouldn’t switch orientations during gameplay. It could, however, display menus and intro sequences based on the current orientation.

Provide ample spacing for interactive elements. Try to maintain a minimum tappable area of 44pt x 44pt for all controls.

Be prepared for text size changes. People 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 text usage in your app, see Typography.

For developer guidance on adaptivity, see Auto Layout Guide.

Typography

San Francisco is the system font on iOS. There are two variants of this font: SF UI Text for text 19 points or smaller, and SF UI Display for text 20 points or larger. When you use the system font for text in labels and other interface elements, iOS automatically applies the most appropriate font variant based on the point size. It also switches fonts automatically as needed and respects accessibility settings. Download the San Francisco fonts here.

Emphasize important information. Use font weight, size, and color to highlight the most important information in your app.

If possible, use a single font. Mixing several different fonts can make your app seem fragmented and sloppy. Consider using one font and just a few styles and sizes.

Use built-in text styles whenever possible. The built-in text styles let you express content in ways that are visually distinct, while retaining optimal legibility. These styles are based on the system fonts and allow you to take advantage of key typographic features, such as Dynamic Type, which automatically adjusts tracking and leading for every font size. iOS includes the following text styles:

Make sure custom fonts are legible. Custom typefaces are supported on iOS, but are often tough to read. Unless your app has a compelling need for a custom font, such as for branding purposes or to create an immersive gaming experience, stick with the system fonts. If you do use a custom font, make sure it’s readable.

Implement accessibility features for custom fonts. System fonts automatically react to accessibility features, such as when bold text is enabled. Apps using custom fonts should implement the same behavior by checking whether accessibility features are enabled and registering for notifications when they change. See Accessibility.

Dynamic Type Sizes

Both variants of the SF UI typeface were designed to be highly legible at both small and large sizes. Dynamic Type provides additional flexibility by letting readers choose their preferred text size.

Prioritize content when responding to text-size changes. Not all content is equally important. When someone chooses a larger 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.

Download a dynamic type size table in Resources.

Font Usage and Tracking

Use the correct font variant in interface mockups. Use SF UI Text for text 19 points or smaller. Use SF UI Display for text 20 points or larger. Adjust tracking—the spacing between letters—appropriately, as follows:

App Icon

Every app needs a beautiful and memorable icon that attracts attention in the App Store and stands out on the Home screen. Your icon is the first opportunity to communicate, at a glance, your app’s purpose. It also appears throughout the system, such as in Settings and search results.

Embrace simplicity. Find a single element that captures the essence of your app and express that element in a simple, unique shape. Add details cautiously. If an icon’s content or shape is overly complex, the details can be hard to discern, especially at smaller sizes.

Provide a single focus point. Design an icon with a single, centered point that immediately captures attention and clearly identifies your app.

Design a recognizable icon. People shouldn’t have to analyze the icon to figure out what it represents. For example, the Mail app icon uses an envelope, which is universally associated with mail. Take time to design a beautiful and engaging abstract icon that artistically represents your app’s purpose.

Keep the background simple and avoid transparency. Make sure your icon is opaque, and don’t clutter the background. Give it a simple background so it doesn’t overpower other app icons nearby. You don’t need to fill the entire icon with content.

Use words only when they’re essential or part of a logo. An app’s name appears below its icon on the Home screen. Don’t include nonessential words that repeat the name or tell people what to do with your app, like "Watch" or "Play." If your design includes any text, emphasize words that relate to the actual content your app offers.

Don’t include photos, screenshots, or interface elements. Photographic details can be very hard to see at small sizes. Screenshots are too complex for an app icon and don’t generally help communicate your app’s purpose. Interface elements in an icon are misleading and confusing.

Don’t use replicas of Apple hardware products. Apple products are copyrighted and can’t be reproduced in your icons or images. In general, avoid displaying replicas of devices, because hardware designs tend to change frequently and can make your icon look dated.

Don’t place your app icon throughout the interface. It can be confusing to see an icon used for different purposes throughout an app. Instead, consider incorporating your icon’s color scheme. See Color.

Test your icon against different wallpapers. You can’t predict which wallpaper people will choose for their Home screen, so don’t just test your app against a light or dark color. See how it looks over different photos. Try it on an actual device with a dynamic background that changes perspective as the device moves.

Keep icon corners square. The system applies a mask that rounds icon corners automatically.

App Icon Sizes

Every app must supply both small and large app icons. The small icons are used on the Home screen and throughout the system once your app is installed. The large icon is used by the App Store.

Device or context Icon size

iPhone 180px × 180px (60pt × 60pt @3x)

120px × 120px (60pt × 60pt @2x)

iPad Pro 167px × 167px (83.5pt × 83.5pt @2x)

iPad, iPad mini 152px × 152px (76pt × 76pt @2x)

App Store 1024px × 1024px

Provide different sized small icons for different devices. Make sure that your app icon looks great on all the devices you support.

Mimic your small app icon with your large app icon. Although the large icon is used differently than the small one, it’s still your app icon. It should generally match the smaller version in appearance, although it can be subtly richer and more detailed since there are no visual effects applied to it.

Spotlight, Settings, and Notification Icons

Every app should also provide a small icon that iOS can display when the app name matches a term in a Spotlight search. Additionally, apps with settings should provide a small icon to display in the built-in Settings app, and apps that support notifications should provide a small icon to display in notifications. All icons should clearly identify your app—ideally, they should match your app icon. If you don’t provide these icons, iOS might shrink your main app icon for display in these locations.

Device Spotlight icon size

iPhone 120px × 120px (40pt × 40pt @3x)

80px × 80px (40pt × 40pt @2x)

iPad Pro, iPad, iPad mini 80px × 80px (40pt × 40pt @2x)

Device Settings icon size

iPhone 87px × 87px (29pt × 29pt @3x)

58px × 58px (29pt × 29pt @2x)

iPad Pro, iPad, iPad mini 58px × 58px (29pt × 29pt @2x)

Device Notification icon size

iPhone 60px × 60px (20pt × 20pt @3x)

40px × 40px (20pt × 20pt @2x)

iPad Pro, iPad, iPad mini 40px × 40px (20pt × 20pt @2x)

Don’t add an overlay or border to your Settings icon. iOS automatically adds a 1-pixel stroke to all icons so that they look good on the white background of Settings.

TIP

If your app creates custom documents, you don't need to design document icons because iOS uses your app icon to create document icons automatically.

User-Selectable App Icons

For some apps, customization is a feature that evokes a personal connection and enhances the user experience. If it provides value in your app, you can let people select an alternate app icon. For example, a sports app might offer icons for different teams or an app with light and dark modes might offer corresponding light and dark icons. Note that your app icon can only be changed at the user’s request and the system always provides the user with confirmation of such a change.

Provide visually consistent alternate icons in all necessary sizes. Like your primary app icon, each alternate app icon is delivered as a collection of related images that vary in size. When the user chooses an alternate icon, the appropriate sizes of that icon replace your primary app icon on the Home screen, in Spotlight, and elsewhere in the system. To ensure that alternate icons appear consistently throughout the system—the user shouldn't see one version of your icon on the Home screen and a completely different version in Settings, for example—provide them in the same sizes you provide for your primary app icon (with the exception of the large App Store icon). See App Icon Sizes.

For developer guidance, see the setAlternateIconName method of UIApplication.

Custom Icons

If your app includes tasks or modes that can’t be represented by a system icon (see System Icons), or if the system icons don’t match your app’s style, you can create your own icons. A custom icon, sometimes called a template, discards color information and uses a mask to produce the appearance you see onscreen in a navigation bar, tab bar, toolbar, or Home screen quick action.

Create simple, recognizable designs. Too many details can make an icon appear sloppy or unreadable. Strive for a design most people will interpret correctly and won’t find offensive.

Design a solid color icon with transparency, anti-aliasing, and no drop shadow. iOS ignores all color information, so there’s no need to use more than one fill color. Allow transparency to define the shape of the icon.

Keep your icons consistent. Whether you use only custom icons or mix custom and system icons, all icons in your app should be the same in terms of size, level of detail, perspective, and stroke weight. If you want an icon to look like it's related to the iOS icon family, use a very thin stroke to draw it. A 1pt stroke (2px for @2x resolution) works well for most icons.

Provide two versions of custom tab bar icons. Provide icons for both the selected and unselected states. The selected icon is often a filled-in version of the unselected icon, but some designs call for variations to this approach. For example, Apple apps sometimes invert icon interiors, increase or reduce strokes, use color variations, and enclose the icon within a shape, such as a circle.

Don’t use text in a tab bar icon. If you need to show text, display a title beneath the tab and adjust its placement accordingly.

Don’t use replicas of Apple hardware products. Apple products are copyrighted and can’t be reproduced in your icons or images. In general, avoid displaying replicas of devices because hardware designs tend to change frequently and can make your icon look dated.

Provide alternative text labels for icons. Alternative text labels aren’t visible onscreen, but they let VoiceOver audibly describe what's onscreen, making navigation easier for people with visual impairments.

Custom Icon Sizes

Navigation bar and toolbar icon size Tab bar icon size

Recommended 75px × 75px (25pt × 25pt @3x) 75px × 75px (25pt × 25pt @3x)

50px × 50px (25pt × 25pt @2x) 50px × 50px (25pt × 25pt @2x)

Maximum 83px × 83px (27.67pt × 27.67pt @3x) 144px × 96px (48pt × 32pt @3x)

56px × 56px (28pt × 28pt @2x) 96px × 64px (48pt × 32pt @2x)

Image Size and Resolution

iOS uses a coordinate system to place content onscreen. This coordinate system is based on measurements in points, which map to pixels in the display. On a standard-resolution screen, one point (1/72 of an inch) is equal to one pixel. High-resolution screens have a higher pixel density. Because there are more pixels in the same amount of physical space, there are more pixels per point. As a result, high-resolution displays require images with more pixels.

You must supply high-resolution images for all artwork in your app, for all devices your app supports. Depending on the device, you accomplish this by multiplying the number of pixels in each image by a specific scale factor.

Device Scale factor

iPhone 6s Plus and iPhone 6 Plus @3x

All other high-resolution iOS devices @2x

A standard resolution image has a scale factor of 1.0 and is referred to as an @1x image. High resolution images have a scale factor of 2.0 or 3.0 and are referred to as @2x and @3x images. Suppose you have a standard resolution @1x image that’s 100px × 100px, for example. The @2x version of this image would be 200px × 200px. The @3x version would be 300px × 300px.

Launch Screen

A launch screen appears instantly when your app starts up. The launch screen is quickly replaced with the first screen of your app, giving the impression that your app is fast and responsive. The launch screen isn’t an opportunity for artistic expression. It’s solely intended to enhance the perception of your app as quick to launch and immediately ready for use. Every app must supply a launch screen.

Because device screen sizes vary, launch screen sizes vary too. To accommodate this, you can provide a launch screen as an Xcode storyboard or as a set of static images for the devices your app supports. Using an Xcode storyboard is the recommended approach, as storyboards are flexible and adaptable. You can use a single storyboard to manage all of your launch screens. To learn about implementing adaptable interfaces, see Auto Layout Guide.

Design a launch screen that’s nearly identical to the first screen of your app. If you include elements that look different when the app finishes launching, people can experience an unpleasant flash between the launch screen and the first screen of the app.

Avoid including text on your launch screen. Because launch screens are static, any displayed text won’t be localized.

Downplay launch. People are likely to switch apps frequently, so design a launch screen that doesn’t draw attention to the app launching experience.

Don’t advertise. The launch screen isn’t a branding opportunity. Don’t design an entry experience that looks like a splash screen or an "About" window. Don’t include logos or other branding elements unless they’re a static part of your app’s first screen.

Static Launch Screen Images

It’s best to use an Xcode storyboard for your launch screen, but you can provide a set of static images if necessary. Create static images in different sizes for different devices, and be sure to include the status bar region.

Device Portrait size Landscape size

iPhone 7 Plus, iPhone 6s Plus 1080px × 1920px 1920px × 1080px

iPhone 7, iPhone 6s 750px × 1334px 1334px × 750px

iPhone SE 640px × 1136px 1136px × 640px

12.9-inch iPad Pro 2048px × 2732px 2732px × 2048px

9.7-inch iPad Pro, iPad Air 2, iPad mini 4, iPad mini 2 1536px × 2048px 2048px × 1536px

System Icons

iOS provides lots of small icons—representing common tasks and types of content—for use in navigation bars, tab bars, toolbars, and Home screen quick actions. It’s a good idea to use these built-in icons as much as possible because they're familiar to people.

Provide alternative text labels for icons. Alternative text labels aren’t visible onscreen, but they let VoiceOver audibly describe what's onscreen, making navigation easier for people with visual impairments.

Navigation Bar and Toolbar Icons

TIP

You can use text instead of icons to represent items in a navigation bar or toolbar. For example, Calendar uses “Today,” “Calendars,” and “Inbox” in the toolbar. You can also use a fixed space element to provide padding between navigation and toolbar icons.

Icon Button name Function

### Shows a modal view containing share extensions, action extensions, and tasks, such as Copy, Favorite, or Find, that are useful in the current context. Creates a new item. Shows app-specific bookmarks. Takes a photo or video, or shows the Photo Library. Closes the current view or ends edit mode without saving changes. Opens a new view in edit mode. Saves the state and closes the current view, or exits edit mode. Enters edit mode in the current context. Fast-forwards through media playback or slides. Moves an item to a new destination, such as a folder. Pauses media playback or slides. Always store the current location when pausing, so playback can resume later. Begins or resumes media playback or slides. Redoes the last action that was undone. Refreshes content. Use this icon sparingly, as your app should refresh content automatically whenever possible. Sends or routes an item to another person or location. Moves backwards through media playback or slides. Saves the current state. Displays a search field. Stops media playback or slides. Deletes the current or selected item. Undoes the last action. **Tab Bar Icons**

| **Icon** | **Button name** | **Function** |
| --- | --- | --- |
| Bookmarks tab bar icon | Bookmarks | Shows app-specific bookmarks. |
| Contacts tab bar icon | Contacts | Shows the person’s contacts. |
| Downloads tab bar icon | Downloads | Shows active or recent downloads. |
| Favorites tab bar icon | Favorites | Shows the person’s favorite items. |
| Featured tab bar icon | Featured | Shows content featured by the app. |
| History tab bar icon | History | Shows recent actions or activity. |
| More tab bar icon | More | Shows additional tab bar items. |
| Most recent tab bar icon | Most Recent | Shows the most recent items. |
| Most viewed tab bar icon | Most Viewed | Shows the most popular items. |
| Recents tab bar icon | Recents | Shows content or items recently accessed within a specific period of time. |
| Search tab bar icon | Search | Enters a search mode. |
| Top rated tab bar icon | Top Rated | Shows the highest-rated items. |

For developer guidance, see [UITabBarItem](https://developer.apple.com/reference/uikit/uitabbaritem).

Quick Action Icons

Icon Button name Function

Add Creates a new item.

Alarm Sets or displays an alarm.

Audio Denotes or adjusts audio.

Bookmark Creates a bookmark or shows bookmarks.

Capture Photo Captures a photo.

Capture Video Captures a video.

Cloud Denotes, displays, or initiates a cloud-based service.

Compose Composes new editable content.

Confirmation Denotes that an action is complete.

Contact Chooses or displays a contact.

Date Displays a calendar or event, or performs a related action.

Favorite Denotes or marks a favorite item.

Home Indicates or displays a home screen. Indicates, displays, or routes to a physical home.

Invitation Denotes or displays an invitation.

Location Denotes the concept of location or accesses the current geographic location.

Love Denotes or marks an item as loved.

Mail Creates a Mail message.

Mark Location Denotes, displays, or saves a geographic location.

Message Creates a new message or denotes the use of messaging.

Pause Pauses media playback. Always store the current location when pausing, so playback can resume later.

Play Begins or resumes media playback.

Prohibit Denotes that something is disallowed.

Search Enters a search mode.

Share Shares content with others or to social media.

Shuffle Indicates or initiates shuffle mode.

Task Denotes an uncompleted task or marks a task as complete.

Task Completed Denotes a completed task or marks a task as not complete.

Time Denotes or displays a clock or timer.

Update Updates content.

Navigation Bars

A navigation bar appears at the top of an app screen, below the status bar, and enables navigation through a series of hierarchical app screens. When a new screen is displayed, a back button, often labeled with the title of the previous screen, appears on the left side of the bar. Sometimes, the right side of a navigation bar contains a control, such as an Edit or a Done button, for managing content within the active view. In a split view, a navigation bar may appear in a single pane of the split view. Navigation bars are translucent, may have a background tint, and can be configured to hide when appropriate, such as when the keyboard is onscreen, a gesture occurs, or a view resizes.

TIP

Use a toolbar when navigation isn’t needed or you want multiple controls for managing content. See Toolbars.

Consider showing the title of the current view in the navigation bar. In most cases, a title provides context by letting people know what they’re looking at. However, 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 needed.

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 additional guidance, see Segmented Controls.

Avoid crowding a navigation bar with too many controls. In general, a navigation bar should contain no more than the view’s current title, a 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 include a title or any controls other than the segmented control.

Don’t include multisegment breadcrumb paths. The back button always performs a single action—returning to the previous screen. If you think people might get lost without the full path to the current screen, consider flattening your app’s hierarchy.

Give text-titled buttons enough room. If your navigation bar includes multiple text buttons, the text of those buttons may appear to run together, making the buttons indistinguishable. Add separation by inserting a fixed space item between the buttons. For developer guidance, see the UIBarButtonSystemItemFixedSpace constant value in UIBarButtonItem.

Consider temporarily hiding the navigation bar when displaying full-screen content. The navigation bar can be distracting when you want to focus on content. Temporarily hide the bar to provide a more immersive experience. The Maps app hides the navigation bar and other interface elements when viewing a full-screen map. If you implement this type of behavior, let users restore the navigation bar with a simple gesture, such as a tap.

Use the standard back button. People know that the standard back button lets them retrace steps through a hierarchy of information. However, if you implement a custom back button, make sure it still looks like a back button, behaves intuitively, matches the rest of your interface, and is consistently implemented throughout your app. If you replace the system-provided back button chevron with a custom image, supply a custom mask image too. iOS uses this mask to animate the button title during transitions.

For developer guidance, see UINavigationBar.

Search Bars

People use a search bar to search through a large collection of values. There are two styles of search bars—prominent (the default) and minimal. Contacts employs the prominent style, which includes a noticeable tinted background. Photos uses the minimal style, which tends to blend more with the surrounding interface. Search bars are translucent by default, but can be made opaque. When applicable, a search bar can be configured to automatically hide the navigation bar.

A search bar contains a single search field, which can include placeholder text and a Clear button. In addition to the search field, a search bar can contain a Cancel button for exiting the search.

Use a search bar rather than a text field to implement search. A text field doesn’t have the standard search bar appearance that people expect.

Enable the Clear and Cancel buttons. Most search bars include a Clear button that erases the contents of the field, and a Cancel button that immediately terminates the search.

Pick a search bar style that reflects the importance of search in your app. If search is a primary function in your app, use the default, prominent style of search bar. If search is infrequent, use the minimal style.

If necessary, provide hints and context in a search bar. The search field can contain placeholder text—such as “Search Clothing, Shoes and Accessories” or simply “Search”—as a reminder of the context being searched. A succinct, one-line sentence with appropriate punctuation can also be displayed directly above the search bar to provide guidance. Stocks, for example, uses text above the search field to let people know they can enter a company name or stock symbol.

Consider providing helpful shortcuts and other content below a search bar. Use the area under a search bar to help people get to content faster. Safari, for example, shows your bookmarks as soon as you tap the search field. Select one to go right to it without entering any search terms. Stocks shows a list of results as you type into the search field. Tap one at any time without typing any more characters.

For developer guidance, see UISearchController and UISearchBar.

Scope Bars

A scope bar can be added to a search bar to let people refine the scope of a search. A scope bar adopts the appearance of its search bar.

Favor improving search results over including a scope bar. A scope bar can be useful when there are clearly defined categories in which to search. However, it’s best to improve search results so scoping isn’t necessary.

For developer guidance, see UISearchBar.

Status Bars

The status bar appears along the upper edge of the screen and displays useful information about the device’s current state, such as the time, cellular carrier, network status, and battery level. The actual information shown in the status bar varies depending on the system configuration.

Use the system-provided status bar. People expect the status bar to be consistent systemwide. Don’t replace it with a custom status bar.

Coordinate the status bar style with your app design. The visual style of the status bar’s text and indicators is either light or dark, and can be set globally for your app or individually for different screens. A dark status bar works well above light-colored content and a light status bar works well above dark-colored content.

Obscure content under the status bar. By default, the background of the status bar is transparent, allowing content beneath to show through. Keep the status bar readable and don’t imply that content behind it is interactive. There are several common techniques for doing this:

Use a navigation bar in your app, which automatically displays a status bar background and ensures that content doesn’t appear under the status bar.

Display a custom image, such as a gradient or solid color, behind the status bar.

Apply a blur to the content behind the status bar.

Consider temporarily hiding the status bar when displaying full-screen media. A status bar can be distracting when users are trying to focus on media. Temporarily hide these elements to provide a more immersive experience. The Photos app, for example, hides the status bar and other interface elements during browsing of full-screen photos.

Avoid permanently hiding the status bar. Without a status bar, people must leave your app to check the time or see if they have a Wi-Fi connection. Let people redisplay a hidden status bar by using a simple, discoverable gesture. When browsing full-screen photos in the Photos app, a single tap shows the status bar again.

Communicate lengthy network activity in the status bar. When your app uses the network, especially for lengthy operations, show the network activity status bar indicator so people know activity is occurring. See Network Activity Indicators.

For developer guidance, see the UIStatusBarStyle constant in UIApplication and the preferredStatusBarStyle property in UIViewController.

Tab Bars

A tab bar appears at the bottom of an app screen and provides the ability to quickly switch between different sections of an app. Tab bars are translucent, may have a background tint, maintain the same height in all screen orientations, and are hidden when a keyboard is displayed. A tab bar may contain any number of tabs, but the number of visible tabs varies based on the device size and orientation. If some tabs can’t be displayed due to limited horizontal space, the final visible tab becomes a More tab, which reveals the additional tabs in a list on a separate screen.

In general, use a tab bar to organize information at the app level. A tab bar is a good way to flatten your information hierarchy and provide access to several peer information categories or modes at once.

Don’t remove or disable a tab when its function is unavailable. If tabs are available in some cases but not in others, your app’s interface becomes unstable and unpredictable. Ensure that all tabs are always enabled, and explain why a tab’s content is unavailable. For example, if there are no songs on an iOS device, the My Music tab in the Music app explains how to download songs.

Use a tab bar strictly for navigation. Tab bar buttons should not be used to perform actions. If you need to provide controls that act on elements in the current view, use a toolbar instead. See Toolbars.

Avoid having too many tabs. Each additional tab reduces the tappable area for selecting a tab and increases the complexity of your app, making it harder to locate information. Although a More tab can display extra tabs, this requires additional taps and is a poor use of space. Include essential tabs only, and use the minimum tabs necessary for your information hierarchy. Too few tabs can be a problem too, as it can make your interface appear disconnected. In general, use between three and five tabs on iPhone. A few more are acceptable on iPad.

Use badging to communicate unobtrusively. You can display a badge—a red oval containing white text and either a number or an exclamation point—on a tab to indicate that new information is associated with that view or mode.

Always switch contexts in the attached view. To keep your interface predictable, selecting a tab should always affect the view that’s directly attached to the tab bar, not another view elsewhere on screen. For example, selecting a tab on the left side of a split view shouldn’t cause the right side of the split view to suddenly change. Selecting a tab in a popover shouldn’t cause the view behind the popup to change.

For developer guidance, see UITabBar.

TIP

It’s important to understand the difference between a tab bar and a toolbar, because both types of bars appear at the bottom of an app screen. A tab bar lets the user switch quickly between different sections of an app, such as the Alarm, Stopwatch, and Timer tabs in the Clock app. A toolbar contains buttons for performing actions related to the current context, such as creating an item, deleting an item, adding an annotation, or taking a photo. See Toolbars. Tab bars and toolbars never appear together in the same view.

Toolbars

A toolbar appears at the bottom of an app screen and contains buttons for performing actions relevant to the current view or content within it. Toolbars are translucent, may have a background tint, and often hide when people are unlikely to need them. For example, in Safari, the toolbar hides when you begin scrolling the page since you are likely reading. You can show it again by tapping the bottom of the screen. Toolbars are also hidden when a keyboard is onscreen.

Provide relevant toolbar buttons. A toolbar should contain frequently used commands that make sense in the current context.

Consider whether icons or text-titled buttons are right for your app. Icons work well when you need more than three toolbar buttons. When you have three buttons or fewer, text can sometimes be clearer. In Calendar, for example, text is used because icons would be confusing. The use of text also allows the Inbox button to show a count of calendar and event invitations.

Avoid using a segmented control in a toolbar. Segmented controls let people switch contexts, whereas a toolbar is specific to the current screen. If you need to provide a way to switch contexts, consider using a tab bar instead. See Tab Bars.

Give text-titled buttons enough room. If your toolbar includes multiple buttons, the text of those buttons may appear to run together, making the buttons indistinguishable. Add separation by inserting fixed space between the buttons. For developer guidance, see the UIBarButtonSystemItemFixedSpace constant value in UIBarButtonItem.

For developer guidance, see UIToolbar.

TIP

It’s important to understand the difference between a toolbar and a tab bar, because both types of bars appear at the bottom of an app screen. A toolbar contains buttons for performing actions related to the current context, such as creating an item, deleting an item, adding an annotation, or taking a photo. A tab bar lets the user switch quickly between different sections of an app, for example, the Alarm, Stopwatch, and Timer tabs in the Clock app. See Tab Bars. Toolbars and tab bars never appear together in the same view.

Apple UI Design Resources

The Apple UI Design Resources include Photoshop and Sketch templates, and other UI materials for quickly designing iOS apps. The included resources are comprehensive and accurately depict the full range of UIKit controls, views and glyphs available to developers using the iOS SDK. They make it easier to understand how iOS apps are constructed and how to design apps that match the iOS design language. Included icon and glyph production files have been preconfigured to automate asset production using Sketch slices or Adobe Generator for Photoshop CC.

Action Sheets

An action sheet is a specific style of alert that appears in response to a control or action, and presents a set of two or more choices related to the current context. Use an action sheet to let people initiate tasks, or to request confirmation before performing a potentially destructive operation. On smaller screens, an action sheet slides up from the bottom of the screen. On larger screens, an action sheet appears all at once as a popover.

Provide a Cancel button if it adds clarity. A Cancel button instills confidence when the user is abandoning a task. Cancel buttons should always be included in action sheets at the bottom of the screen.

Make destructive choices prominent. Use red for buttons that perform destructive or dangerous actions, and display these buttons at the top of an action sheet.

Avoid enabling scrolling in an action sheet. If an action sheet has too many options, people must scroll to see all of the choices. Scrolling requires extra time to make a choice and is hard to do without inadvertently tapping a button.

For developer guidance, see the UIAlertControllerStyleActionSheet constant in UIAlertController.

Activity Views

An activity is a task, such as Copy, Favorite, or Find, that’s useful in the current context. Once initiated, an activity can perform a task immediately, or ask for more information before proceeding. Activities are managed by an activity view, which appears as a sheet or popover, depending on the device and orientation. Use activities to give people access to custom services or tasks your app can perform.

The system provides a number of built-in activities, including Print, Twitter, Message, and Airplay. These tasks always appear first in activity views and can’t be reordered. You don’t need to create custom activities that perform these built-in tasks. Activity views also display share and action extensions from other apps. See Sharing and Actions.

Design simple template images to represent your custom activities. A template image uses a mask to create an icon. Use black and white with appropriate transparency and antialiasing, and don’t include a drop shadow. Template images should be centered in an area measuring about 70px × 70px.

Craft activity titles that succinctly describe your tasks. Titles appear below the icons in an activity view. Short titles work best. When a title is too long, iOS first shrinks the text and then—if the title is still too long—truncates it. In general, avoid including your company or product name in a title.

Make sure activities are appropriate for the current context. Although system-provided tasks can’t be reordered in an activity, they can be excluded if they aren’t applicable to your app. For example, to prevent people from printing images, you can exclude the Print activity. You can also identify which custom tasks to show at any given time.

Use the Action button to display an activity view. People are accustomed to accessing system-provided activities when they tap the Action button. Avoid confusing people by providing an alternative way to do the same thing.

For developer guidance, see UIActivityViewContoller and UIActivity.

Alerts

Alerts convey important information related to the state of your app or the device, and often request feedback. An alert consists of a title, an optional message, one or more buttons, and optional text fields for gathering input. Aside from these configurable elements, the visual appearance of an alert is static and can’t be customized.

Minimize alerts. Alerts disrupt the user experience and should only be used in important situations like confirming purchases and destructive actions (such as deletions), or notifying people about problems. The infrequency of alerts helps ensure that people take them seriously. Ensure that each alert offers critical information and useful choices.

Test the appearance of alerts in both orientations. An alert may appear differently in landscape mode and portrait mode. Optimize alert text so it reads well in any orientation without scrolling.

For developer guidance, see UIAlertController.

Alert Titles and Messages

Write short, descriptive, multiword alert titles. The less text people have to read onscreen, the better. Try to craft a title that avoids adding extra text as a message. Because single-word titles rarely provide useful information, consider asking a question or using short sentences. Whenever possible, keep titles to a single line. Use sentence-style capitalization and appropriate punctuation for complete sentences. Don’t use ending punctuation for sentence fragments.

If you must provide a message, write short, complete sentences. Try to keep messages short enough to fit on one or two lines to prevent scrolling. Use sentence-style capitalization and appropriate punctuation.

Avoid sounding accusatory, judgmental, or insulting. People know that alerts notify them about problems and dangerous situations. As long as you use a friendly tone, it’s better to be negative and direct than positive and oblique. Avoid pronouns such as you, your, me, and my, which are sometimes interpreted as insulting or patronizing.

Avoid explaining the alert buttons. If your alert text and button titles are clear, there should be no need to explain what the buttons do. In rare cases where you must provide guidance, use the word tap, preserve capitalization when referencing buttons, and don’t enclose button titles in quotes.

Alert Buttons

Generally, use two-button alerts. Two-button alerts provide an easy choice between two alternatives. Single-button alerts inform, but give no control over the situation. Alerts with three or more buttons create complexity and can require scrolling, which is a bad user experience. If you find that you need more than two choices, consider using an action sheet instead. See Action Sheets.

Give alert buttons succinct, logical titles. The best button titles consist of one or two words that describe the result of selecting the button. As with all button titles, use title-style capitalization and no ending punctuation. To the extent possible, use verbs and verb phrases that relate directly to the alert title and message—for example, View All, Reply, or Ignore. Use OK for simple acceptance. Avoid using Yes and No.

Place buttons where people expect them. In general, buttons people are most likely to tap should be on the right. Cancel buttons should always be on the left.

Label cancellation buttons appropriately. A button that cancels an alert’s action should always be labeled Cancel.

Identify destructive buttons. If an alert button results in a destructive action, such as deleting content, set the button’s style to Destructive so that it gets appropriate formatting by the system. For developer guidance, see the UIAlertActionStyleDestructive constant of UIAlertAction. Additionally, provide a Cancel button so people can safely opt out of the destructive action. Make the Cancel button bold by marking it as the default button.

Allow the Home button to cancel alerts. Pressing Home while an alert is visible exits the app. It should also produce the same effect as tapping the Cancel button—that is, the alert is dismissed without performing any action. If your alert doesn’t have a Cancel button, consider implementing a cancel action in your code that runs when the Home button is pressed.

Collections

A collection manages an ordered set of content, such as a set of photos, and presents it in a customizable and highly visual layout. Because a collection doesn’t enforce a strictly linear format, it’s particularly well-suited to displaying items that vary in size. Generally speaking, collections are ideal for showing off image-based content. Backgrounds and other decorative views can optionally be implemented to visually distinguish subsets of items.

Collections support both interactivity and animation. By default, you can tap to select, touch and hold to edit, and swipe to scroll. If your app requires it, more gestures can be added for performing custom actions. Within a collection, animations can be enabled whenever items are inserted, deleted, or reordered, and custom animations are also supported.

Avoid creating radical new designs when a standard row or grid layout is sufficient. A collection should enhance the user experience, not become the center of attention. Make it easy to select an item. If it’s hard to tap an item in your collection, people will get frustrated and lose interest before reaching the content they want. Use adequate padding around content to keep the layout clean and prevent overlapping of content.

Consider using a table instead of a collection for text. It’s generally simpler and more efficient to view and digest textual information when it’s displayed in a scrollable list.

Use caution when making dynamic layout changes. The layout of a collection can be changed at any time. If you dynamically change the layout while people are viewing and interacting with it, be sure the change makes sense and is easy to track. Unmotivated layout changes can make your app seem unpredictable and difficult to use. If context is lost due to a layout change, people are likely to feel like they’re no longer in control.

For developer guidance, see UICollectionView.

Image Views

An image view displays a single image or an animated sequence of images over a transparent or opaque background. Within an image view, images may be stretched, scaled, sized to fit, or pinned to a specific location. Image views are noninteractive by default.

If possible, ensure that all images within an image view are consistently sized. If your images are different sizes, the image view adjusts them separately. Using consistently sized images is more efficient than using images of varying size. It’s even more efficient to use images that are prescaled and don’t require any adjustments.

For developer guidance, see UIImageView.

NOTE

An image that’s been configured as a template image discards its color and adopts any tint that’s been applied to the enclosing image view. See UIImageRenderingModeAlwaysTemplate in UIImage.

Maps

A map view lets you display geographical data within your app and supports most of the functionality provided by the built-in Maps app. A map view can be configured to show a standard map, satellite imagery, or both. It can include pins and overlays, and supports zooming and panning. If your app supports routing, such as in a run-tracking app, you can use a map view to display the route.

In general, keep your map interactive. People are accustomed to using gestures to interact with the Maps app, and expect to be able to interact with your map in similar ways.

Use expected pin colors. A pin identifies a point of interest on your map. People are familiar with the standard pin colors in the Maps app. Avoid redefining the meaning of these colors in your app. Use red for a destination, green for a starting location, and purple for a user-specified location.

For developer guidance, see MapKit.

Pages

A page view controller provides a way to implement linear navigation between pages of content, such as in a document, book, notepad, or calendar. A page view controller uses one of two styles to manage transitions between pages during navigation—scrolling or page-curl. A scrolling transition has no specific appearance; pages fluidly scroll from one to the next. A page-curl transition causes pages to curl over as you swipe across the screen, turning like pages in a physical book.

If appropriate, implement a way to navigate nonlinearly. When a page view controller is used, pages flow sequentially and there’s no way to jump between nonadjoining pages. If people may need to access pages out of sequence in your app, implement a custom control that provides this functionality.

For developer guidance, see UIPageViewController.

Popovers

A popover is a transient view that appears above other content onscreen when you tap a control or in an area. Typically, a popover includes an arrow pointing to the location from which it emerged. Popovers can be nonmodal or modal. A nonmodal popover is dismissed by tapping another part of the screen or a button on the popover. A modal popover is dismissed by tapping a Cancel or other button on the popover.

Popovers are most appropriate on larger screens and can contain any variety of elements, including navigation bars, toolbars, tab bars, tables, collections, images, maps, and custom views. When a popover is visible, interactions with other views are normally disabled until the popover is dismissed. Use a popover to show options or information related to the content onscreen. Many iPad apps, for example, show a popover of sharing options when you tap the Share button.

Avoid displaying popovers on iPhones. Generally, popovers should be reserved for use in iPad apps. In iPhone apps, utilize all available screen space by presenting information in a fullscreen modal view, rather than in a popover. For related guidance, see Modality.

Use a Close button for confirmation and guidance only. A Close button, such as Cancel or Done, is worth including if it provides clarity, such as exiting with or without saving changes. In general, a popover should close automatically when its presence is no longer necessary. In most cases, a popover should close when someone taps outside of its bounds or selects an item in the popover. If multiple selections can be made, the popover should remain open until someone explicitly dismisses it or taps outside of its bounds.

Always save work when automatically closing a nonmodal popover. It’s easy to dismiss a nonmodal popover unintentionally by tapping another part of the screen. Discard work only when someone taps an explicit Cancel button.

Position popovers appropriately onscreen. A popover’s arrow should point as directly as possible to the element that revealed it. Because popovers can’t be dragged around the screen, a popover shouldn’t cover essential content people may need to see while using the popover. A popover also shouldn’t cover the element that was tapped to show the popover.

Show one popover at a time. Displaying multiple popovers clutters the interface and causes confusion. Never show a cascade or hierarchy of popovers, in which one emerges from another. If you need to show a new popover, close the open one first.

Don’t show another view over a popover. Except for an alert, nothing should be displayed on top of a popover.

When possible, let users close one popover and open another with a single tap. Avoiding extra taps is especially desirable when several different bar buttons each open a popover.

Avoid making a popover too big. A popover shouldn’t take over the entire screen. Make it only big enough to display its contents and point to the place it came from. Be aware that the system might adjust the size of a popover to ensure it fits well onscreen.

Make sure custom popovers look like popovers. Although you can customize many of the visual aspects of a popover, avoid creating a design people might not recognize as a popover. Popovers tend to work best when they contain standard controls and views.

Provide a smooth transition when changing the size of a popover. Some popovers provide both condensed and expanded views of the same information. If you adjust the size of a popover, animate the change to avoid giving the impression that a new popover replaced the old one.

For developer guidance, see UIPopoverPresentationController.

Scroll Views

A scroll view allows users to browse content, such as text in a document or a collection of images, that’s larger than the visible area. As people swipe, flick, drag, tap, and pinch, a scroll view follows the gesture, revealing or zooming content in a way that feels natural. A scroll view itself has no appearance, but does display transient scrolling indicators as people interact with it. A scroll view can also be configured to operate in paging mode, where scrolling reveals an entirely new page of content rather than moving around the current page.

Support zoom behavior appropriately. If it makes sense in your app, let people pinch or double-tap to zoom in and out. When you enable zoom, set maximum and minimum scale values that make sense. For example, zooming in on text until a single character fills the screen probably doesn’t make sense in most apps.

Consider showing a page control element when a scroll view is in paging mode. A page control shows how many pages, screens, or other chunks of content are available and indicates which one is currently visible. If you show a page control with a scroll view, disable the scrolling indicator on the same axis to avoid confusion. For additional guidance, see Page Controls.

Don’t place a scroll view inside of another scroll view. Doing so creates an unpredictable interface that’s difficult to control.

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

For developer guidance, see UIScrollView.

Split Views

A split view manages the presentation of two side-by-side panes of content, with persistent content in the primary pane and related information in the secondary pane. Each pane can contain any variety of elements, including navigation bars, toolbars, tab bars, tables, collections, images, maps, and custom views. Split views are often used with filterable content; a list of filter categories appears in the primary pane, and the filtered results for the selected category are shown in the secondary pane. If your app requires it, the primary pane can overlay the secondary pane and can be hidden offscreen when not in use. This is particularly useful when the device is in portrait orientation, as it allows more room for viewing content in the secondary pane.

Choose a split-view layout that works well with your content. By default, a split view devotes a third of the screen to the primary pane and two-thirds to the secondary pane. The screen can also be split into halves. Choose an appropriate split based on your content, and make sure the panes don’t look unbalanced. Avoid creating a secondary pane that’s narrower than the primary pane.

Persistently highlight the active selection in the primary pane. Although the secondary pane’s content can change, it should always correspond to a clearly identifiable selection in the primary pane. This helps people understand the relationship between the panes.

In general, restrict navigation to one side of a split view. Placing navigation in both panes of a split view makes it hard for people to stay oriented and discern the relationship between the two panes.

Provide multiple ways to access a hidden primary pane. On layouts where the primary pane may be offscreen, be sure to provide a button—typically in a navigation bar—to reveal the pane. Unless your app uses the swipe gesture to perform other functions, let people swipe from the side of the screen to access the primary pane too.

For developer guidance, see UISplitViewController.

Tables

A table presents data as a scrolling, single-column list of rows that can be divided into sections or groups. Use a table to display large or small amounts of information cleanly and efficiently in the form of a list. Generally speaking, tables are ideal for text-based content, and often appear as a means of navigation on one side of a split view, with related content shown on the opposite side. See Split Views.

iOS implements two styles of table, plain and grouped.

Plain. Rows can be separated into labeled sections, and an optional index can appear vertically along the right edge of the table. A header can appear before the first item in a section, and a footer can appear after the last item.

Grouped. Rows are displayed in groups, which can be preceded by a header and followed by a footer. This style of table always contains at least one group and each group always contains at least one row. A grouped table doesn’t include an index.

Think about table width. Thin tables can cause truncation and wrapping, making them hard to read and scan quickly at a distance. Wide tables can also be difficult to read and scan, and can take away space from content.

Begin showing table content quickly. Don’t wait for extensive table content to load before showing something. Fill onscreen rows with textual data immediately and show more complex data—such as images—as it becomes available. This technique gives people useful information right away and increases the perceived responsiveness of your app. In some cases, showing stale, older data may make sense until fresh, new data arrives.

Communicate progress as content loads. If a table’s data takes time to load, show a progress bar or spinning activity indicator to reassure people that your app is still running.

Keep content fresh. Consider updating your table’s content regularly to reflect newer data. Just don’t change the scrolling position. Instead, add the content to the beginning or end of the table, and let people scroll to it when they’re ready. Some apps display an indicator when new data has been added, and provide a control for jumping right to it. It’s also a good idea to include a refresh control, so people can manually perform an update at any time. See Refresh Content Controls.

Avoid combining an index with table rows containing right-aligned elements. An index is controlled by performing large swiping gestures. If other interactive elements reside nearby, such as disclosure indicators, it may be difficult to discern the user’s intent when a gesture occurs and the wrong element may be activated.

For developer guidance, see UITableView.

Table Rows

You use standard table cell styles to define how content appears in table rows.

Default. An optional image on the left side of the row, followed by a left-aligned title. It’s a good option for displaying items that don’t require supplementary information. For developer guidance, see the UITableViewCellStyleDefault constant of UITableViewCell.

Subtitle. A left-aligned title on one line and a left-aligned subtitle on the next. This style works well in a table where rows are visually similar. The additional subtitle helps distinguish rows from one another. For developer guidance, see the UITableViewCellStyleSubtitle constant of UITableViewCell.

Value 1. A left-aligned title with a right-aligned subtitle in a lighter font on the same line. For developer guidance, see the UITableViewCellStyleValue1 constant of UITableViewCell.

Value 2. A right-aligned title, followed by a left-aligned subtitle in a lighter font on the same line. For developer guidance, see UITableViewCellStyleValue2 constant of UITableViewCell.

All standard table cell styles also allow graphical elements, such as a checkmark or disclosure indicator. Of course, adding these elements decreases the space available for titles and subtitles.

Keep text succinct to avoid clipping. Truncated words and phrases are hard to scan and decipher. 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 where truncation occurs.

Consider using a custom title for a Delete button. If a row supports deletion and it helps provide clarity, replace the system-provided Delete title with a custom title.

Provide feedback when a selection is made. People expect a row to highlight briefly when its content is tapped. Then, people expect a new view to appear or something to change, such as a checkmark appearing, that indicates a selection has been made.

Design a custom table cell style for nonstandard table rows. Standard styles are great for use in a variety of common scenarios, but some content or your overall app design may call for a heavily customized table appearance. To learn how to create your own cells, see Customizing Cells in Table View Programming Guide for iOS.

For developer guidance, see UITableViewCell.

Text Views

A text view displays multiline, styled text content. Text views can be any height and enable scrolling when the content extends outside of the view. By default, content within a text view is left-aligned and uses the system font in black. If a text view is editable, a keyboard appears when you tap inside the view.

Keep text legible. Although you can use multiple fonts, colors, and alignments in creative ways, it’s essential to maintain the readability of your content. It’s a good idea to adopt Dynamic Type so your text still looks good if people change text size on their device. You should also test your content with accessibility options enabled, such as bold text.

Show the appropriate keyboard type. iOS provides several different keyboard types, each designed to facilitate a different type of input. To streamline data entry, the keyboard displayed during the editing of a text view should be appropriate for the type of content in the field. For a complete list of available keyboard types, see the UIKeyboardType constant of UITextInputTraits.

For developer guidance, see UITextView.

Web Views

A web view loads and displays rich web content, such as embedded HTML and websites, directly within your app. Mail uses a web view, for example, to show HTML content in messages.

Enable forward and back navigation when appropriate. Web views support forward and back navigation, but this behavior is disabled by default. If people will use your web view to visit multiple pages, enable forward and back navigation, and provide corresponding controls to initiate these features.

Avoid using a web view to build a web browser. Using a web view to let people briefly access a website without leaving the context of your app is fine, but Safari is the primary way people browse the web on iOS. Attempting to replicate the functionality of Safari in your app is unnecessary and discouraged.

For developer guidance, see WKWebView.

Buttons

Buttons initiate app-specific actions, have customizable backgrounds, and can include a title or an icon. The system provides a number of predefined button styles for most use cases. You can also design fully custom buttons.

For developer guidance, see UIButton.

System Buttons

System buttons often appear in navigation bars and toolbars, but may be used anywhere.

Use verbs in titles. An action-specific title shows that a button is interactive and says what happens when you tap it.

Use title-case for titles. Capitalize every word except articles, coordinating conjunctions, and prepositions of four or fewer letters.

Keep titles short. Overly long text can crowd your interface and may get truncated on smaller screens.

Consider adding a border or a background only when necessary. By default, a system button has no border or background. In some content areas, however, a border or background is necessary to denote interactivity. In the Phone app, bordered number keys reinforce the traditional model of making a call, and the background of the Call button provides an eye-catching target that’s easy to hit.

For developer guidance, see the UIButtonTypeSystem button type of UIButton.

Detail Disclosure Buttons

A Detail Disclosure button opens a view—typically, a modal view—containing additional information or functionality related to a specific item onscreen. Although you can use them in any type of view, Detail Disclosure buttons are commonly used in tables to access information about specific rows.

Use Detail Disclosure buttons appropriately in tables. When a Detail Disclosure button is present in a table row, tapping the button shows additional information. Tapping elsewhere selects the row or results in app-defined behavior. If you want people to tap the entire row to see additional detail, don’t use a Detail Disclosure button. Instead, use a detail disclosure accessory control, which appears as a chevron. See UITableViewCellAccessoryType in UITableViewCell.

For developer guidance, see the UIButtonTypeDetailDisclosure button type of UIButton.

Info Buttons

An Info button reveals configuration details about an app, sometimes on the back of the current view, after flipping the view around. Info buttons come in two styles—light and dark. Pick the style that best coordinates with your app’s design and doesn’t get lost onscreen.

For developer guidance, see the UIButtonTypeInfoLight and UIButtonTypeInfoDark button types of UIButton.

Add Contact Buttons

Users can tap an Add Contact button to browse a list of existing contacts and to select one for insertion into a text field or other view. In Mail, for example, you can tap the Add Contact button in the To field of a message to select a recipient from your list of contacts.

Allow keyboard entry in addition to the Add Contact button. The Add Contact button provides an alternative, not a replacement, to typing contact information. It’s fine to provide it as a shortcut for adding existing contacts, but let people enter contact information with the keyboard too.

For developer guidance, see the UIButtonTypeContactAdd button type of UIButton.

Edit Menus

People can touch and hold or double-tap an element in a text field, a text view, a web view, or an image view to select content and reveal edit options, such as Copy and Paste.

Show appropriate commands for the current context. By default, the options include Cut, Copy, Paste, Select, Select All, and Delete commands, any of which can optionally be disabled. If nothing is selected, the menu shouldn’t show options that require a selection, such as Copy or Cut. Similarly, the menu shouldn’t have a Select option if something is already selected.

Let people use the standard gestures to access edit options. People expect to reveal the menu by touching and holding or double-tapping content, such as text or a photo. Your app should respond to both gestures. You can define what becomes selected when someone double-taps. In a text view, selection by word should be the default.

Adjust placement of edit options, if necessary. By default, the menu is positioned above or below the insertion point or selection, depending on available space, and includes a pointer to the related content. Although you can’t change the shape of the menu, its position is configurable—you can prevent it from covering important content or parts of your interface.

Don’t implement other controls with the same functionality as the edit menu. Providing multiple ways to initiate an operation results in an inconsistent user experience and leads to confusion. If your app lets people use the menu to copy content, for example, don’t implement a copy button too.

Allow potentially useful noneditable text to be selected and copied. People often want to add static content, such as an image label or social media status, to an email, a note, or a web search.

Don’t add edit options to a button. If you do this, people attempting to reveal the options will end up activating the button instead.

Make edit operations undoable. The menu doesn’t require confirmation before its actions are performed. Because someone could change their mind after performing an operation, always implement undo and redo support.

Expand edit options with useful custom commands. You can add value by providing additional app-specific commands. Like the standard commands, any custom commands should operate on selected text or objects.

Show custom commands after the system-provided ones. Don’t intersperse custom commands with the system-provided ones, which are well known and frequently used.

Minimize the number of custom commands. Don’t overwhelm people with too many choices.

Keep custom command names short. Command names should be verbs or short verb phrases that succinctly describe the action to be performed. Use title-style capitalization—capitalize every word except articles, coordinating conjunctions, and prepositions of four or fewer letters.

For developer guidance, see Copy, Cut, and Paste Operations in Text Programming Guide for iOS and UIMenuController.

Labels

A label describes an onscreen interface element or provides a short message. Although people can’t edit labels, they can sometimes copy the content of labels. Labels can display any amount of static text, but are best kept short.

Keep labels legible. Labels can include plain or styled text. If you adjust the style of a label or use custom fonts, be sure to not sacrifice legibility. It’s best to adopt Dynamic Type so your labels still look good when a user changes text size on their device. See Dynamic Type. You should also test your labels with accessibility options enabled, such as bold text. See Accessibility.

To learn about text, see String Programming Guide. To learn about creating styling text, see Attributed String Programming Guide. For label developer guidance, see UILabel.

Page Controls

A page control shows the position of the current page in a flat list of pages. It appears as a series of small indicator dots, representing the available pages in the order they were opened. A solid dot denotes the current page. Visually, these dots are always equidistant, and are clipped if too many appear onscreen. Users can tap the leading or trailing edge of a page control to visit the next or previous page, but they can't tap a specific dot to go to a specific page. Navigation always occurs sequentially, usually by swiping the page to one side.

Don’t use a page control with hierarchical pages. A page control doesn’t show how pages are related or indicate which page corresponds to each dot. This type of control is designed for pages that are peers to one another.

Don’t display too many pages. More than about 10 dots are hard to count at a glance, and more than about 20 open pages are time-consuming to visit in sequence. If your app needs to display more than 20 pages as peers, consider using a different arrangement—such as a grid—that enables nonsequential navigation.

Center page controls at the bottom of the screen. A page control should always be centered and positioned between the bottom of the content and the bottom of the screen. This keeps it visible, without blocking content.

For developer guidance, see UIPageControl.

Pickers

A picker includes one or more scrollable lists of distinct values, each of which has a single selected value—appearing in darker text in the center of the view. A picker is often displayed at the bottom of the screen or in a popover when the user is editing a field or tapping a menu. Pickers can also appear inline, such as while editing a date in a Calendar event. The height of a picker is roughly the height of five rows of list values. The width of a picker is either the width of the screen or its enclosing view, depending on the device and context.

Use predictable and logically ordered values. Many values in a picker may be hidden when the scrollable lists are stationary. It's best when people can predict what these values are, such as with a list of alphabetized countries, so they can move through the lists quickly.

Avoid switching screens to show a picker. A picker works well when displayed in context, below or in close proximity to the field being edited.

Use a table instead of a picker for large value lists. Long lists can be tedious to navigate in a picker. A table has adjustable height and can include an index, making scrolling much faster.

For developer guidance, see UIPickerView.

Date Pickers

A date picker is an efficient interface for selecting a specific date, time, or both. It also provides an interface for displaying a countdown timer.

A date picker has four modes, each of which presents a different set of selectable values.

Date. Displays months, days of the month, and years.

Time. Displays hours, minutes, and (optionally) an AM/PM designation.

Date and time. Displays dates, hours, minutes, and (optionally) an AM/PM designation.

Countdown timer. Displays hours and minutes, up to a maximum of 23 hours and 59 minutes.

The exact values shown in a date picker and their order depend upon the user’s locale.

Consider providing less granularity when specifying minutes. By default, a minute list includes 60 values (0 to 59). You can optionally increase the minute interval as long as it divides evenly into 60. For example, you might want quarter-hour intervals (0, 15, 30, and 45).

For developer guidance, see UIDatePicker.

Progress Indicators

Don’t make people sit around staring at a static screen waiting for your app to load content or perform lengthy data processing operations. Use activity indicators and progress bars to let people know your app isn’t stalled and to give them some idea of how long they’ll be waiting.

See also Loading.

Activity Indicators

An activity indicator spins while an unquantifiable task, such as loading or synchronizing complex data, is performed. It disappears when the task completes. Activity indicators are noninteractive.

Favor progress bars over activity indicators. If activity is quantifiable, use a progress bar instead of an activity indicator so people can better gauge what’s happening and how long it will take.

Keep activity indicators moving. People associate a stationary activity indicator with a stalled process. Keep it spinning so they know something’s happening.

If it’s helpful, provide useful information while waiting for a task to complete. Include a label above an activity indicator to give extra context. Avoid vague terms like loading or authenticating because they don’t usually add any value.

For developer guidance, see UIActivityIndicatorView.

Progress Bars

A progress bar includes a track that fills from left to right to show the progression of a task with a known duration. Progress bars are noninteractive, although they are often accompanied by a button for canceling the corresponding operation.

Always report progress accurately. Don’t display inaccurate progress information just to make your app appear busy. Only use progress bars for tasks that are quantifiable. Otherwise, use an activity indicator.

Use progress bars for tasks with a well-defined duration. Progress bars are great for showing the status of a task, especially when it helps convey how much longer the task needs to complete.

Hide the unfilled portion of track in navigation bars and toolbars. By default, a progress bar’s track includes both filled and unfilled portions. When used in a navigation bar or toolbar, such as to denote a page loading, a progress bar should be configured to hide the unfilled portion of the track.

Consider customizing a progress bar’s appearance to match your app. A progress bar’s appearance can be adjusted to match your app’s design. You can specify, for example, a custom tint or image for both the track and fill.

For developer guidance, see UIProgressView.

Network Activity Indicators

The network activity indicator spins in the status bar at the top of the screen as networking occurs. It disappears when networking is complete. This indicator looks just like an activity indicator and is noninteractive.

Show this indicator only for network operations lasting more than a few seconds. Don’t display the indicator for quick network operations because it’s likely to disappear before anyone notices its presence or realizes what it’s meant to communicate.

See also Status Bars. For developer guidance, see the networkActivityIndicatorVisible method of UIApplication.

Refresh Content Controls

A refresh control is manually initiated to immediately reload content, typically in a table view, without waiting for the next automatic content update to occur. A refresh control is a specialized type of activity indicator, is hidden by default, and becomes visible when dragging down on the view to be reloaded. In Mail, for example, you can drag the list of Inbox messages down to check for new messages.

Perform automatic content updates. Although people appreciate being able to trigger an immediate content refresh, they also expect automatic refreshes to occur periodically. Don’t make users responsible for initiating every update. Keep data fresh by updating it regularly.

Supply a short title only if it adds value. Optionally, a refresh control can include a title. In most cases, this is unnecessary, as the animation of the control indicates that content is loading. If you do include a title, don’t use it to explain how to perform a refresh. Instead, provide information of value about the content being refreshed. A refresh control in Podcasts, for example, uses a title to tell people when the last podcast update occurred.

For developer guidance, see UIRefreshControl.

Segmented Controls

A segmented control is a linear set of two or more segments, each of which functions as a mutually exclusive button. Within the control, all segments are equal in width. Like buttons, segments can contain text or images. Segmented controls are often used to display different views. In Maps, for example, a segmented control lets you switch between Map, Transit, and Satellite views.

Limit the number of segments to improve usability. Wider segments are easier to tap. On iPhone, a segmented control should have five or fewer segments.

Try to keep segment content size consistent. Because all segments have equal width, it doesn’t look great if content fills some segments but not others.

Avoid mixing text and images in a segmented control. Although individual segments can contain text or images, mixing the two in a single control can lead to a disconnected and confusing interface.

Position content appropriately in a custom a segmented control. If you change the background appearance of a segmented control, make sure content still looks good and doesn’t appear misaligned.

For developer guidance, see UISegmentedControl.

Sliders

A slider is a horizontal track with a control called a thumb, which you can slide with your finger to move between a minimum and maximum value, such as screen brightness level or position during media playback. As a slider’s value changes, the portion of track between the minimum value and the thumb fills with color. A slider can optionally display left and right icons that illustrate the meaning of the minimum and maximum values.

Customize a slider’s appearance if it adds value. A slider’s appearance, including track color, thumb image, and left and right icons, can be adjusted to blend with your app’s design and to communicate intent. A slider that adjusts image size, for example, could show a small image icon on the left and a large image icon on the right.

Don’t use a slider to adjust audio volume. If you need to provide volume control in your app, use a volume view, which is customizable and includes a volume-level slider and a control for changing the active audio output device. To learn about implementing a volume view, see MPVolumeView.

For developer guidance, see UISlider.

Steppers

A stepper is a two-segment control used to increase or decrease an incremental value. By default, one segment of a stepper displays a plus symbol and the other displays a minus symbol. These symbols can be replaced with custom images, if desired.

Make the value affected by the stepper obvious. A stepper itself doesn’t display any values, so make sure people know which value they’re changing when they use a stepper.

Don’t use a stepper when large value changes are likely. Steppers work well for making small changes that require a few taps. On a printing screen, for example, it makes sense to use a stepper to set the number of copies because people rarely change this setting by much. On the other hand, it doesn’t make sense to use a stepper to choose a page range because even a reasonable page range would require lots of taps.

For developer guidance, see UIStepper.

Switches

A switch is a visual toggle between two mutually exclusive states—on and off.

Consider adjusting a switch’s appearance to match the style of your app. If it works well in your app, change the colors of a switch in its on and off states or use custom imagery to represent the switch.

Use switches in table rows only. Switches are intended for use in tables, such as in a list of settings that can be toggled on and off. If you need similar functionality in a toolbar or navigation bar, use a button instead, and provide two distinct icons that communicate the states.

Avoid adding labels to describe the values of a switch. Switches are either on or off. Providing labels that describe these states is redundant and clutters the interface.

Consider using switches to manage the availability of related interface elements. Switches often affect other content onscreen. Enabling the Airplane Mode switch in Settings, for example, disables certain other settings, such as Cellular and Personal Hotspot. Disabling the Wi-Fi switch in Settings > Wi-Fi causes available networks and other options to disappear.

For developer guidance, see UISwitch.

Text Fields

A text field is a single-line, fixed-height field, often with rounded corners, that automatically brings up a keyboard when the user taps it. Use a text field to request a small amount of information, such as an email address.

Show a hint in a text field to help communicate purpose. A text field can contain placeholder text—such as "Email" or "Password"—when there’s no other text in the field. Don’t use a separate label to describe a text field when placeholder text is sufficient.

Use secure text fields when appropriate. Always use a secure text field when your app asks for sensitive data, such as a password.

TIP

For multiline or multistyle text entry, use a text view instead. See Text Views.

Show the appropriate keyboard type. iOS provides several different keyboard types, each designed to facilitate a different type of input. To streamline data entry, the keyboard displayed when editing a text field should be appropriate for the type of content in the field. If your app asks for an email address, for example, it should display the email address keyboard. For a complete list of available keyboard types, see the UIKeyboardType constant of UITextInputTraits.

Use images and buttons to provide clarity and functionality in text fields. You can display custom images in the left or right sides of a text field, or you can add a system-provided button, such as the Bookmarks button. In general, use the left end of a text field to indicate a field’s purpose and the right end to indicate the presence of additional features, such as bookmarking.

Display a 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, eliminating the need to keep tapping the Delete key.

For developer guidance, see UITextField.

Custom Keyboards

A keyboard extension replaces the standard keyboard with a custom keyboard. Custom keyboards are enabled in the Settings app, under General > Keyboards. Once enabled, the keyboard is available during text entry within any app, except when editing secure text fields and phone number fields. People can enable multiple custom keyboards, and switch between them at any time.

Make sure you really need a custom keyboard. Custom keyboards make sense when you want to expose unique keyboard functionality systemwide, such as a novel way of inputting text or the ability to type in a language not supported by iOS. If you only want a custom keyboard in your app, consider creating a custom input view instead. See Custom Input Views.

Provide an obvious and easy way to switch between keyboards. People know that the Globe key on the standard iOS keyboard, which replaces the Emoji key when you have multiple keyboards enabled, quickly switches to other keyboards. They expect a similarly intuitive experience in your keyboard. Note that the Globe key replaces the Emoji key when you have multiple keyboards installed.

Consider providing a keyboard tutorial in your app. People are used to the standard keyboard, and learning a new keyboard takes time. Make the onboarding process easier by providing usage instructions in your app—not in the keyboard itself. Tell people how to enable your keyboard, activate it during text entry, use it, and switch back to the standard keyboard.

For developer guidance, see Custom Keyboard in App Extension Programming Guide.

Custom Input Views

A custom input view replaces the standard keyboard with a custom keyboard, but only in your app, rather than systemwide. Use a custom input view to provide a unique and efficient method of data entry. Numbers, for example, implements custom input views for entering amounts, dates, and other values while editing a spreadsheet.

Make functionality obvious. The controls on a custom input view should make sense in the context of your app. Data entry should be clear and intuitive, so additional instruction isn’t necessary.

Play the standard keyboard click sound during typing. The keyboard click sound provides audible feedback while the user is tapping buttons on the keyboard. Tapping custom controls in your input view should produce this sound too. Note that this sound is only available to visible custom input views, and people can disable the sound systemwide in Settings > Sounds. For developer guidance, see the playInputClick method of UIDevice.

Provide an input accessory view if necessary. Some apps implement an additional custom input accessory view, which appears above the keyboard. In Numbers, an input accessory view helps people enter standard or custom calculations.

For developer guidance, see Custom Data Input Views in Text Programming Guide for iOS.

Document Providers

A document provider extension implements a custom interface for importing, exporting, opening, or moving an app’s documents from within other apps on the system. When a document provider extension loads, its interface displays in a modal view containing a navigation bar.

When users open or import files, show only context-specific documents and information. When someone uses your extension to open or import documents, only display documents that are appropriate for the current context. For example, if a PDF-editing app loads your extension, only list PDF files as possible documents that can be opened or imported. Be sure to list other information that may be helpful too, such as modification dates, sizes, and whether documents are local or remote.

Let people select a destination when exporting and moving documents. Unless your app stores documents in a single directory, let users navigate to a specific destination in your directory hierarchy. Consider providing a way to add new subdirectories.

Don’t provide a custom navigation bar. Your extension loads within a modal view that already includes a navigation bar. Providing a second navigation bar is confusing and takes space away from your content.

For developer guidance, see Document Provider in App Extension Programming Guide.

Home Screen Quick Actions

Home screen quick actions are a convenient way to perform useful, app-specific actions right from the Home screen, using 3D Touch. Apply a little pressure to an app icon with your finger—more than you use for tap and hold—to see a list of available quick actions. Tap one to activate it. Mail, for example, includes quick actions that take you right to the Inbox or the VIP mailbox, initiate search, and create a new message. Each Home screen quick action includes a title, an icon on the left or right (depending on your app’s position on the home screen), and an optional subtitle. The title and subtitle are always left-aligned in left-to-right languages. Your app can even dynamically update its quick actions when new information is available. For example, Messages provides quick actions for opening your most recent conversations.

TIP

Some apps display a widget above their list of quick actions. For example, the Stocks quick action list appears below a widget showing your current stock values. Consider implementing a widget to provide an even more engaging experience. For widget design guidelines, see Home Screen Quick Action Widgets.

Create quick actions for compelling, high-value tasks. For example, Maps lets people search near their current location or get directions home without first opening the Maps app. Every app should enable at least one useful quick action; you can provide a total of four.

Avoid using quick actions to ease navigation. If it’s difficult or time-consuming to visit important areas in your app, first fix your navigation to work well for everyone. Next, focus on providing quick actions that enable useful, creative tasks.

Avoid making unpredictable changes to quick actions. Dynamic quick actions are a great way to keep actions relevant. For example, it may make sense to update quick actions based on the current location or recent activities in your app, time of day, or changes in settings. However, actions shouldn’t change in ways that are unpredictable or confusing.

Provide a succinct title for each quick action. An action’s title should instantly communicate the results of the action; for example, “Directions Home,” “Create New Contact,” and “New Message.” If you need to give more context, provide a subtitle too. Mail uses subtitles to indicate whether there are unread messages in the inbox and VIP folder. Don’t include your app name or any extraneous information in the title or subtitle, keep the text short to avoid truncation, and take localization into account as you write the text.

Don’t use quick actions for notification. People expect to receive notifications from apps in other ways. See Notifications.

Provide a recognizable icon for each quick action. Whenever possible, use the familiar system icons. See Quick Action Icons. If you design your own icons, download the Quick Action Icon Template and follow the guidelines in Custom Icons.

Don't use an emoji in place of an icon. Emojis don't align properly with right-aligned text. Also, emojis are full color, whereas quick action icons are monochromatic.

Messaging

iMessage apps and Sticker packs are messaging extensions that let people share content with friends, communicate expressively, and share collaborative experiences.

iMessage Apps

An iMessage app provides an interactive experience within the context of a Messages conversation. iMessage apps enable people to share data and media, collaborate on shared tasks, and play games with one another.

Design an intuitive interface. If your iMessage app provides static content, such as text, photos, or videos, make sure it’s easy for people to browse and select items for insertion into the conversation. If your app is interactive, such as a multiplayer game board or collaborative takeout order, make sure its functionality is useful and understandable.

Provide focused content. For clarity and ease of use, each messaging extension should have a single focus. Don’t try to design one app that combines both stickers and ridesharing, for example.

Provide a fun, collaborative experience. iMessage apps are typically used in the context of fast-paced, informal conversations between two or more people. Creatively leverage this environment to encourage back-and-forth participation through sharing, editing, or augmenting content. If all parties are using the app together, content updates occur in real time.

Highlight interesting iOS app content. Extend the functionality of your iOS app by surfacing its content in the Messages space. Think about what kinds of information people might want to share and how they can interact with your app within the context of an active conversation.

Inset content to avoid cropping. Your app's content is presented in message bubbles with rounded corners, so don't put essential information in the corners.

Differentiate compact and expanded views. Your app appears in a compact view below a conversation. This view should visually distinguish your app from other apps and provide access to frequently used features. People can also open your app in an expanded view to access advanced features or see more options at once. Note that horizontal scrolling isn’t permitted in the compact view.

Allow text editing in the expanded view only. The compact view is roughly the same height as the keyboard. To ensure that users can see what they’re editing, allow text entry only in the expanded view.

Stickers

Stickers offer a fun, engaging way for people to express themselves in a Messages conversation without typing or using emojis. A sticker is an image or an animation that can be sent or placed on messages, photos, and other stickers to add emphasis and communicate emotion.

Design for expressiveness. People use stickers to visually convey moods and reactions. Strive to deliver stickers that connect with people at an emotional level. Consider incorporating imagery, words, and phrases to add new dimensions to conversations.

Think globally. Messaging is a universal form of communication. Aim for stickers that have broad, international appeal.

Use descriptive image names or provide alternative text labels. Although they aren’t visible onscreen, image names and alternative text labels let VoiceOver audibly describe stickers, making navigation easier for people with visual impairments.

Add vitality through animation. Although stickers can be static images, animated stickers are a great way to impart energy in a conversation. Be sure to use a frame rate high enough to keep motion fluid.

Test placement possibilities. Users can scale, rotate, and place stickers over parts of a conversation. Make sure your stickers are legible against different colors and imagery, regardless of their position or size.

Consider using vibrant colors and transparency. Vivid colors add richness and excitement to a conversation. Transparency allows meaningful placement of stickers onto messages, photos and other stickers.

Sticker Sizes

Messages supports three sticker sizes, which are displayed in a grid-based browser. Pick a size that works best for your content and prepare all of your stickers at that size.

Scale stickers appropriately. Although Messages scales larger stickers down if necessary, it’s best to provide prescaled stickers for optimal quality and performance.

Be mindful of file size restrictions. For efficiency, each individual sticker may not exceed 500KB. Note that Xcode saves PNG animations using a 24-bit palette, which may lead to larger than expected file sizes.

Sticker Formats

Messages supports stickers in the following file formats:

Format

Recommended

Supports transparency

Supports animation

PNG

8-bit transparency

APNG

8-bit transparency

GIF

single color transparency

JPEG

iMessage App and Sticker Pack Icons

Like iOS apps, iMessage apps and sticker packs need recognizable app icons.

Keep the background simple and provide a single focus point. Design an icon with a single, centered point that captures attention at a glance. Include a simple background that doesn’t overpower other icons.

Keep icon corners square. The system applies a mask that rounds icon corners automatically.

Provide different sized icons. Your icon appears in the App Store, Messages, notifications, and Settings. To ensure that your icon looks great in any context and on any device, you can provide variations of your icon in the following sizes:

@2x @3x

iMessage app icons 148px × 110px —

134px × 100px —

120px × 90px 180px × 135px

64px × 48px 96px × 72px

54px × 40px 81px × 60px

Settings icons 58px × 58px 87px × 87px

App Store (prior to iOS 10) 1024px × 1024px

For additional guidance on designing great app icons, much of which can be applied to iMessage app and sticker pack icons, see App Icon.

To download an iMessage app and sticker pack icon Photoshop template, see Resources.

For developer guidance, see The Messages Framework.

Photo Editing

Photo editing extensions let people modify photos and videos within the Photos app by applying filters or making other changes. Edits are always saved in the Photos app as new files, safely preserving the original versions.

To access a photo editing extension, a photo must be in edit mode. While in edit mode, tapping the extension icon in the toolbar displays an action menu of available editing extensions. Selecting one displays the extension’s interface in a modal view containing a navigation bar. Dismissing this view confirms and saves the edit, or cancels it and returns to the Photos app.

Confirm cancellation of edits. Editing a photo or video can be time consuming. If someone taps the Cancel button, don’t immediately discard their changes. Ask them to confirm that they really want to cancel, and inform them that any edits will be lost after cancellation. There’s no need to show this confirmation if no edits have been made yet.

Don’t provide a custom navigation bar. Your extension loads within a modal view that already includes a navigation bar. Providing a second navigation bar is confusing and takes space away from the content being edited.

Let people preview edits. It’s hard to approve an edit if you can’t see what it looks like. Let people see the result of their work before closing your extension and returning to the Photos app.

Use your app icon for your photo editing extension icon. This instills confidence that the extension is in fact provided by your app.

For developer guidance, see Photo Editing in App Extension Programming Guide.

Sharing and Actions

Share extensions provide a convenient way to share information from the current context with apps, social media accounts, and other services. Action extensions let people initiate content-specific tasks, such as adding a bookmark, copying a link, or saving an image. People access share extensions and action extensions by tapping an Action button in an app to display an activity view. An activity view shows only extensions relevant to the current context. You wouldn’t see text manipulation actions while editing a video, for example. Within an activity view, share extensions are listed above action extensions.

Enable a single, focused task. An extension isn’t a mini-app. It performs a narrowly scoped task related to the current context.

Craft a familiar interface. For share extensions, the system-provided composition view is familiar and provides a consistent sharing experience throughout the system. Use it whenever possible. For action extensions, include your app name, or design an interface that’s recognizable and feels like a natural extension of your app.

Streamline and limit interaction. The best extensions let people perform a task in just a few steps. For example, a share extension might immediately post an image to a social media account with a single tap. Only provide an interface when necessary.

Avoid placing modal views above your extension. Extensions are displayed within a modal view by default. While an alert might make sense above an extension, avoid layering additional modal views.

Use your main app to denote the progress of lengthy operations. An activity view should dismiss immediately after initiating sharing or an action. Time-consuming tasks should continue in the background, and your main app should provide some way to check the status of these tasks. Don’t use notifications for this. People don’t want to see a notification every time a task completes, although it’s fine to notify them if there’s a problem.

Use a template image for an action extension icon. A template image uses a mask to create an icon. Use black and white with appropriate transparency and antialiasing, and don’t include a drop shadow. Template images should be centered in an area measuring about 70px × 70px.

TIP

Share extensions automatically use your app icon, instilling confidence that the extension is in fact provided by your app.

For additional guidelines, see Activity Views. For developer guidance, see Share and Action in App Extension Programming Guide.

Widgets

A widget is an extension that displays a small amount of timely, useful information or app-specific functionality. For example, the News widget shows top headlines. Calendar provides two widgets, one that shows today’s events and one that shows what’s up next. Notes lets you preview recent notes and quickly create new notes, reminders, photos, and drawings. Widgets are highly customizable and can contain buttons, text, layout customizations, images, and more.

Widgets appear above the quick action list when you apply pressure to an app icon on the Home screen using 3D Touch. People also add the widgets they care about to the Search screen, which is accessed by swiping to the right on the Home screen and the Lock screen. Your goal should be to design a widget that people want to add to the Search screen.

Design a great glanceable experience. People use widgets to get brief updates and perform very simple tasks, so it’s essential to deliver the right amount of information and interactivity. Wherever possible, provide tasks that can be completed in a single tap. Panning and scrolling within widgets isn’t supported.

Show content quickly. People spend very little time looking at widgets and shouldn’t need to wait for content to load. Cache information locally so you can always show recent information while getting updates.

Provide ample margins and padding. Avoid extending content to the edges of a widget. In general, provide a margin of at least a few pixels between each edge and the content. Use the app icon at the top of your widget for alignment guidance. Content tends to work well when lined up with the center of this icon. If your app offers a grid-style layout, make sure you provide sufficient and equal padding between grid items. If possible, limit grids of icons and buttons to four per row.

Be adaptable. The width of a widget varies, depending on the device and orientation. The height and information displayed by a widget depends on whether it’s collapsed or expanded (not all widgets support expansion). A collapsed widget is the height of roughly two and a half table rows. An expanded widget is ideally no taller than the height of the screen. The quick action list only shows widgets in a collapsed state. When collapsed, a widget shows essential information that can stand alone. When expanded, a widget shows additional information that enhances the primary information. The Weather widget, for example, shows the current weather conditions when collapsed, but adds the hourly forecast when expanded.

Avoid customizing the background of a widget. The light, blurred widget background provided by the system is designed for consistency and legibility. Use it whenever possible. Never use a photo as a background, as it may clash with the Lock and Home screen wallpaper.

In general, use the system font in black or dark gray for text. The system font is designed for legibility, and dark colors work well with the standard widget background.

When appropriate, let people jump to your app to do more. Your widget should operate independently from your app. However, if people might occasionally need to do more than what your widget offers, make it easy to do so. Don’t include an “Open App” button that takes space away from your content. Instead, let people tap the content itself to see or edit it in your app. In the Calendar widget, for example, you can tap an event to open it in the Calendar app. Never use your widget to open other apps.

Pick a good widget name. An app icon and a title appear above each widget’s content. In general, your widget’s name should match your app’s name. If your app offers multiple widgets, consider using your app name for the primary one and clear, concise names for the others. If you use custom titles, consider prefixing them with your app name. For example, the Maps widget for showing nearby locations is titled Maps Nearby. The inclusion of your app name instills confidence that the widget is in fact provided by your app.

Let people know when authentication adds value. If your widget provides additional functionality when someone is logged into your app, make sure people know about this. For example, an app that shows upcoming reservations might include a message that says “Sign into the app to view your reservations.” when people aren’t logged in.

Choose a widget for the quick action list. If your app has multiple widgets, pick one to appear in the quick action menu that appears when someone applies pressure to your app icon on the Home screen using 3D Touch.

For developer guidance, see App Extension Programming Guide.

Apple Pay

Apple Pay is a secure, private, hassle-free way to make payments for physical goods and services, as well as donations, within iOS and watchOS apps. Users quickly and easily authorize payments and provide shipping and contact information, using Touch ID and credentials that are securely stored on the device.

Apps indicate when they accept Apple Pay. During checkout, a payment sheet shows the credit or debit card, purchase amount (including tax and fees), shipping options, and contact information. Users make any necessary adjustments and then use Touch ID to authorize payment and complete the purchase.

Websites can also accept Apple Pay. For website-specific design guidance, see Apple Pay on the Web Human Interface Guidelines.

TIP

It’s important to understand the difference between Apple Pay and In-App Purchase. Use Apple Pay to sell physical goods such as groceries, clothing, and appliances; for services such as club memberships, hotel reservations, and tickets for events; and for donations. Use In-App Purchase to sell virtual goods, such as premium content for your app, and subscriptions for digital content. See In-App Purchase.

Buttons

The system implements the following button styles for use by apps that accept Apple Pay.

Apple Pay or Buy with Apple Pay button. Use one of these buttons in your app wherever payment is made, such as on a product detail page or shopping cart page. Tapping this button immediately displays a payment sheet, where users complete the checkout process.

Set up Apple Pay button. Show this button when the device supports Apple Pay, but Apple Pay hasn’t been set up yet. Tapping this button opens the Wallet app and initiates the process of adding a card.

Donate with Apple Pay button. Approved nonprofits can use this button to facilitate donations. On devices running older systems that don't support the Donate with Apple Pay button, display the Apple Pay button instead.

Pay with Apple Pay button. Card issuers can use this button to let people quickly bring up their card in the Wallet app when making a purchase in a physical store.

Styles

All Apple Pay buttons can be displayed using the following styles:

Black. Use on white or light-colored backgrounds that provide sufficient contrast. Don’t use on black or dark backgrounds.

White with outline rule. Use on white or light-colored backgrounds that don’t provide sufficient contrast. Don’t place on dark or saturated backgrounds.

White. Use on dark or colored backgrounds that provide sufficient contrast.

Size and Position

Maintain the minimum width. The minimum width for all Apple Pay buttons is 32pt (32px @1x, 64px @2x).

Maintain minimum clear space. The minimum amount of clear space required around an Apple Pay button is 1/10 of the button's height. Other content, such as graphics and text, should not infringe on this space.

Display the Apple Pay button prominently. Make Apple Pay buttons the same size or larger than other payment buttons. Ideally, scrolling shouldn't be required to see the Apple Pay button.

Position Apple Pay buttons consistently in regard to Add to Cart buttons. Place the Apple Pay button to the right of or above an Add to Cart button.

Apple Pay Mark

Use the Apple Pay mark graphic to communicate that Apple Pay is an available payment option when showing other payment options in a similar manner. Download the Apple Pay mark graphic and its usage guidelines here.

Referring to Apple Pay in Text

You can use plain text to promote Apple Pay and indicate that Apple Pay is a payment option.

Purchase with Apple Pay simply by using Touch ID on your iPhone.

Incorrect

Purchase with ApplePay simply by using Touch ID on your iPhone.

Incorrect

Purchase with  Pay simply by using Touch ID on your iPhone.

Incorrect

Purchase with APPLE PAY simply by using Touch ID on your iPhone.

Capitalize Apple Pay in text as it appears on the Apple Trademark list. Use two words with an uppercase A, an uppercase P, and lowercase for all other letters. Display Apple Pay entirely in uppercase only when doing so is necessary for conforming to an established typographic interface style, such as if your app capitalizes all text. See Apple Trademark List.

Never use the Apple logo to represent the name Apple in text. In the United States, use the registered trademark symbol (®) the first time Apple Pay appears in body text. Do not include a registered trademark symbol when Apple Pay appears as a selection option during checkout.

Coordinate the font face and size with your app. Don't mimic Apple typography. Instead, use text attributes that are consistent with the rest of your app.

Don't translate Apple Pay. Always use Apple trademarks in English, even when they appear within non-English text.

When promoting your app's use of Apple Pay, refer to the App Store Marketing Guidelines.

Presenting Apple Pay as a Payment Option

Offer Apple Pay on all devices that support it. If the device supports Apple Pay but Apple Pay hasn't been set up yet, present the Set up Apple Pay button. If the device doesn’t support Apple Pay, don’t offer Apple Pay as a payment option.

Use only the Apple-provided API to display Apple Pay buttons. Unlike a button graphic, the buttons produced by the API always have the correct appearance and are localized automatically. Never create custom payment buttons.

Don’t disable or hide an Apple Pay button. If an Apple Pay button can’t be used yet, such as when a product size or color hasn’t been selected, gracefully point out the problem after the button is tapped.

Use the Apple Pay mark only to communicate that Apple Pay is accepted. The mark doesn’t facilitate payment. Never use it as a payment button or position it as a button.

For developer guidance, see Apple Pay Programming Guide.

Streamline the Checkout Process

People appreciate using Apple Pay to make purchases quickly and easily. The payment sheet should let them promptly authorize payment and complete the transaction.

Make Apple Pay the default payment option when possible. If Apple Pay is enabled, assume the person wants to use it. Consider presenting the Apple Pay button as the first or only payment option, displaying it larger than other options, or using a line to visually separate it from other choices.

Accelerate single-item purchases with Apple Pay buttons on product detail pages. In addition to offering a shopping cart, consider putting Apple Pay buttons on product detail pages so users can quickly purchase an individual item. Purchases initiated in this way should be for an individual item only, and should exclude any items that already reside in the user's shopping cart. If the user's shopping cart contains an item purchased directly from a product detail page, remove the item from the cart after the purchase is complete.

Accelerate multi-item purchases with express checkout. Consider providing an express checkout feature that immediately displays the payment sheet, allowing users to quickly purchase multiple items using a single shipping speed and destination.

Don't interrupt checkout after the Apple Pay button is tapped. Gather the information you need before people reach the Apple Pay button. When a user taps the Apple Pay button, they should immediately see the payment sheet. Displaying alerts or views that force people to take additional steps after deciding to complete their purchase adds friction and frustration to the process. For example, let people choose options, such as color and size, on a product detail page. On a checkout page, use highlighting or an alert to identify missing information before showing the payment sheet.

Collect optional information before checkout begins. There’s no way to input data on the payment sheet, so collect any optional information such as promo codes, redemption codes, gift messages, and delivery instructions ahead of time.

Gather multiple shipping speeds and destinations before showing the payment sheet. The payment sheet lets people select a single shipping speed and destination for an entire order. If your customers can choose different shipping speeds and destinations for individual items in an order, collect those details before Apple Pay checkout begins.

Prefer information from Apple Pay. Assume that Apple Pay information is complete and up to date. Even if your app has existing contact, shipping, and payment information, consider fetching the latest from Apple Pay during checkout to reduce potential corrections.

Display an order confirmation or thank-you page. After checkout, use an order confirmation page to provide details about when the order will ship and how to check its status. Listing Apple Pay on the confirmation page isn’t necessary, but if you do list it, show it after the last four digits of the account used to process the transaction or as a separate note. For example: "1234 (Apple Pay)" or "Paid with Apple Pay."

Don't require account creation prior to purchase. If you want people to register for an account, ask them to do so on the order confirmation or thank you page. Prepopulate as many registration fields as possible using information provided by the payment sheet during checkout.

Customize the Payment Sheet

You can customize the contents of the payment sheet depending on the information needed to complete the transaction.

Only present and request essential information. People may get confused or concerned if the payment sheet includes extraneous information. For example, it makes sense to see a contact email address but not a shipping address if the purchase, such as a gift card, will be delivered electronically. Showing or asking for a shipping address in this scenario may give the false impression that something will be physically delivered.

Let people choose the shipping method in the payment sheet. To the extent space permits, show a clear description, a cost, and, optionally, an estimated delivery date for each available option.

Keep line items short. Make line items specific and easily understandable at a glance. Whenever possible, they should fit on a single line.

Use line items to explain additional charges, discounts, and pending costs. A line item includes a label and cost. Don’t use line items to show an itemized list of products that make up the purchase.

Provide a business name after the word PAY on the same line as the total. Use the same business name people will see when they look for the charge on their bank or credit card statement. This provides reassurance that payment is going to the right place. If your app acts as an intermediary and is not the end merchant for a payment, clearly indicate this in the following format: PAY [END\_MERCHANT\_NAME] (VIA [YOUR\_APP\_NAME]).

Clearly disclose when additional costs may be incurred after payment authorization. In some apps, the total cost may be unknown at checkout time. For example, the price of a car ride based on distance or time might change after checkout. Or, a customer might want to add a tip after a product has been delivered. In situations like these, provide a clear explanation in the payment sheet and a subtotal marked as "AMOUNT PENDING." If you are preauthorizing a specific amount, also make sure the payment sheet accurately reflects this information.

Handle payment errors gracefully. If an error occurs during checkout, dismiss the payment sheet and display an alert or screen that explains the problem and offers advice for resolving it. For related guidance, see Alerts.

Supporting Subscriptions

Your app can use Apple Pay to request authorization for a recurring fee. This may be a fixed amount such as a monthly movie ticket subscription, or a variable amount such as a weekly produce order. The initial authorization can also include discounts and additional fees.

Clarify subscription details before showing the payment sheet. Before asking the user to authorize a recurring payment, make sure they fully understand the billing frequency and any other terms of service.

Include line items that reiterate billing frequency, discounts, and additional upfront fees. Use these line items to remind the user what they're authorizing.

Clarify the current payment amount in the total line. Make sure the user knows what they're being billed at the time of authorization.

Only show the payment sheet when a subscription change results in additional fees. When the user changes their subscription, authorization isn't necessary if the cost decreases or remains the same.

Accepting Donations

Approved nonprofits can use Apple Pay to accept donations.

Use a line item to denote a donation. Display a line item on the payment sheet that reminds users they're authorizing a donation. For example: DONATION $50.00.

Streamline checkout by offering predefined donation amounts. You can reduce steps in the donation process by offering one-tap recommended donations, such as $25, $50, $100. Be sure to include an Other Amount option too, so users can customize the donation if they prefer.

GameKit

GameKit provides features you can use to create great social games. Games can post scores and achievements, display leaderboards, add friends, discover new games, and more.

Don’t design a custom sign-in screen. People are automatically prompted to sign into GameKit-enabled games if they’re not already signed in on their device. Displaying a custom sign-in screen is repetitive and can be confusing to people who are expecting the standard one.

In general, use standard GameKit interfaces. In rare cases, it might make sense for a game to include a custom version of a GameKit-provided screen, but doing so risks confusing people. The standard GameKit screens are designed to provide a familiar experience to a large gaming community across all Apple platforms.

Let people turn off voice chat. Some people might not want voice chat to be on automatically when they start your game, and everyone appreciates the ability to turn off voice chat in certain situations.

For developer guidance, see GameKit.

HealthKit

The Health app serves as a central repository for health and fitness data in iOS. With the user’s permission, apps built with HealthKit can communicate with the Health app to access and share information. A user might permit a nutrition app, for example, to retrieve weight and activity data, so the app can define calorie consumption goals and make dietary recommendations. A nutrition app could also send the number of calories actually consumed to the Health app to include in global progress metrics and share with other apps.

Request access to health data for valid reasons. HealthKit is designed for use in health and fitness apps. People are unlikely to trust apps that request access to unnecessary or seemingly unrelated private health data.

Request access to health data only when needed. It makes sense to request access to weight information when filling out a weight-loss profile, for example, not immediately after the app launches. Requesting in-context health data helps communicate your app’s intentions.

Use the standard permission screen to clarify intent. People expect to see the system-provided permission screen when asked to approve access to health data. Add short but descriptive messages to this screen to clarify why the information is needed and the benefits of granting access. Avoid adding custom screens that replicate the standard permission screen’s behavior or content.

Manage health data sharing solely through the system’s privacy settings. People expect to globally manage access to health information in Settings > Privacy. Don’t build additional screens in your app that affect the flow of health data. Doing so results in confusion and an inefficient interface.

Don't use Health app icons, images, or screenshots. Like all Apple imagery, these designs are copyrighted and shouldn’t appear in your app or marketing materials. You can include an Activity ring element in your app to display Move, Exercise, and Stand progress. See Activity Rings.

Don't use the term HealthKit. HealthKit is the development framework used to access data stored in the Health app. If you need to explain how your app works with this data, use the term the Health app. For example, you might say that your app “saves information to the Health app” or “uses data from the Health app.”

For developer guidance, see HealthKit.

Activity Rings

Apps can enhance their health and wellness offerings by displaying an Activity ring element that shows an individual’s progress toward Move, Exercise, and Stand goals. This element always contains three rings, whose colors and meanings match those provided by the Activity app.

Use Activity rings for Move, Exercise, and Stand information only. Activity rings are designed to consistently represent progress in these specific areas. Don’t attempt to replicate or modify Activity rings for other purposes. Never use Activity rings to display other types of data. Never show Move, Exercise, and Stand progress in another ring-like element.

Use Activity rings to show progress for a single person. Never use Activity rings to represent data for more than one person, and make sure it’s obvious whose progress is shown, such as by using a label, a photo, or an avatar.

Don’t use Activity rings for ornamentation. Activity rings should provide information to people, not embellish your app’s design. Never display Activity rings in labels or background graphics.

Don’t use Activity rings for branding. Use Activity rings strictly to display Activity progress in your app. Never use Activity rings in your app’s icon or marketing materials.

Maintain Activity ring and background colors. For a consistent user experience, the visual appearance of Activity rings must always be the same, regardless of the context in which they appear. Never change the look of the rings or background by using filters, changing colors, or modifying opacity. Instead, design the surrounding interface to blend with the rings. For example, enclose the rings within a circle. Always scale the rings appropriately so they don’t seem disconnected or out of place.

Maintain Activity ring margins. An Activity ring element must include a minimum outer margin of no less than the distance between rings. Never allow other elements to crop, obstruct, or encroach upon this margin or the rings themselves. To display an Activity ring element within a circle, adjust the corner radius of the enclosing view rather than applying a circular mask.

Differentiate other ring-like elements from Activity rings. Mixing different ring styles can lead to a visually confusing interface. If you must include other rings, use padding, lines, or labels to separate them from Activity rings. Color and scale can also help provide visual separation.

Provide app-specific information only in Activity notifications. The system already delivers Move, Exercise, and Stand progress updates. Don’t repeat this same information, and never show an Activity ring element in your app’s notifications. It’s fine to reference Activity progress in a notification, but do so in a way that’s unique to your app and doesn’t replicate the same information provided by the system.

For developer guidance, see HKActivityRingView.

HomeKit

HomeKit lets people securely control connected accessories in their homes using Siri or the Home app on iPhone, iPad, and Apple Watch. In iOS, the Home app also lets users manage and configure accessories. Your iOS, tvOS, or watchOS app can also integrate with HomeKit to provide a custom or branded home automation experience.

HomeKit Terminology

Home automation can be intimidating. To help make it approachable, all HomeKit apps should use a friendly, conversational language that people understand.

Homes

In HomeKit, a home represents a physical home, office, or other location of relevance to the user. People might have multiple homes that are far apart, such as a main home and a vacation home. Or, they may have homes that are close together, but considered separate, such as a main home and a guest cottage.

Rooms

A room represents a physical room in a home. Rooms don’t have specific characteristics such as size or location. They’re simply meaningful names, such as Bedroom or Office. Rooms allow accessories to be organized and controlled based on their actual location within a home. Rooms also enable voice commands like “Siri, turn on the bedroom lights.”

Zones

A zone represents a specific area in a home, such as Upstairs or Downstairs. Zones allow rooms, typically in close proximity to one another, to be grouped, organized, and controlled. Zones also enable voice commands like “Siri, turn on all of the lights upstairs.”

Accessories, Services, Characteristics, and Actions

An accessory is a physical, connected HomeKit accessory, like a ceiling fan, light, or camera. People add accessories to rooms in their homes. Accessories communicate with iCloud or a server to receive instructions, respond to those instructions, and provide status updates. This communication takes place directly or through an iOS device such as Apple TV or iPad in the home that is operating as a hub.

A controllable feature of an accessory, such as the light on a connected lamp, is known as a service. Some accessories offer multiple services. For example, a connected garage door might let you individually control both the light and the door, or a connected outlet might let you control the top outlet and the bottom outlet separately. Apps don't actually use the word service. Instead, they use terms that describe the service, such as Kitchen Light and Kitchen Fan.

A characteristic is a controllable attribute of a service. For example, on a ceiling fan, the fan service might have a speed (off, low, medium, high) characteristic and the light service might have a brightness characteristic. Apps don't actually use the word characteristic. Instead, they use terms that describe the service, such as speed and brightness.

An action is the act of changing a characteristic of a service. For example, adjusting the speed of a fan or the brightness of a light. Actions are initiated by users and through automation.

Scenes

A scene is a group of actions for controlling one or more services on one or more accessories. Someone might, for example, create a Movie Time scene that lowers the shades and dims the lights in the living room, or a Good Morning scene that turns on the lights, raises the shades, and starts the coffee maker in the kitchen.

Automations

Automations cause accessories to react to certain situations, such as when your location changes, a particular time of day occurs, another accessory turns on or off, or a sensor detects something. An automation could be used, for example, to turn your house lights on at sunset or when you pull up the driveway.

Provide a Great Experience

Through HomeKit, your app can help people:

Set up and manage homes, rooms, zones, and scenes

Find, add, remove, and interact with accessories

Give trusted family and friends access to control accessories

Use Siri to control accessories with voice commands

Check the status of accessories

Automate common accessory behaviors and settings

Simplify Accessory Setup and Interaction

Make accessory setup quick, intuitive, and automatic whenever possible. Ideally, your app automatically detects new accessories and presents them prominently. Don’t force people to create an account before they can add an accessory.

Prompt users to assign accessories to rooms after setup. After configuring a new accessory, immediately suggest adding the accessory to a room. If a home or room doesn't exist yet, provide a way to create one. If only one room exists, suggest adding the accessory to that room but also provide a way to add another room.

Make sure accessories are easy to identify. Accessories should be recognizable when users interact with them, during the setup process, and when adjusting settings later. If a living room contains multiple lights, for example, provide a way to differentiate each one. In this scenario, you might provide a control that physically flashes a light to identify it and let people give each light’s service a unique name, such as desk lamp or chandelier.

Provide multiple ways to locate accessories. People want to interact with accessories quickly, such as unlocking the door while walking up to the house. Implement ways to filter and search for accessories by name, type, or location in the home. Consider providing a dashboard of favorite accessory services.

Report accessory statuses accurately and help the user resolve problems. If an accessory can’t be reached, don’t assume its status. For instance, if a lock is unreachable, don’t say it’s locked or unlocked based on the last known status. Instead, explain that the lock is currently unreachable, suggest possible reasons why, and offer suggestions for fixing the problem.

Enable Hands-Free Interaction

Voice-driven and automatic interactions make it easy to control accessories.

During setup, teach people the commands Siri understands. Siri makes it easy to perform sophisticated operations with a single statement. Siri automatically recognizes the names of homes, rooms, zones, services, and scenes in HomeKit. When appropriate throughout your app, show examples of phrases that work with Siri, such as “You can say ‘Siri, set the house to movie mode.’ to activate this configuration.”

When there are multiple homes, set a primary home. Siri needs to know which home to target when saying something like “Turn on my living room lights.” If there’s only one home, that home is targeted for all commands. If the person has multiple homes and no primary home is configured, the current home—the one the person is in at the moment—is targeted.

Make the configuration of automations intuitive. HomeKit accessories can be controlled automatically, based on conditional relationships such as time, location, and the behavior of other accessories. For example, the kitchen lights could be set to come on when the garage door opens and it’s after sunset. Setting up conditional relationships like this can be confusing, so make this process as simple as possible. Design clear interfaces and use verbiage that mirrors the way people speak.

Make Your App Approachable

Use friendly, conversational terminology throughout your app. Avoid using acronyms and technical jargon that might cause confusion or frustration. Use standard HomeKit terms (home, room, zone, accessory, automation) to provide a consistent and approachable experience.

Use everyday language when referring to services and characteristics. For services, use terms like Kitchen Light and Kitchen Fan. For characteristics, use terms like brightness and speed.

Use appropriate controls to let people change characteristics. For example, use a switch to let people toggle a service on and off.

Provide good defaults. Every user's home setup is different. During home configuration, offer generic defaults that are widely applicable. For example, your app might assume that all users have a primary home they want to name Home. Default names should be concise, well-known words or phrases.

Allow customization. Make sure home, room, zone, accessory, service, and automation names are customizable. For example, someone might want to rename a connected outlet or switch to desk lamp if that’s what the outlet or switch controls. Never use company names or model numbers for service names.

Assist with name conflicts. Siri requires that home, room, zone, service, and scene names be unique. When name conflicts arise, explain the problem clearly and offer alternatives that make sense and are easy to remember.

Accessory Considerations

As you design your HomeKit app, consider the user experience of specific accessories.

HomeKit Cameras

HomeKit apps can display still images or streaming video from a connected HomeKit IP camera.

Don't block camera imagery. It's fine to supplement the camera's content with useful features, such as an alert calling attention to potentially interesting activity. However, avoid covering portions of the camera's imagery with other content.

Show a microphone button only if the camera supports bidirectional audio. A microphone button that can't be used is a waste of space and confusing to the user.

Learn More

For developer guidance, see HomeKit Developer Guide and HomeKit. If you’re an MFi licensee, visit the MFi portal for guidance on naming and messaging for accessory packaging.

iCloud

iCloud is a service that lets people seamlessly access the content they care about—photos, videos, documents, and more—from any device, without performing explicit synchronization. A fundamental aspect of iCloud is transparency. People don’t need to know where content resides.They can just assume they’re always accessing the latest version.

Make it easy to use your app with iCloud. People enable iCloud in Settings and expect apps to work with it automatically. If you think people might want to choose whether to use iCloud with your app, show a simple option the first time your app opens that provides a choice between using iCloud for all data or not at all.

Avoid asking which documents to keep in iCloud. Most people expect all of their content to be available in iCloud and don’t want to manage the storage of individual documents. Consider how your app handles and exposes content, and try to perform more file-management tasks automatically.

Keep content up to date when possible. In iCloud-enabled apps, its best when people always have access to the most recent content. However, you need to balance this experience with respect to device storage and bandwidth constraints. If your app works with very large documents, it may be better to let people control when updated content is downloaded. If your app fits in this category, design a way to indicate that a more recent version of a document is available in iCloud. When a document is updating, provide subtle feedback if the download takes more than a few seconds.

Respect iCloud storage space. iCloud is a finite resource for which people pay. Use iCloud to store information people create and understand, and avoid using it for app resources or content you can regenerate. Even if your app doesn’t implement iCloud support, remember that iCloud backups include the contents of every app’s Documents folder. To avoid using up too much space, be picky about the content you place in the Documents folder.

Make sure your app behaves appropriately when iCloud is unavailable. If someone manually disables iCloud or turns on Airplane Mode, you don’t need to display an alert notifying them the iCloud is unavailable. They already know this. However, it may still be helpful to unobtrusively let them know that changes they make won’t be available on other devices until iCloud access is restored.

Keep app state information in iCloud. In addition to storing documents and other files, you can use iCloud to store preferences and information about the state of your app. For example, a magazine app might store the last page viewed so when the app is opened on another device, someone can continue reading from where they left off. If you use iCloud to store preferences, make sure it’s for ones people want applied to all of their devices. For example, some settings might be more useful at work than at home.

Warn about the consequences of deleting a document. When someone deletes a document in an iCloud-enabled app, the document is removed from iCloud and all other devices too. Show a warning and ask for confirmation before performing the deletion.

Make conflict resolution prompt and easy. To the extent possible, try to detect and resolve version conflicts automatically. If this can’t be done, display an unobtrusive notification that makes it easy to differentiate and choose between the conflicting versions. Conflict resolution should always occur as early as possible, so time isn’t wasted in the wrong version.

Include iCloud content in search results. People with iCloud accounts assume their content is universally available, and they expect search results to reflect this perspective.

For developer guidance, see iCloud Design Guide and CloudKit Quick Start.

In-App Purchase

In-App Purchase enables people to securely pay for digital products through a custom store contained within your app. Apps often use In-App Purchase to let people add premium features, subscribe to recurring content, and purchase virtual items, such as a new level or a weapon in a game.

TIP

It’s important to understand the difference between In-App Purchase and Apple Pay. Use In-App Purchase to sell virtual goods, such as premium content for your app, and subscriptions for digital content. Use Apple Pay to sell physical goods such as groceries, clothing, and appliances. Also use Apple Pay for services such as club memberships, hotel reservations, and tickets for events. See Apple Pay.

Design an integrated shopping experience. People shouldn’t think they’ve entered a different app when they use your store. Present products and handle transactions in a way that feels at home in your app.

Use simple, succinct product names and descriptions. It’s best when people can scan a set of products quickly and find the ones they’re interested in. Use plain, direct language and titles that don’t truncate or wrap.

Use the default confirmation alert. When someone initiates an in-app purchase, the system displays a confirmation alert to avoid accidental purchases. Don’t attempt to modify this alert.

Note that In-App Purchase collects payment for products you’ve registered with the App Store, but your app must implement any functionality needed to actually present a store, unlock features, or download content. For business guidelines and developer guidance, see In-App Purchase for Developers and In-App Purchase Programming Guide. For detailed information about what you can and can’t sell in your app, as well as In-App Purchase usage requirements and restrictions, see your Apple Developer Program license agreement.

Live Photos

Live Photos capture favorite memories in a sound- and motion-rich interactive experience that adds vitality to traditional still photos. With this feature enabled, the Camera app captures additional content—including audio and extra frames before and after the photo is taken. Simply press a Live Photo to see it spring to life.

Display Live Photos as traditional photos in environments that don’t support Live Photos. Don’t attempt to replicate the Live Photos experience provided in a supported environment. Instead, show a traditional, still representation of the photo.

Make Live Photos easily distinguishable from still photos. The best way to identify a Live Photo is through a hint of movement. Note that there are no built-in Live Photo motion effects, like the one that occurs as you swipe through photos in the full-screen browser of Photos app. Any motion effects like this must be custom designed and implemented. In cases where movement isn’t possible, show a system-provided badge above the photo. This badge can be displayed as an overlay with a shadow or as a solid color without a shadow. A badge variant is also available for situations where a Live Photo appears as a traditional photo. Never include a playback button that could be interpreted as a video playback button.

Keep badge placement consistent. If you show a badge, put it in the same location on every photo. Typically, a badge looks best in a corner of a photo.

Apply adjustments to all frames. If your app lets people apply effects or adjustments to a Live Photo, make sure those changes are applied to the entire photo. If you don’t support this, give people the option of converting it to a still photo.

Keep Live Photo content intact. It’s important for people to experience Live Photos in a consistent way that uses the same visual treatment and interaction model across all apps. Don’t disassemble a Live Photo and present its frames or audio separately.

Implement a great photo sharing experience. If your app supports photo sharing, let users preview the entire contents of Live Photos before deciding to share. Always offer the option to share Live Photos as traditional photos.

Clearly indicate when a Live Photo is downloading and when the photo is playable. Show a progress indicator during the download process and provide some indication when the download is complete.

Live Photo support is available in iOS 9.1 and later.

ResearchKit

A research app lets people everywhere participate in important medical research studies. The ResearchKit framework provides predesigned screens and transitions that make it easy to design and build an engaging custom research app. For developer guidance, see researchkit.org.

These guidelines are for informational purposes only and do not constitute legal advice. Contact an attorney to obtain advice with respect to the development of a research app and any applicable laws.

Creating the Onboarding Experience

When opening a research app for the first time, people encounter a series of screens that introduce them to the study, determine their eligibility to participate, request permission to proceed with the study, and, when appropriate, grant access to personal data. These screens aren’t typically revisited once they’ve been completed, so clarity is essential.

Always display the onboarding screens in the correct order.

1. Introduction

Provide an introduction that informs and provides a call to action. Clearly describe the subject and purpose of your study. Also allow existing participants to quickly log in and continue an in-progress study.

2. Determine Eligibility

Determine eligibility as soon as possible. People don’t need to move on to the consent section if they’re not eligible for the study. Only present eligibility requirements that are necessary for your study. Use simple, straightforward language that describes the requirements, and make it easy to enter information.

3. Get Informed Consent

Make sure participants understand your study before you get their consent. ResearchKit helps you make the consent process concise and friendly, while still allowing you to incorporate into the consent any legal requirements or requirements set by an institutional review board or ethics review board. Make sure that your app complies with the applicable App Store Guidelines, including the consent requirements. Typically, the consent section explains how the study works, ensures that participants understand the study and their responsibilities, and gets the participant’s consent.

Break a long consent form into easily digestible sections. Each section can cover one aspect of the study, such as data gathering, data use, potential benefits, possible risks, time commitment, how to withdraw, and so on. For each section, use simple, straightforward language to provide a high-level overview. If necessary, provide a more detailed explanation of the section that participants can read by tapping a Learn More button. Participants should be able to view the entire consent form before they agree to participate.

If it makes sense, provide a quiz that tests the participant’s understanding. You might do this for questions the participant would otherwise be asked when obtaining consent in person.

Get the participant’s consent and, if appropriate, some contact information. After agreeing to join the study, participants receive a confirmation dialog, which should be followed by screens in which they provide their signature and contact details. Most research apps email participants a PDF version of the consent form for their records.

4. Request Permission to Access Data

Get permission to access the participant’s device or data, and to send notifications. Clearly explain why your research app needs access to location, Health, or other data, and don’t request access to data that isn’t critical to your study. If your app requires it, also ask for permission to send notifications to the participant’s device.

Conducting Research

To get input from participants, your study might use surveys, active tasks, or a combination of both. Depending on the architecture of your study, participants may interact with each section multiple times or only once.

Create surveys that keep participants focused and engaged. ResearchKit provides many customizable screens you can use in your surveys, and makes it easy to present questions that require different types of answers, such as true or false, multiple choice, dates and times, sliding scales, and open-ended text entry. As you use ResearchKit screens to design a survey, follow these guidelines to provide a great experience:

Tell participants how many questions there are and about how long the survey will take.

Use one screen per question.

Show participants their progress in the survey.

Keep the survey as short as possible. Several short surveys tend to work better than one long survey.

For questions that require some additional explanation, use the standard font for the question and a slightly smaller font for the explanatory text.

Tell participants when the survey is complete.

Make active tasks easy to understand. An active task requires the participant to engage in an activity, such as speaking into the microphone, tapping fingers on the screen, walking, or performing a memory test. Follow these guidelines to encourage participants to perform an active task and give them the best chance of success:

Describe how to perform the task using clear, simple language.

Explain any requirements, such as if the task must be performed at a particular time or under specific circumstances.

Make sure participants can tell when the task is complete.

Managing Personal Information and Providing Encouragement

ResearchKit offers a profile screen you can use to let participants manage personal information while they’re in your research app. It’s also a good idea to design a custom screen that motivates people and gives them a way to track progress in the study. Ideally, both screens should be accessible at any time in your app.

Use a profile to help participants manage personal data related to your study. A profile screen can let people edit data that might change during the course of the study—such as weight or sleep habits—and remind them of upcoming activities. A profile screen can also provide an easy way to leave a study and view important information, such as the consent document and privacy policy.

Use a dashboard to show progress and motivate participants to continue. If appropriate for your study, use a dashboard to provide encouraging feedback, such as daily progress, weekly assessments, results from specific activities, and even results that compare the participant’s results with aggregated results from others in the study.

Social Media

People expect to have quick access to their favorite social media accounts, regardless of the current context. iOS makes it easy to integrate social media interactions into your app, including sharing content and displaying an activity feed.

Don't ask people to sign in. iOS employs a single sign-on model for accessing social media accounts, so you can get authorization to access an account without requiring authentication. If the user isn’t signed into an account, you can display a system-provided interface that allows them to do so.

Make sharing easy. Many people enable share extensions that make it easy to post content from anywhere to their favorite social media services. These extensions are accessed by tapping a sharing button, usually in a toolbar, to display a list of available services. Selecting a service then displays a composition view. Whenever possible, prepopulate this view with content before presenting it for editing. For example, a photo app could attach the active photo, a travel app could add the current location, or a cooking app could insert a URL for a favorite recipe.

For information about implementing social media interactions in your app, see Social. To learn about designing share extensions other apps can use to post content to your social media service, see Sharing and Actions.

Wallet

People use the Wallet app to view and manage passes—digital representations of information that is otherwise printed on small pieces of paper or plastic and kept in a physical wallet. Passes are used to take action in the real world, such as to board a plane, enter a venue, or receive a discount. Wallet also stores credit, debit, and store cards used by Apple Pay.

The PassKit framework makes it easy for apps to access, create, distribute, and update passes. This framework implements several templates, each of which defines the layout and fields for a specific style of pass.

Style Uses

Boarding pass Train tickets, airline boarding passes, and other types of transit passes. Each pass usually corresponds to a single trip with a specific starting and ending point.

Coupons Coupons, special offers, and other discounts.

Event tickets Entry into concerts, movies, plays, sporting events, or other events. Typically, each pass corresponds to a specific event, but you can also use a single pass for several events as with a season ticket.

Store cards Store loyalty cards, discount cards, points cards, and gift cards. If an account related to a store card carries a balance, the pass usually shows the current balance.

Generic pass Any other type of pass, such as a gym membership card or coat-check claim ticket.

Design a pass that looks great on all devices. Be mindful that passes appear differently on different devices. For example, strip and thumbnail images aren’t displayed on Apple Watch. Don’t put essential information in elements that might be unavailable on certain devices. Note that, even if you don’t create a watch-specific app, people can add your passes to their Apple Watch.

To the extent possible, avoid reproducing existing physical passes. Wallet matches the general design aesthetic of iOS to build trust and familiarity. Instead of replicating the appearance of a physical item, take this opportunity to design a clean, simple pass that follows the form and function of Wallet.

Avoid using a plain white background. A pass looks best when its background is a vivid, solid color or an image with strong, vibrant colors. As you design the background, make sure it doesn’t interfere with the readability of the content.

Use pass fields to display text. Each pass style defines fields that give your pass a consistent look and work with accessibility features. Avoid embedding text in images because not all images are displayed on all devices. Avoid using custom fonts that make text difficult to read.

Avoid using device-specific language. You can’t predict the device people will use to view your pass, so don’t include language that might not make sense on a particular device. For example, text that tells people to “slide to view” content doesn’t make sense when it appears on Apple Watch.

Be selective about the information on the front of a pass. People expect to glance at a pass and quickly get the information they need, so the front of a pass should be uncluttered and easy to read. If there’s additional information you think people might need, put it on the back of the pass. Just be aware that passes on Apple Watch don't include a back view.

Use the logo text field for your company name. Text in this field is rendered in a consistent font on all passes. To avoid clashing with other passes in Wallet, consider entering text here instead of using a custom font.

Update passes when appropriate. Even though physical passes don’t typically change, digital passes can be updated to reflect real-world events. An airline boarding pass, for example, could be updated to reflect flight delays.

Use a rectangular barcode when possible. A square barcode creates empty gutters on both sides and can vertically crowd the fields above and below it. Because of the layout of a pass, a rectangular barcode—such as a PDF417-style barcode—typically looks better.

Optimize images for performance. People often receive passes via email or Safari. To make downloads as fast as possible, use the smallest image files that still look great.

For developer guidance, see Wallet Developer Guide and PassKit.