Apple Human Interface Guidelines

Human Interface Guidelines

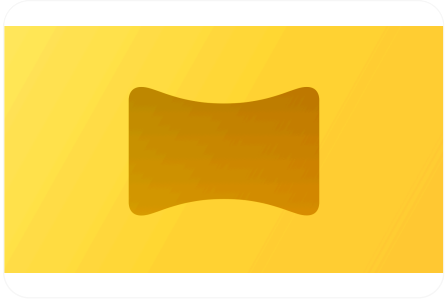
The HIG contains guidance and best practices that can help you design a great experience for any Apple platform.

New and updated

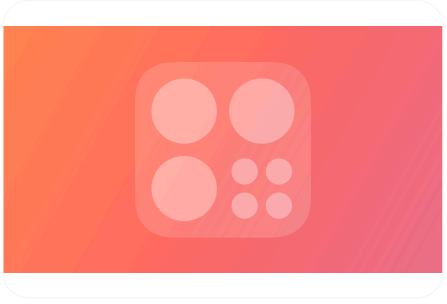
Camera Control

Gestures

Designing for games



Immersive experiences

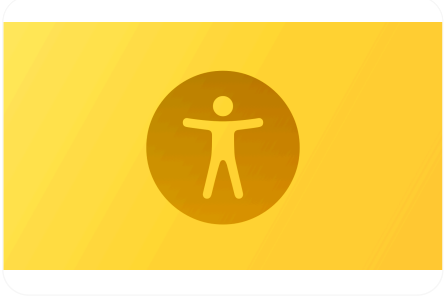


Controls

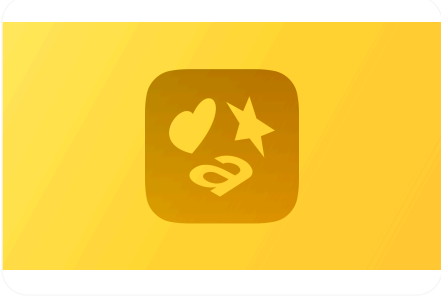
App icons

Featured

Designing for visionOS



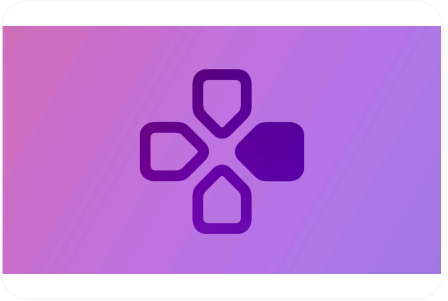
Accessibility



SF Symbols

Typography

Writing



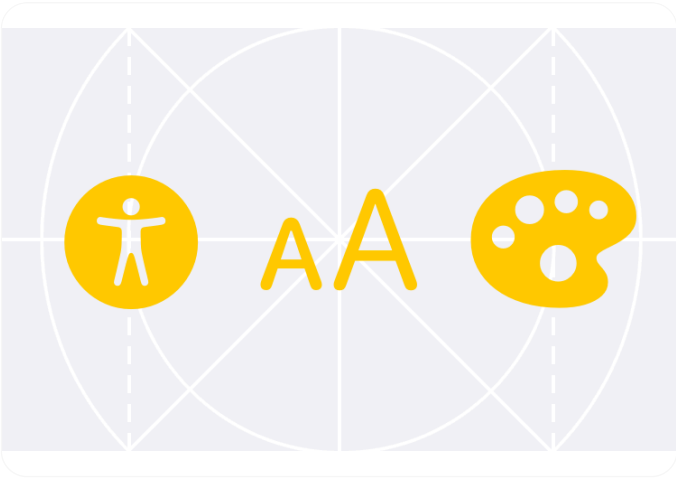
Game controls

Topics

Getting started

Create an app or game that feels at home on every platform you support.

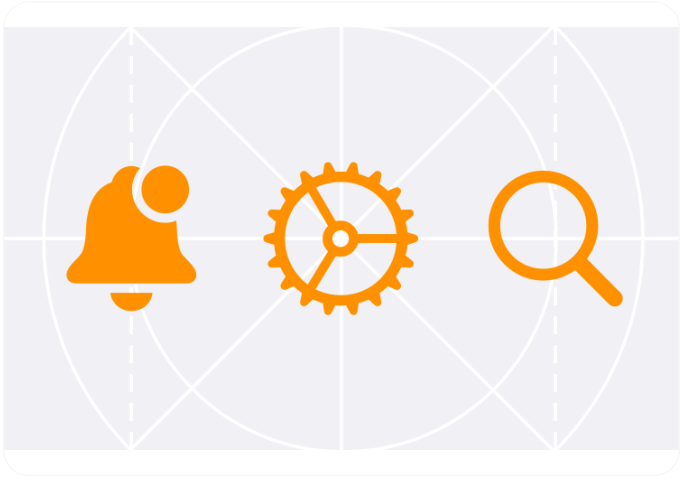
Learn More ›



Foundations

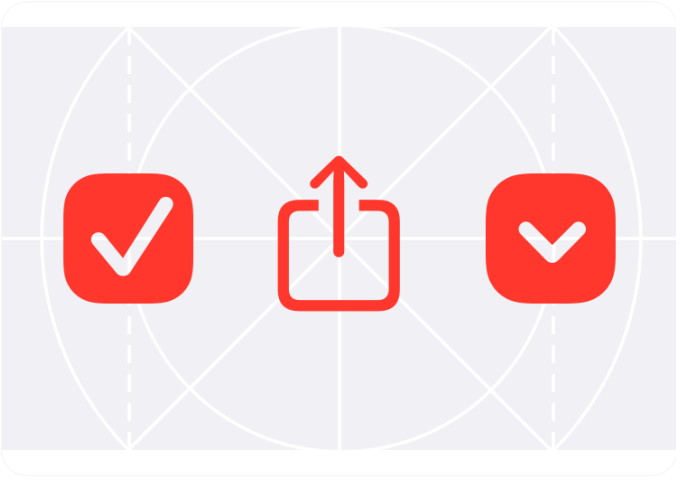
Understand how fundamental design elements help you create rich experiences.

Learn More ›



Patterns

Get design guidance for supporting common user actions, tasks, and experiences.

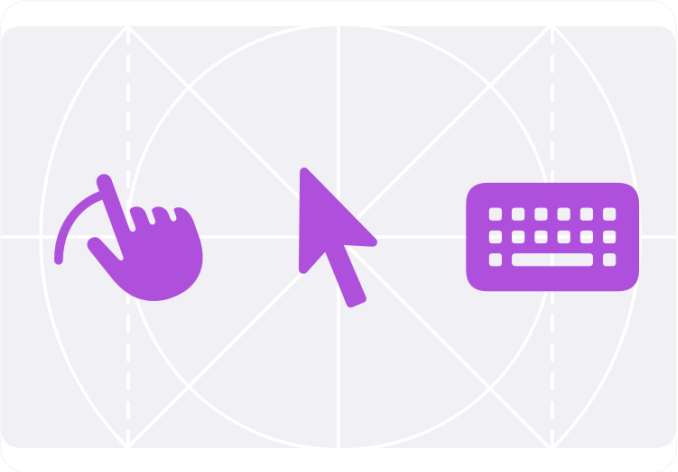


Components

Learn how to use and customize system-defined components to give people a familiar and consistent experience.

Learn More ›

Learn More ›



Inputs

Learn about the various methods people use to control your app or game and enter data.

Learn More ›

Technologies

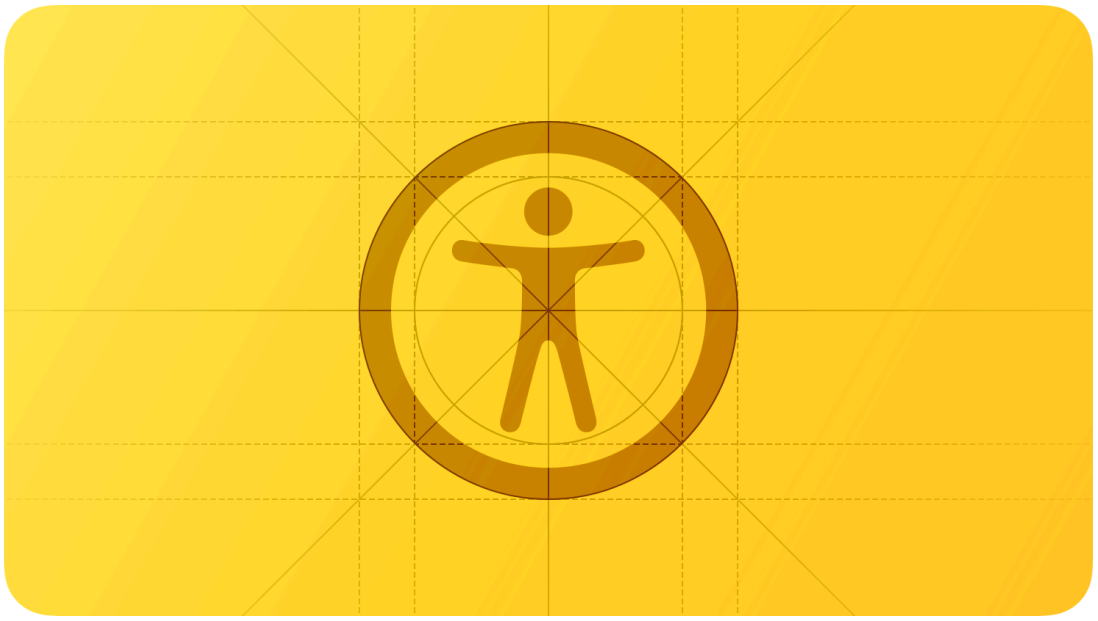
Discover the Apple technologies, features, and services you can integrate into your app or game.

Learn More ›

Accessibility

People use Apple's accessibility features to personalize how they interact with their devices in ways that work for them.

Supported platforms



An accessible app or game supports accessibility personalizations by design and helps everyone have a great experience, regardless of their capabilities or how they use their devices.

Approximately one in seven people have a disability that affects the way they interact with the world and their devices. People can experience disabilities at any age, for any duration, and at varying levels of severity. For example, situational disabilities — such as a wrist injury from a fall or voice loss from overuse — can affect the way almost everyone interacts with their devices at various times.

Best practices

Design with accessibility in mind. Accessibility is not just about making information available to people with disabilities — it's about making information available to everyone, regardless of their capabilities or situation. Designing your app with accessibility in mind means prioritizing *simplicity* and *perceivability* and examining every design decision to ensure that it doesn't exclude people with disabilities or who interact with their devices in different ways.

Simplicity — Support familiar, consistent interactions that make complex tasks simple and straightforward to perform.

Perceivability — Make sure that all content can be perceived whether people are using sight, hearing, or touch.

Support personalization. You already design your experience to adapt to environmental variations — such as device orientation, display size, resolution, color gamut, and split view — because you want people to enjoy it in any context and on all supported devices. With minimal

additional effort, you can design your app to support the accessibility features people use to personalize the ways they interact with their devices.

When you use standard components to implement your interface, text and controls automatically adapt to several accessibility settings, such as Bold Text, Larger Text, Invert Colors, and Increase Contrast. In your Unity-based game, you can use Apple's Unity plug-ins to support Dynamic Type (for developer guidance, see Apple Unity Plug-Ins).

Audit and test your app or game for accessibility. An audit examines every element in your experience and gives you a comprehensive list of issues to fix. When you test important user flows with accessibility features turned on, you gain an appreciation for the challenges of interacting with a device in different ways. You also discover places where your app might fail to deliver a great user experience.

For example, a common user flow in a social media app might be "post a response to a comment." The tasks that make up this flow could include:

- Read posted comments.
- Choose a comment for a response.
- Open the response view.
- Edit the response.
- Post the response.

For each critical user flow in your app or game, turn on an accessibility feature, such as VoiceOver, Reduce Motion, or Large Text Size, and make sure that you can complete every task in the flow without difficulty. After you fix the problems you uncover, turn on a different accessibility feature and run through the user flow again. To help you audit, test, and fix your app or game, consider using Xcode's Accessibility Inspector.

Interactions

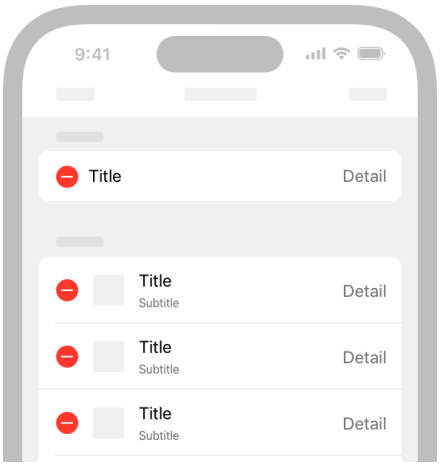
Assistive technologies like VoiceOver, Assistive Touch, Pointer Control, and Switch Control expand the ways people can interact with their devices. Because these technologies integrate with system-provided interactions, it's essential that you support the system interactions correctly in your app.

Gestures

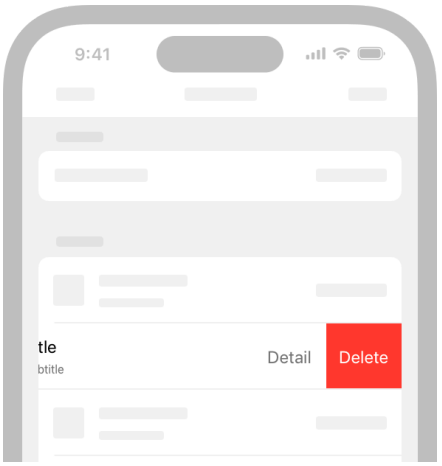
Don't override the platform gestures. People expect gestures that target system features — like swiping down to reveal Notification Center — to work regardless of the app they're using.

Prefer simplified gestures for common interactions. Complex gestures such as multifinger or multihand gestures, long presses, or gestures that require repeated movements can be challenging for many people. Using the simplest gestures possible improves the experience for everyone who interacts with your app.

Provide alternative ways to perform gesture-based actions. Include an option for people who may not be able to perform a specific gesture. For example, if people can use a gesture to delete a row in a table, you can also provide a way to delete items through an edit mode or by offering a Delete button in an item detail view.



Edit to delete.



Swipe to delete.

When possible, make your app's core functionality accessible through more than one type of physical interaction. For example, Camera on iPhone and iPad lets people take a photo by tapping the onscreen button or by pressing the device's volume down button. In addition to making photo-capture more convenient for everyone, these alternative interactions provide options to people who might have limited grip strength or dexterity.

If you define custom gestures, be sure to support assistive technologies that give people alternative ways to interact with your app. For example, with Pointer Control, people can use a wrist, index finger, or head-based pointer; with Dwell Control, they can use only their eyes to select and activate objects. One way to support technologies like VoiceOver, Dwell Control, and Switch Control is to implement custom actions; for developer guidance, see `UIAccessibilityCustomAction`.

Make drag and drop accessible in your iOS or iPadOS app. When you use the accessibility APIs to identify drag sources and drop targets in your app, assistive technologies can help people drag and drop items. For developer guidance, see `accessibilityDragSourceDescriptors` and `accessibilityDropPointDescriptors`.

Buttons and controls

Give all controls and interactive elements a hit target that's large enough. For example, on touchscreen devices, a hit target needs to measure at least 44x44 pt; in visionOS, place controls so that their centers are at least 60 pt apart. People with limited mobility need larger hit targets to help them interact with your app. It can be frustrating to interact with too-small controls in any platform, even when people use a pointer.

Characterize the accessibility of custom elements. You can use system APIs to tell assistive technologies how a component behaves. For example, using `button` or `NSAccessibilityButton` to characterize a view as a button means that VoiceOver speaks the view's description followed by the word *button*, which tells people that the view behaves like a button.

Use a consistent style hierarchy to communicate the relative importance of buttons. In iOS, iPadOS, and tvOS, for example, you can use the visually prominent filled style for the button that performs the most likely action in a view, using less prominent styles — such as gray or plain — for buttons that perform less important actions. (For developer guidance, see `UIButton.Configuration`.) In visionOS, system-provided buttons generally include a visible background by default. In iOS, iPadOS, visionOS, and for some buttons in macOS, people can also turn on Button Shapes to make it easier to distinguish active buttons from surrounding content.

Prefer the system-provided switch component. SwiftUI provides a switch that indicates its state by the position of its knob and its fill color. For some people, however, the addition of labels

makes it easier to perceive whether a switch is on or off. When you use system-provided switches, iOS, iPadOS, tvOS, visionOS, and watchOS automatically display on/off glyphs within them when people turn on On/Off Labels.

Consider giving links a visual indicator in addition to color, such as an underline. It's fine to use color to identify a link, but if you use it as the only indicator, people — such as those with color blindness or cognitive or situational attention impairments — may not be able to perceive the distinction.

User input

Let people input information by speaking instead of typing or gesturing. Adding a dictation button in a text entry field lets people choose speech as their preferred input method. If you create a custom keyboard, be sure to include a microphone key for dictation.

Support Siri or Shortcuts for performing important tasks by voice alone. To learn more about helping people use Siri interactions in your app, see Siri.

When possible, don't prevent people from selecting plain text. Many people rely on using selected text as input for translations and definitions.

Haptics

Support the system-defined haptics where available. Many people rely on haptics to help them interact with apps when they can't see the display. For example, system apps play haptics to notify people when a task has succeeded or failed or when an event is about to happen. Be sure to use the system-defined haptics consistently in your app so that you don't confuse people. For guidance, see Playing haptics.

> **Note**
> In platforms that don't play haptics, use other ways to provide feedback when people interact with custom objects, such as sound.

VoiceOver

VoiceOver gives audible descriptions of visible content, helping people get information and navigate when they can't see the display. In visionOS, VoiceOver uses Spatial Audio to help communicate the location of accessible objects.

> **Important**
> When VoiceOver is on in visionOS, apps that define custom gestures don't receive hand input by default. Instead, people can perform VoiceOver gestures to explore apps without worrying about an app interpreting their hand input. In VoiceOver's Direct Gesture mode, VoiceOver doesn't process its standard gestures, instead letting an app process hand input directly. For developer guidance, see Improving accessibility support in your visionOS app.

Content descriptions

Provide alternative descriptions for all images that convey meaning. If you don't describe the meaningful images in your content, you prevent VoiceOver users from fully experiencing your app. To create a useful description, start by reporting what would be self-explanatory to someone who is able to see the image. Because VoiceOver reads the text surrounding the image and any captions, describe only the information that's conveyed by the image itself.

The alternative description for this element is "Moving: 125 percent; Exercise: zero percent; Standing: 58 percent."

Make infographics fully accessible. Provide a concise description of the infographic that explains what it conveys. If people can interact with the infographic to get more or different information, you need to make these interactions available to VoiceOver users, too. The accessibility APIs provide ways to represent custom interactive elements so that assistive technologies can help people use them.

When an image is purely decorative and isn't intended to communicate anything important, hide it from assistive technologies. Making VoiceOver describe a purely decorative image can waste people's time and add to their cognitive load without providing any benefit.

Give each page a unique title and provide headings that identify sections in your information hierarchy. When people arrive on a page, the title is the first piece of information they receive from an assistive technology. To help people understand the structure of your app, create a unique title for each page that succinctly describes its contents or purpose. Similarly, people need accurate section headings to help them build a mental map of the information hierarchy of each page.

Help everyone enjoy your video and audio content. When you provide closed captions, audio descriptions, and transcripts, you can help people benefit from audio and video content in ways that work for them.

Closed captions give people a textual equivalent for the audible information in a video. You can also use closed captions to provide multiple translations for the same content, letting the system choose the version that matches the device's current settings. Because closed captions aren't always available, it's important to provide subtitles, too.

Audio descriptions provide a spoken narration of important information that's presented only visually.

A *transcript* provides a complete textual description of a video, covering both audible and visual information, so that people can enjoy the video in different ways.

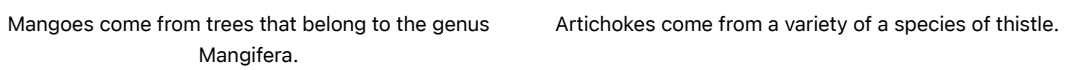
For developer guidance, see Selecting Subtitles and Alternative Audio Tracks.

Navigation

Make sure VoiceOver users can navigate to every element. VoiceOver uses accessibility information from UI elements to help people understand the location of each element and what it can do. System-provided UI components include this accessibility information by default, but VoiceOver can't help people discover and use custom elements unless you provide the information. For developer guidance, see Accessibility modifiers.

Improve the VoiceOver experience by specifying how elements are grouped, ordered, or linked. Proximity, alignment, and other contextual cues can help sighted people perceive the relationships among visible elements, but these cues don't work well for VoiceOver users. Examine your app for places where relationships among elements are visual only, and describe these relationships to VoiceOver.

For example, the layout below relies on proximity and centering to imply that each phrase is a caption for the image above it. However, if you don't tell VoiceOver that each image needs to be grouped with its phrase, VoiceOver reads, "A large container holding a variety of mangoes. A large container holding many green artichokes. Mangoes come from trees that belong to the genus Mangifera. Artichokes come from a variety of a species of thistle." This happens because VoiceOver reads elements from top to bottom by default. For developer guidance, see `shouldGroupAccessibilityChildren` and `accessibilityTitleUIElement()`.



Tell VoiceOver when visible content or layout changes. An unexpected change in content or layout can be very confusing to VoiceOver users, because it means that their mental map of the content is no longer accurate. It's crucial to report visible changes so that VoiceOver and other assistive technologies can help people update their understanding of the content. For developer guidance, see `UIAccessibility.Notification` (UIKit) or `NSAccessibility.Notification` (AppKit).

Help people predict when a control opens a different webpage or app. An unexpected change in context can cause confusion and require people to suddenly rebuild their mental model of the current experience. One way to draw attention to a potential change in context is append an ellipsis to a button's title. Throughout the system, an ellipsis trailing the title is the standard way for a button to communicate that it opens another window or view in which people can complete the action. For example, Mail in iOS and iPadOS appends an ellipsis to the Move Message button, signaling that a separate view opens, listing the destinations people can choose.

Provide alternative text labels for all important interface elements. Alternative text labels aren't visible, but they let VoiceOver audibly describe app elements, making navigation easier for people with visual disabilities. System-provided controls have useful labels by default, but you need to create labels for custom elements. For example, if you create an accessibility element that represents a custom rating button, you might supply the label "Rate."

Support the VoiceOver rotor when necessary. VoiceOver users can use a control called the *rotor* to navigate a document or webpage by headings, links, or other section types. The rotor can also bring up the braille keyboard. You can help VoiceOver users navigate among related items in your app by identifying these items to the rotor. For developer guidance, see `UIAccessibilityCustomRotor` and `NSAccessibilityCustomRotor`.

In iPadOS, macOS, and visionOS, make sure people can use the keyboard to navigate and interact with all components in your app. Ideally, people can turn on Full Keyboard Access and perform every task in your experience using only the keyboard. In addition to accessibility keyboard shortcuts, the system defines a large number of other keyboard shortcuts that many people use all the time. To support everyone, it's important to avoid overriding any system-defined keyboard shortcuts in your app. For guidance, see Keyboards.

Text display

In iOS, iPadOS, tvOS, visionOS, and watchOS, use Dynamic Type and test that your app's layout adapts to all font sizes. Dynamic Type lets people pick the font size that works for them. Verify that your design can scale and that both text and glyphs are legible at all font sizes. On iPhone or iPad, for example, turn on Larger Accessibility Text Sizes in Settings > Accessibility > Display & Text Size > Larger Text, and make sure your app remains comfortably readable. You can download the Dynamic Type size tables in Apple Design Resources for each platform.

As font size increases, keep text truncation to a minimum. In general, aim to display as much useful text in the largest accessibility font size as you do in the largest standard font size. Avoid truncating text in scrollable regions unless people can open a separate view to read the rest of the content. You can prevent text truncation in a label by configuring it to use as many lines as needed to display a useful amount of text; for developer guidance, see `numberOfLines`.

Consider adjusting layout at large font sizes. When font size increases in a horizontally constrained context, inline items and container boundaries can crowd text, making it less readable. For example, if you display text inline with secondary items — such as glyphs or timestamps — the text has less horizontal space. At large font sizes, an inline layout might cause text to truncate or result in overlapping text and secondary items. In this case, consider using a stacked layout where the text appears above the secondary items. Similarly, multiple columns of text can become less readable at large font sizes because each column constrains horizontal space. In this case, consider reducing the number of columns when font size increases to avoid text truncation and improve overall readability. For developer guidance, see `isAccessibilityCategory`.

At smaller text sizes, Mail displays the date inline with the sender's name.

At the largest accessibility text size, Mail displays the date below the recipient's name.

Increase the size of meaningful interface icons as font size increases. If you use interface icons to communicate important information, make sure they are easy to view at larger font sizes, too. When you use SF Symbols, you get icons that scale automatically with Dynamic Type size changes.

Maintain a consistent information hierarchy regardless of the current font size. For example, keep primary elements toward the top of a view even when the font size is very large, so that people don't lose track of these elements.

Prefer regular or heavy font weights in your app. Consider using Regular, Medium, Semibold, or Bold font weights, because they are easier to see. Avoid Ultralight, Thin, and Light font weights, which can be more difficult to see.

Ensure your app responds correctly and looks good when people turn on bold text. In iOS, iPadOS, tvOS, visionOS, and watchOS, people turn on the bold text accessibility setting to make text and symbols easier to see. In response, your app needs to make all text bolder and give all glyphs an increased stroke weight. The system fonts and SF symbols automatically adjust to the bold text accessibility setting.

Make sure custom fonts are legible. Custom typefaces can sometimes be difficult 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, it's usually best to use the system fonts. If you do use a custom font, make sure it's easy to read, even at small sizes.

Avoid full text justification. The whitespace created by fully justified text can create patterns that make it difficult for many people to read and focus on the text. Left justification — or right justification in right-to-left languages — provides a framing reference for people with learning and literacy challenges, such as dyslexia.

Avoid using italics or all caps for long passages of text. Italics and all caps are great for occasional emphasis, but overuse of these styles makes text hard to read.

Color and effects

Don't rely solely on color to differentiate between objects or communicate important information. If you use color to convey information, be sure to provide text labels or glyph shapes to help everyone perceive it.

Prefer system colors for text. When you use system colors in text, it responds correctly to accessibility settings such as Invert Colors and Increase Contrast.

Avoid using color combinations as the only way to distinguish between two states or values. Many colorblind people find it difficult to distinguish blue from orange; other problematic combinations are red and green, red and black, and either red or green combined with gray. When it makes sense to use a combination of colors to communicate states or values, include additional visual indicators so everyone can perceive the information. For example, instead of using red and green circles to indicate offline and online, you could use a red square and a green circle. Some image-editing software includes tools that can help you proof for colorblindness.

As seen without colorblindness.

As seen with red-green colorblindness.

Ensure your views respond correctly to Invert Colors. People can turn on Invert Colors when they prefer to view items on a dark background. In the Smart Invert mode of Invert Colors, images, video, and full-color icons (such as app icons and nontemplate images) don't invert, and dark UI stays dark. Test your app or game to find places where you might need to prevent an image — like a photo in a custom view — from inverting.

Use strongly contrasting colors to improve readability. Many factors affect the perception of color, including font size and weight, color brightness, screen resolution, and lighting conditions. When you increase color contrast of visual elements like text, glyphs, and controls, you can help more people use your app in more situations. To find out if the contrast of adjacent colors in your UI meets minimum acceptable levels, you can use Xcode's Accessibility Inspector or an online color calculator based on the Web Content Accessibility Guidelines (WCAG) color contrast formula. In general, smaller or lighter-weight text needs to have greater contrast to be legible. Use the following values for guidance.

Text size	Text weight	Minimum contrast ratio
Up to 17 points	All	4.5:1
18 points and larger	All	3:1
All	Bold	3:1

Change blurring and transparency when people turn on Reduce Transparency. For example, make areas of blurred content and translucency mostly opaque. For best results, use a color value in the opaque area that's different from the original color value you used when the area was blurred or translucent.

Motion

Avoid requiring animations unless they're essential for your experience. In general, let people use your app without relying on any animations.

Play tightened animations when Reduce Motion is on. People can turn on Reduce Motion if they tend to get distracted or experience dizziness or nausea when viewing animations that include effects such as zooming, scaling, spinning, or peripheral motion. In response to this setting, you need to turn off or reduce animations that are known to cause problems (to learn more, see Responsive design for motion). If you use a problematic animation to communicate important information, consider designing a non animated alternative or tightening the physics of the animation to reduce its motion. For example:

- Tighten springs to reduce bounce effects or track 1:1 as a person gestures
- Avoid animating depth changes in z-axis layers
- Avoid animating into or out of blurs
- Replace a slide with a fade to avoid motion

Let people control video and other motion effects. Avoid autoplaying video or effects without also providing a button or other way to control them.

Be cautious when displaying moving or blinking elements. Although subtle movement and blinking can draw people's attention, these effects can also be distracting and they aren't useful for people with visual disabilities. Worse, some blinking elements can cause epileptic episodes. In all cases, avoid using movement and blinking as the only way to convey information.

For additional guidance on helping people remain comfortable while they experience motion in your visionOS app, see Motion > visionOS. For developer guidance, see Improving accessibility support in your visionOS app.

Platform considerations

No additional considerations for iOS, iPadOS, macOS, tvOS, or watchOS.

visionOS

Avoid anchoring content to the wearer's head. In addition to making people feel stuck or confined, anchoring content to their head can prevent someone from using Pointer Control to interact with that content. Head-anchored content can also prevent people with low vision from reading it because they can't move closer to it or position the content inside the Zoom lens.

Pointer control - hand | Pointer control - head | Zoom lens

Play

Resources

Related

Inclusion

Developer documentation

Accessibility for developers

Accessibility

Accessibility modifiers — SwiftUI

Accessibility for UIKit — UIKit

Accessibility for AppKit — AppKit

Videos

Create accessible spatial experiences

Design considerations for vision and motion

The practice of inclusive design

Change log

Date	Changes
June 10, 2024	Added a link to Apple's Unity plug-ins for supporting Dynamic Type.
December 5, 2023	Updated visionOS Zoom lens artwork.
June 21, 2023	Updated to include guidance for visionOS.

App icons

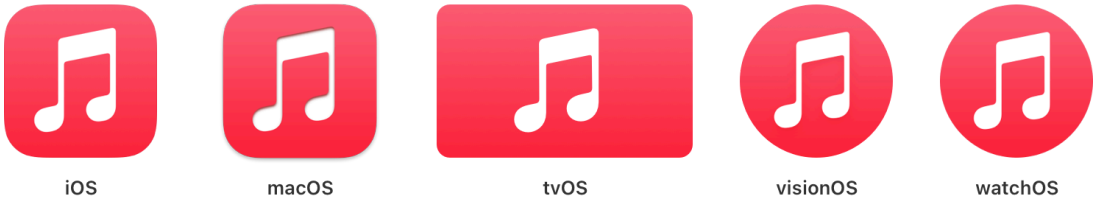
A unique, memorable icon communicates the purpose and personality of your app or game and can help people recognize your product at a glance in the App Store and on their devices.

Beautiful app icons are an important part of the user experience on all Apple platforms and every app and game must have one. Each platform defines a slightly different style for app icons, so create a design that adapts well to different shapes and levels of detail while maintaining strong visual consistency and messaging. To download templates that help you create icons for each platform, see Apple Design Resources. For guidance on creating other types of icons, see Icons.

Best practices

Embrace simplicity. Simple icons tend to be easier for people to understand and recognize. Find a concept or element that captures the essence of your app or game, make it the core idea of the icon, and express it in a simple, unique way. Avoid adding too many details, because they can be hard to discern and can make an icon appear muddy, especially at smaller sizes. Prefer a simple background that puts the emphasis on the primary image — you don't need to fill the entire icon with content.

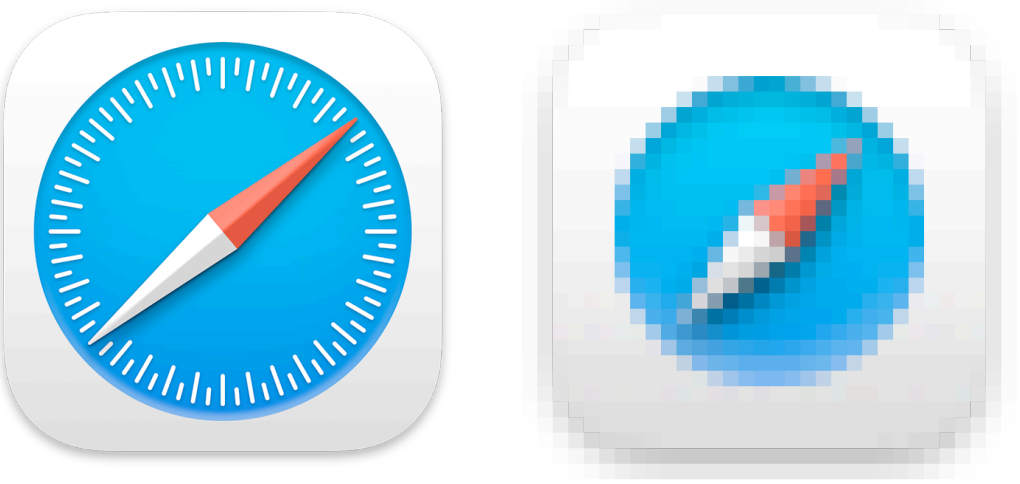
Create a design that works well on multiple platforms so it feels at home on each. If your app or game runs on more than one platform, use similar images and color palettes in all icons while rendering them in the style that's appropriate for each platform. For example, in iOS, tvOS, and watchOS, the Music app icon depicts the white musical notes on a red background using a streamlined, graphical style; macOS displays the same elements, while adding shadow that makes the notes look recessed. Similarly, the Music app icon in visionOS uses the same color scheme and content, but offers a true 3D appearance when viewed while wearing the device.



Prefer including text only when it's an essential part of your experience or brand. Text in icons is often too small to read easily, can make an icon appear cluttered, and doesn't support accessibility or localization. In some contexts, the app name appears near the icon, making it redundant to display the name within it. Although using a mnemonic like the first letter of your app's name can help people recognize your app or game, avoid including nonessential words that tell people what to do with it — like "Watch" or "Play" — or context-specific terms like "New" or "For visionOS."

Prefer graphical images to photos and avoid replicating UI components in your icon. Photos are full of details that don't work well when viewed at small sizes. Instead of using a photo, create a graphic representation of the content that emphasizes the features you want people to notice. Similarly, if your app has an interface that people recognize, don't just replicate standard UI components or use app screenshots in your icon.

If needed, optimize your icon for the specific sizes the system displays in places like Spotlight search results, Settings, and notifications. For iOS, iPadOS, and watchOS, you can tell Xcode to generate all sizes from your 1024×1024 px App Store icon, or you can provide assets for some or all individual icon sizes. For macOS and tvOS, you need to supply all sizes; for visionOS, you supply a single 1024x1024 px asset. If you create your own versions of your app icon, make sure the image remains distinct at all sizes. For example, you might remove fine details and unnecessary features, simplifying the image and exaggerating primary features. If you need to make such changes, keep them subtle so that your app icon remains visually consistent in every context.



The 512x512 px Safari app icon (on the left) uses a circle of tick marks to indicate degrees; the 16x16 px version of the icon (on the right) doesn't include this detail.

Design your icon as a square image. On most platforms, the system applies a mask that automatically adjusts icon corners to match the platform's aesthetic. For example, visionOS and watchOS automatically apply a circular mask. Although the system applies the rounded rectangle appearance to the icon of an app created with Mac Catalyst, you need to create your macOS app icon in the correct rounded shape; for guidance, see macOS.

In most cases, design your icon with full edge-to-edge opacity. For layered app icons in visionOS and tvOS, prefer fully opaque content on the bottom layer. Note that the dark variants of iOS and iPadOS icons omit a solid background because the system provides one automatically.

For downloadable production templates that help you create app icons for each platform, see Apple Design Resources.

Consider offering an alternate app icon. In iOS, iPadOS, and tvOS, and iPadOS and iOS apps running in visionOS, people can choose an alternate version of an icon, which can strengthen their connection with the app or game and enhance their experience. For example, a sports app might offer different icons for different teams. Make sure that each alternate app icon you design remains closely related to your content and experience; avoid creating a version that people might mistake for the icon of a different app. When people want to switch to an alternate icon, they can visit your app's settings.

> Note
>
> Alternate app icons in iOS and iPadOS require their own dark and tinted variants. As with the default app icon, all alternate and variant icons are also subject to app review and must adhere to the App Review Guidelines.

Don't use replicas of Apple hardware products. Apple products are copyrighted and can't be reproduced in your app icons.

Platform considerations

iOS, iPadOS

People can customize the appearance of their app icons to be *light*, *dark*, or *tinted*. You can create your own variations to ensure that each one looks exactly the way you want. See Apple Design Resources for icon templates.

Design your dark and tinted icons to feel at home next to system app icons and widgets. You can preserve the color palette of your default icon, but be mindful that dark icons are more subdued, and tinted icons are even more so. A great app icon is visible, legible, and recognizable, even with a different tint and background.

Consider a simplified version of your icon that captures its essential features. Because dark and tinted icons appear against a dark background, fine details tend to stand out more and can look messy or cluttered.

Use your light app icon as a basis for your dark icon. Choose complementary colors that reflect the default design, and avoid excessively bright images. For guidance, see Dark Mode colors. To look at home on the platform, omit the background so the system-provided background can show through.

You provide a transparent dark icon.

The system provides the gradient background.

The system composites your dark icon on the gradient background.

Provide your tinted icon as a grayscale image. Most app icons look great with a vertical gradient applied uniformly over the icon image.

You provide a fully opaque, grayscale icon.

The system provides the gradient background.

The system generates a tinted icon, compositing your grayscale icon on the gradient background.

In some cases, you might want to vary the opacity in other ways; for example, the Home app icon uses varying shades of gray on concentric house shapes to create contrast between the elements of the icon.

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.

macOS

In macOS, app icons share a common set of visual attributes, including a rounded-rectangle shape, front-facing perspective, level position, and uniform drop shadow. Rooted in the macOS design language, these attributes showcase the lifelike rendering style people expect in macOS while presenting a harmonious user experience.

Consider depicting a familiar tool to communicate what people use your app to do. To give context to your app's purpose, you can use the icon background to portray the tool's environment or the items it affects. For example, the TextEdit icon pairs a mechanical pencil with a sheet of lined paper to suggest a utilitarian writing experience. After you create a detailed, realistic image of a tool, it often works well to let it float just above the background and extend slightly past the icon boundaries. If you do this, make sure the tool remains visually unified with the background and doesn't overwhelm the rounded-rectangle shape.

If you depict real objects in your app icon, make them look like they're made of physical materials and have actual mass. Consider replicating the characteristics of substances like fabric, glass, paper, and metal to convey an object's weight and feel. For example, the Xcode app icon features a hammer that looks like it has a steel head and polymer grip.

Use the drop shadow in the icon design template. The macOS app icon template includes the system-defined drop shadow that helps your app icon coordinate with other macOS icons.

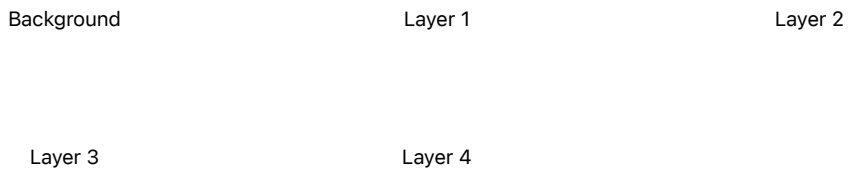
Consider using interior shadows and highlights to add definition and realism. For example, the Mail app icon uses both shadows and highlights to give the envelope authenticity and to suggest that the flap is slightly open. In icons that include a tool that floats above a background — such as TextEdit or Xcode — interior shadows can strengthen the perception of depth and make the tool look real. Use shadows and highlights that suggest a light source facing the icon, positioned just above center and tilted slightly downward.

Avoid defining contours that suggest a shape other than a rounded rectangle. In rare cases, you might want to fine-tune the basic app icon shape, but doing so risks creating an icon that looks like it doesn't belong in macOS. If you must alter the shape, prefer subtle adjustments that continue to express a rounded rectangle silhouette.

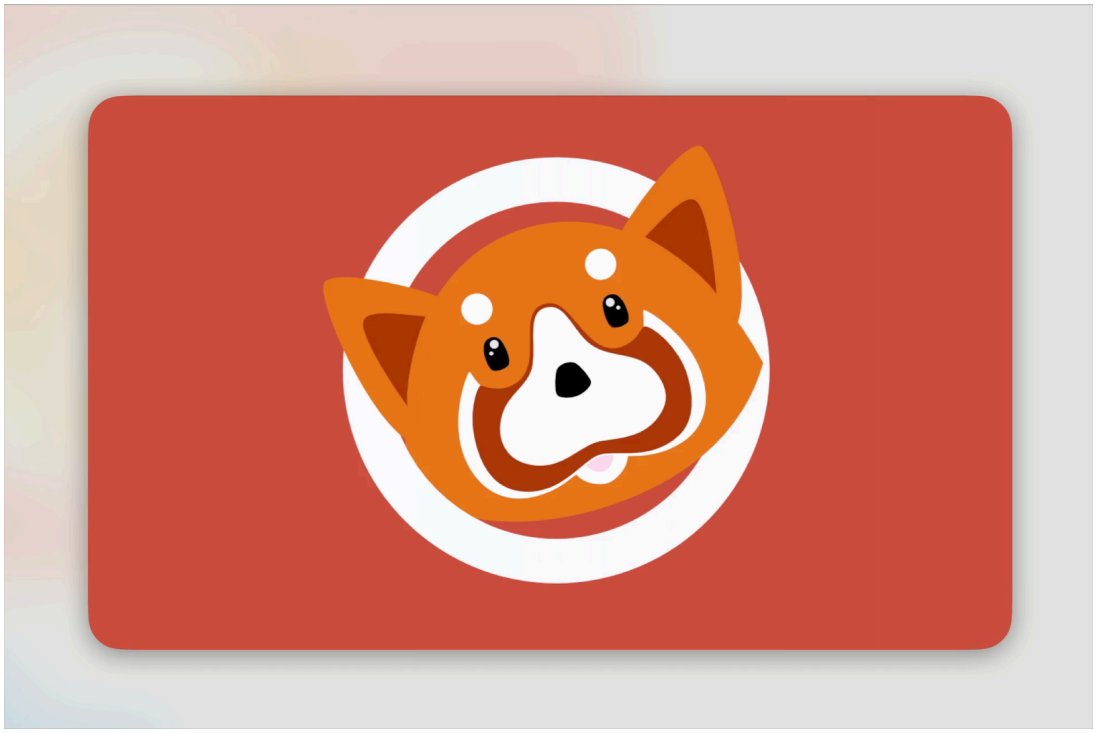
Keep primary content within the icon grid bounding box; keep all content within the outer bounding box. If an icon's primary content extends beyond the icon grid bounding box, it tends to look out of place. If you overlay a tool on your icon, it works well to align the tool's top edge with the outer bounding box and its bottom edge with the inner bounding box, as shown below. You can use the grid to help you position items within an icon and to ensure that centered inner elements like circles use a size that's consistent with other icons in the system.

tvOS

tvOS app icons use between two and five layers to create a sense of dynamism as people bring them into focus. For guidance, see Layered images.



When focused, the app icon elevates to the foreground and gently sways while the surface illuminates. The separation between layers and the use of transparency produce a feeling of depth during the parallax effect.



Play

Use appropriate layer separation. If your icon includes a logo, separate the logo from the background. If your icon includes text, bring the text to the front so it's not hidden by other layers when the parallax effect occurs.

Play

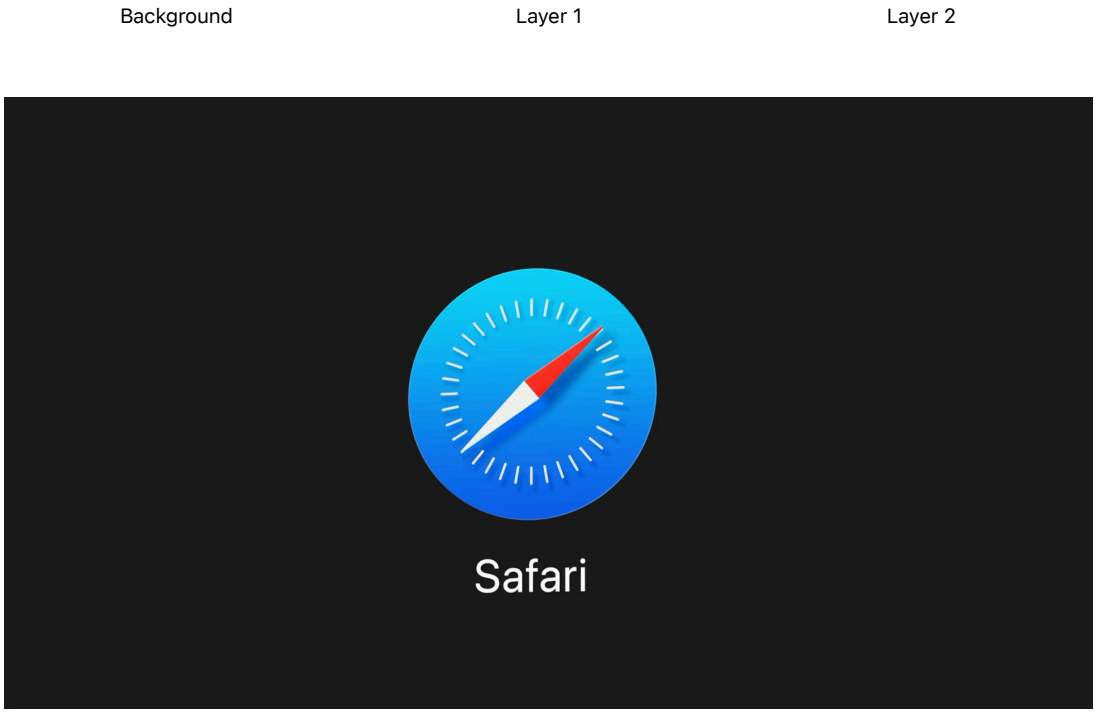
Use gradients and shadows cautiously. Background gradients and vignettes can clash with the parallax effect. For gradients, prefer top-to-bottom, light-to-dark styles. Shadows usually look best as sharp, hard-edged tints that are baked into the background layer and aren't visible when the app icon is stationary.

Leverage varying opacity levels to increase the sense of depth and liveliness. Creative use of opacity can make your icon stand out. For example, the Photos icon separates its centerpiece into multiple layers that contain translucent pieces, bringing greater liveliness to the design.

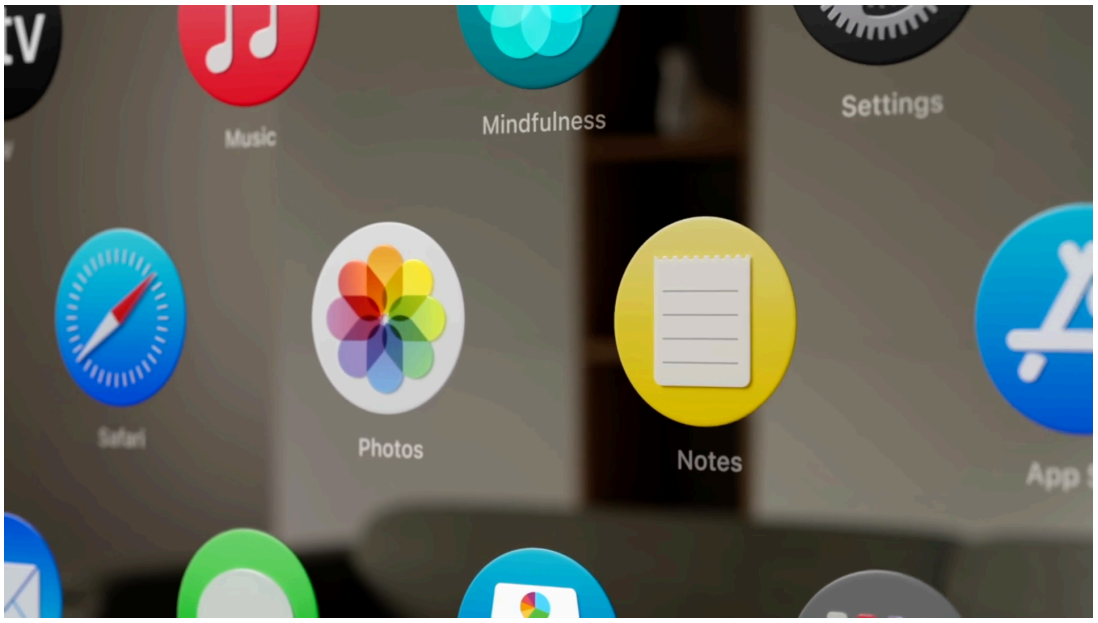
Include a safe zone to ensure content isn't cropped. When focused, content around the edges of your app icon may be cropped as the icon scales and moves. To ensure that your icon's content is always visible, keep a safe zone around it. Be aware that the safe zone can vary, depending on the image size, layer depth, and motion, and that foreground layers are cropped more than background layers.

visionOS

A visionOS app icon is circular and includes a background layer and one or two layers on top, producing a three-dimensional object that subtly expands when people view it.



The system enhances an app icon's visual dimensionality by adding shadows to convey a sense of depth between layers and using the alpha channel of the upper layers to create an embossed appearance.



Play

Use a full-bleed, non-transparent image for the background layer of your icon. In contrast, avoid using full-bleed images in non-background layers. Using transparent areas in non-background layers lets visual information from underlying layers show through.

Provide each layer as a square image. The system uses a circular mask to crop all layers of an app icon. Providing layers that are already cropped can negatively impact the result.

Avoid using large areas of semi transparency. Although using semi-transparent pixels to anti-alias a shape works fine, a large semi-transparent area doesn't blend well with alpha and can combine with the system-provided shadow to produce a dark result. Unless you're anti-aliasing a shape, keep pixels fully opaque or transparent.

In non-background layers, prefer well-defined edges between distinct regions that are either fully opaque or transparent pixels. The system-drawn highlights and shadows look best when non-background layers contain shapes that have clearly defined edges. Avoid using soft or feathered edges.

Avoid adding a shape that's intended to look like a hole or concave area to the background layer. The system-added shadow and specular highlights can make such a shape stand out instead of recede.

Keep distinct shapes or images in non-background layers close to the center. The circular mask can clip a shape or image when it's too close to the edge, causing the shape to look off-center and spoiling the icon's three-dimensional appearance.

Avoid visual elements that give the appearance of depth from a fixed vantage point. If people can perceive the depth of a layer's inner element from only one perspective, this depth disappears when they look at the icon. Avoid using a technique like extruding the bottom edge of a layer's element, because doing so conflicts with the perpendicular perspective of other app icons.

Avoid adding custom specular highlights or shadows to your visionOS app icon. In addition to interfering with the system-provided visual effects, custom highlights and shadows are static whereas visionOS supplies dynamic ones.

For developer guidance, see Configuring your app icon.

watchOS

A watchOS app icon is circular and displays no accompanying text.

Avoid using black for your icon's background. Lighten a black background or add a border so the icon doesn't blend into the display background.

Specifications

App icon attributes

App icons in all platforms use the PNG format and support the following color spaces:

- sRGB (color)
- Gray Gamma 2.2 (grayscale)

In addition, app icons in iOS, iPadOS, macOS, tvOS, and watchOS support Display P3 (wide-gamut color).

The layers, transparency, and corner radius of an app icon can vary per platform. Specifically:

Platform	Layers	Transparency	Asset shape
iOS, iPadOS	Single	No	Square
macOS	Single	Yes, as appropriate	Square with rounded corners
tvOS	Multiple	No	Rectangle
visionOS	Multiple	Yes, as appropriate	Square
watchOS	Single	No	Square

App icon sizes

iOS, iPadOS app icon sizes

For the App Store, create an app icon that measures 1024x1024 px.

You can let the system automatically scale down your 1024x1024 px app icon to produce all other sizes, or — if you want to customize the appearance of the icon at specific sizes — you can supply multiple versions such as the following.

@2x (pixels)	@3x (pixels) iPhone only	Usage
120x120	180x180	Home Screen on iPhone
167x167	–	Home Screen on iPad Pro
152x152	–	Home Screen on iPad, iPad mini
80x80	120x120	Spotlight on iPhone, iPad Pro, iPad, iPad mini
58x58	87x87	Settings on iPhone, iPad Pro, iPad, iPad mini
76x76	114x114	Notifications on iPhone, iPad Pro, iPad, iPad mini

macOS app icon sizes

For the App Store, create an app icon that measures 1024x1024 px.

In addition to the App Store version, you also need to supply your app icon in the following sizes.

@1x (pixels)	@2x (pixels)
512x512	1024x1024
256x256	512x512
128x128	256x256
32x32	64x64
16x16	32x32

tvOS app icon sizes

For the App Store, create an app icon that measures 1280x768 px.

In addition to the 1280x768 px version of your app icon, you also need to supply the following sizes.

@1x (pixels)	@2x (pixels)	Usage
400x240	800x480	Home Screen

visionOS app icon sizes

For the App Store and the Home View, create an app icon that measures 1024x1024 px.

watchOS app icon sizes

For the App Store, create an app icon that measures 1024x1024 px.

You can let the system automatically scale down your 1024x1024 px app icon to all other sizes, or — if you want to customize the appearance of your icon at specific sizes — you can supply the sizes listed in the following table. All icon dimensions are shown in pixels @2x.

38mm	40mm	41mm	42mm	44mm	45mm	4
80x80	88x88	92x92	80x80	100x100	102x102	1(
48x48	55x55	58x58	55x55	58x58	66x66	6
172x172	196x196	196x196	196x196	216x216	234x234	2

If you have a companion iPhone app, you also need to supply your watchOS app icon in the following sizes.

@2x (pixels)	@3x (pixels)
58x58	87x87

Resources

Related

Apple Design Resources

Developer documentation

Configuring your app icon — Xcode

Videos

App Icon Design

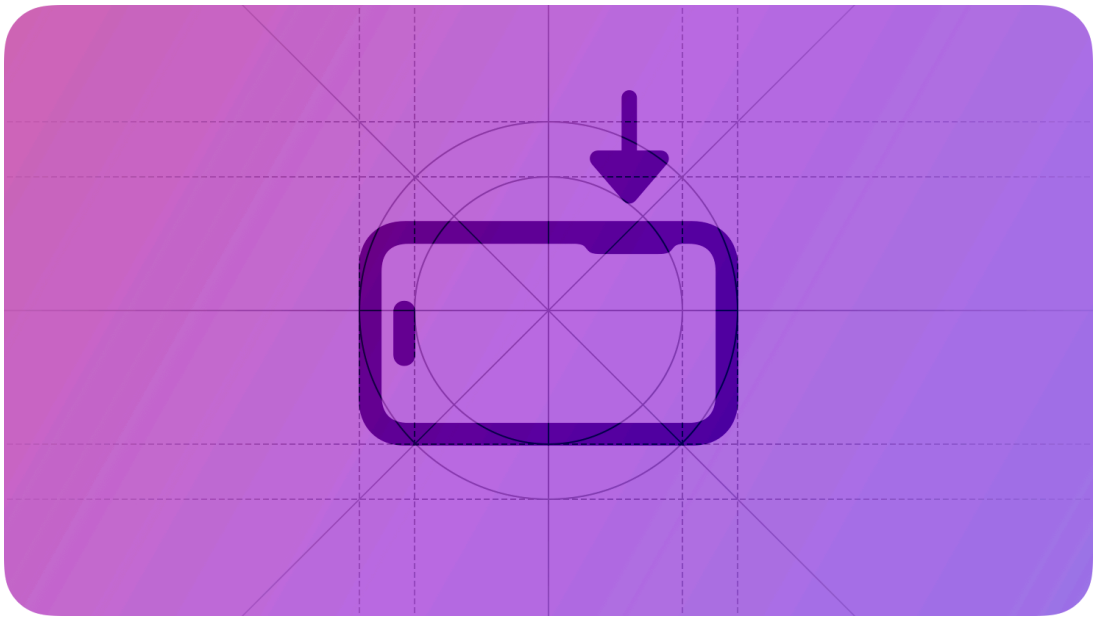
Change log

Date	Changes
June 10, 2024	Added guidance for creating dark and tinted app icon variants for iOS and iPadOS.
January 31, 2024	Clarified platform availability for alternate app icons.
June 21, 2023	Updated to include guidance for visionOS.
September 14, 2022	Added specifications for Apple Watch Ultra.

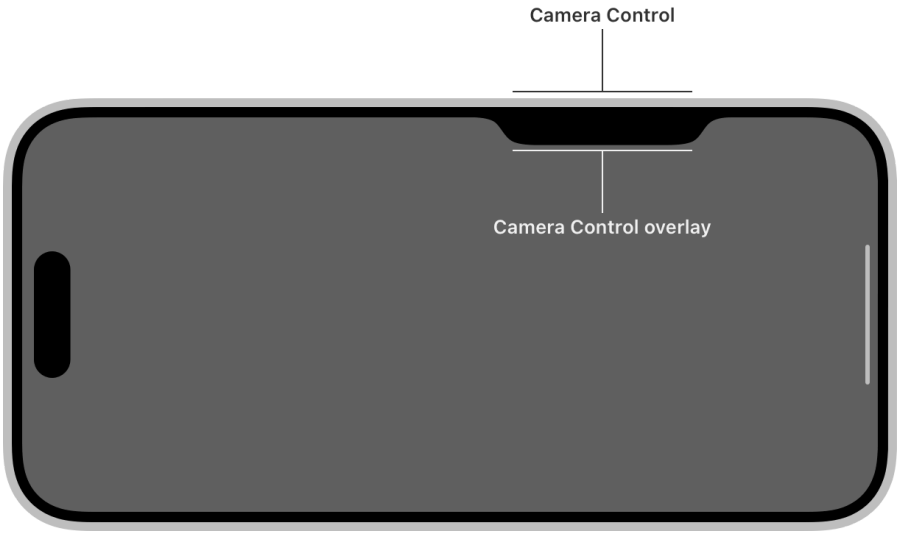
Camera Control

The Camera Control provides direct access to your app's camera experience.

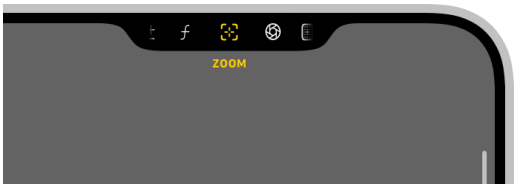
Supported platforms



On iPhone 16 and iPhone 16 Pro models, the Camera Control quickly opens your app's camera experience to capture moments as they happen. When a person lightly presses the Camera Control, the system displays an overlay that extends from the device bezel.



The overlay allows people to quickly adjust controls. A person can view the available controls by lightly double-pressing the Camera Control. After selecting a control, they can slide their finger on the Camera Control to adjust a value to capture their content as they want.

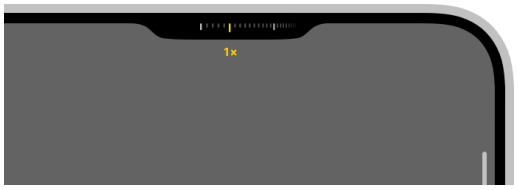


Controls in the overlay

Anatomy

The Camera Control offers two types of controls for adjusting values or changing between options:

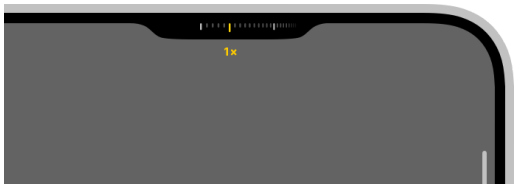
- A *slider* provides a range of values to choose from, such as how much contrast to apply to the content.
- A *picker* offers discrete options, such as turning a grid on and off in the viewfinder.



Slider control

Picker control

In addition to custom controls that you create, the system provides a set of standard controls that you can optionally include in the overlay to allow someone to adjust their camera's zoom and exposure.

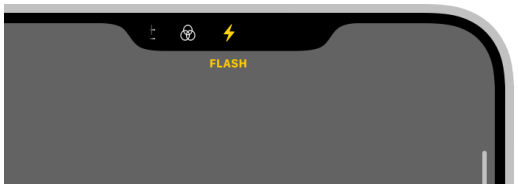


Zoom factor control

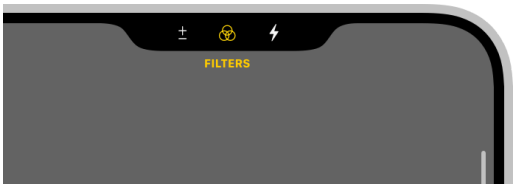
Exposure bias control

Best practices

Use SF Symbols to represent control functionality. The system doesn't support custom symbols; instead, pick a symbol from SF Symbols that clearly denotes a control's behavior. iOS offers thousands of symbols you can use to represent the controls your app shows in the overlay. Symbols for controls don't represent their current state. To view available symbols, see the Camera & Photos section in the SF Symbols app.



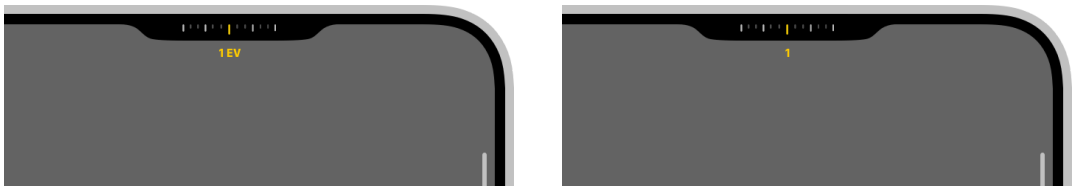
The `bolt.fill` symbol that represents a control for the camera flash



The `camera.filters` symbol that represents a control for filters

Keep names of controls short. Control labels adhere to Dynamic Type sizes, and longer names may obfuscate the camera's viewfinder.

Include units or symbols with slider control values to provide context. Providing descriptive information in the overlay, such as EV, %, or a custom string, helps people understand what the slider controls. For developer guidance, see `localizedValueFormat`.



Value with context

Value without context

Define prominent values for a slider control. Prominent values are ones people choose most frequently, or values that are evenly spaced, like the major increments of zoom factor. When a person slides on the Camera Control to adjust a slider control, the system more easily lands on prominent values you define. For developer guidance, see `prominentValues`.

Make space for the overlay in the viewfinder. The overlay and control labels occupy the screen area adjacent to the Camera Control in both portrait and landscape orientations. To avoid overlapping the interface elements of your camera capture experience, place your UI outside of the overlay areas. Maximize the height and width of the viewfinder and allow the overlay to appear and disappear over it.

Minimize distractions in the viewfinder. When capturing a photo or video, people appreciate a large preview image with as few visual distractions as possible. Avoid duplicating controls, like sliders and toggles, in your UI and the overlay when the system displays the overlay.

Keep UI minimal.

Avoid showing controls in the viewfinder that people access in the overlay.

Enable or disable controls depending on the camera mode. For example, disable video controls when taking photos. The overlay supports multiple controls, but you can't remove or add controls at runtime.

Consider how to arrange your controls. Order commonly used controls toward the middle to allow quick access, and include lesser used controls on either side. When a person lightly presses the Camera Control to open the overlay again, the system remembers the last control they used in your app.

Allow people to use the Camera Control to launch your experience from anywhere. Create a locked camera capture extension that lets people configure the Camera Control to launch your app's camera experience from their locked device, the Home Screen, or from within other apps. For guidance, see Camera experiences on a locked device.

Platform considerations

Not supported in iPadOS, macOS, watchOS, tvOS, or visionOS.

Resources

Related

SF Symbols

Controls

Developer documentation

`AVCaptureControl` — AVFoundation

LockedCameraCapture

Change log

Date	Changes
September 9, 2024	New page.

Components

Learn how to use and customize system-defined components to give people a familiar and consistent experience.

Content

Layout and organization

Menus and actions

Navigation and search

Presentation

Selection and input

Status

System experiences

Controls

In iOS and iPadOS, a control provides quick access to a feature of your app from Control Center, the Lock Screen, or the Action button.

Supported platforms

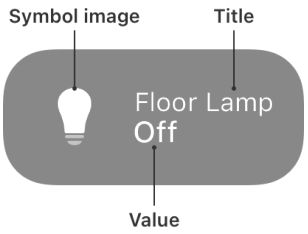
Starting in iOS 18 and iPadOS 18, a control is a button or toggle that provides quick access to your app's features from other areas of the system.

Control buttons perform an action, link to a specific area of your app, or launch a camera experience on a locked device. Control toggles switch between two states, such as on and off.

People can add controls to Control Center by pressing and holding in an empty area of Control Center, to the Lock Screen by customizing their Lock Screen, and to the Action button by configuring the Action button in the Settings app.

Anatomy

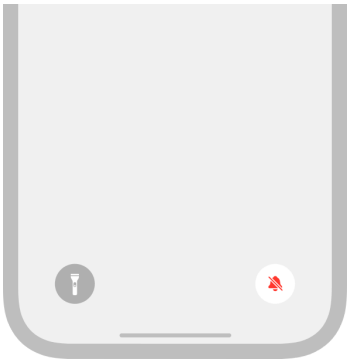
Controls contain a symbol image, a title, and, optionally, a value. The symbol visually represents what the control does and can be a symbol from SF Symbols or a custom symbol. The title describes what the control relates to, and the value represents the state of the control. For example, the title can display the name of a light in a room, while the value can display whether it's on or off.



Controls display their information differently depending on where they appear:

- In Control Center, a control displays its symbol and, at larger sizes, its title and value.
- On the Lock Screen, a control displays its symbol.
- On iPhone devices with a control assigned to the Action button, pressing and holding it displays the control's symbol in the Dynamic Island, as well as its value (if present).

Control toggle in Control Center



Control toggle on the Lock Screen

Control toggle in the Dynamic Island performed from the Action button

Best practices

Offer controls for actions that provide the most benefit without having to launch your app. For example, launching a Live Activity from a control creates an easy and seamless experience that informs someone about progress without having to navigate to your app to stay up to date. For guidance, see Live Activities.

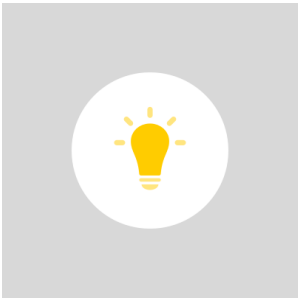
Update controls when someone interacts with them, when an action completes, or remotely with a push notification. Update the contents of a control to accurately reflect the state and show if an action is still in progress.

Choose a descriptive symbol that suggests the behavior of the control. Depending on where a person adds a control, it may not display the title and value, so the symbol needs to convey enough information about the control's action. For control toggles, provide a symbol for both the on and off states. For example, use the SF Symbols `door.garage.open` and `door.garage.closed` to represent a control that opens and closes a garage door. For guidance, see SF Symbols.

Use symbol animations to highlight state changes. For control toggles, animate the transition between both on and off states. For control buttons with actions that have a duration, animate indefinitely while the action performs and stop animating when the action is complete. For developer guidance, see Symbols and `SymbolEffect`.

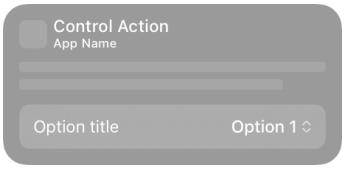
Select a tint color that works with your app's brand. The system applies this tint color to a control toggle's symbol in its on state. When a person performs the action of a control from the Action button, the system also uses this tint color to display the value and symbol in the Dynamic Island. For guidance, see Branding.

Nontinted control toggle in the off state

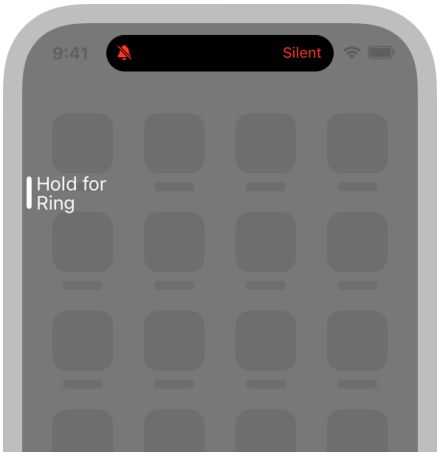


Tinted control toggle in the on state

Help people provide additional information the system needs to perform an action. A person may need to configure a control to perform a desired action — for example, select a specific light in a house to turn on and off. If a control requires configuration, prompt people to complete this step when they first add it. People can reconfigure the control at any time. For developer guidance, see `promptsForUserConfiguration()`.

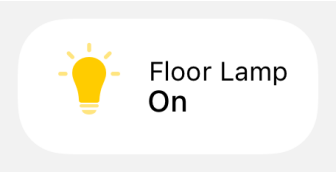


Provide hint text for the Action button. When a person presses the Action button, the system displays hint text to help them understand what happens when they press and hold. When someone presses and holds the Action button, the system performs the action configured to it. Use verbs to construct the hint text. For developer guidance, see `controlWidgetActionHint(_:)`.



If your control title or value can vary, include a placeholder. Placeholder information tells people what your control does when the title and value are situational. The system displays this information when someone brings up the controls gallery in Control Center or the Lock Screen and chooses your control, or before they assign it to the Action button.

Hide sensitive information when the device is locked. When the device is locked, consider having the system redact the title and value to hide personal or security-related information. Specify if the system needs to redact the symbol state as well. If specified, the system redacts the title and value, and displays the symbol in its off state.



Control toggle with no information hidden

Control toggle with information hidden on a locked device

Require authentication for actions that affect security. For example, require people to unlock their device to access controls to lock or unlock the door to their house or start their car. For developer guidance, see `IntentAuthenticationPolicy`.

Camera experiences on a locked device

If your app supports camera capture, starting with iOS 18 you can create a control that launches directly to your app's camera experience while the device is locked. For any task beyond capture, a person must authenticate and unlock their device to complete the task in your app. For developer guidance, see LockedCameraCapture.

Use the same camera UI in your app and your camera experience. Sharing UI leverages people's familiarity with the app. By using the same UI, the transition to the app is seamless when someone captures content and taps a button to perform additional tasks, such as posting to a social network or editing a photo.

Provide instructions for adding the control. Help people understand how to add the control that launches this camera experience.

Platform considerations

No additional considerations for iOS or iPadOS. Not supported in macOS, watchOS, tvOS, or visionOS.

Resources

Related

Widgets

Action button

Developer documentation

LockedCameraCapture

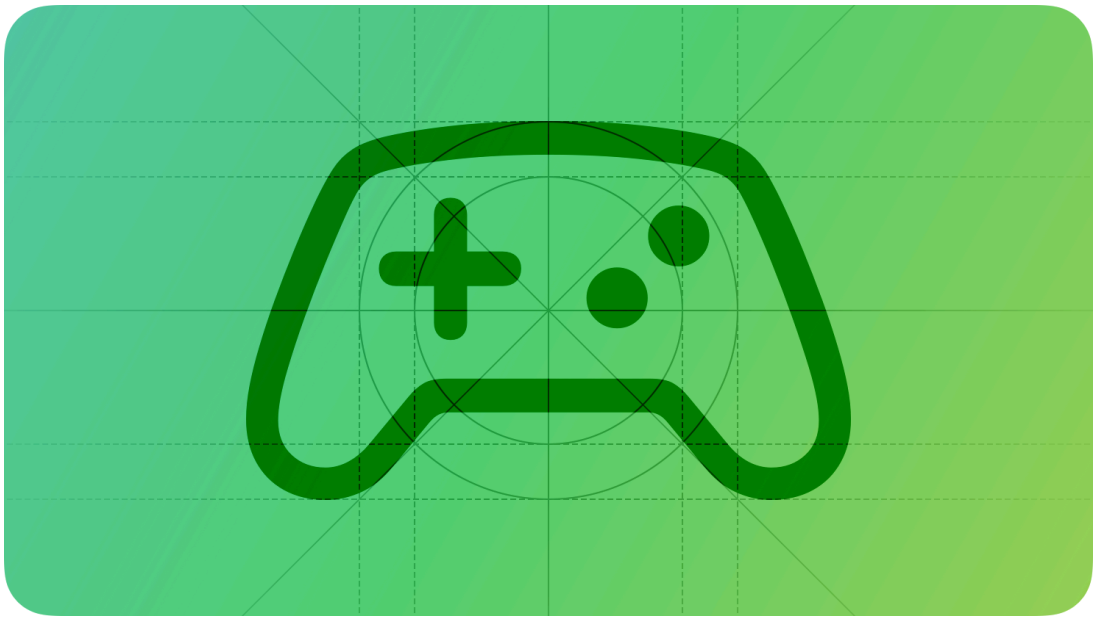
WidgetKit

Change log

Date	Changes
June 10, 2024	New page.

Designing for games

When people play your game on an Apple device, they dive into the world you designed while relying on the platform features they love.



As you create or adapt a game for Apple platforms, learn how to integrate the fundamental platform characteristics and patterns that help your game feel at home on all Apple devices. To learn what makes each platform unique, see Designing for iOS, Designing for iPadOS, Designing for macOS, Designing for tvOS, Designing for visionOS, and Designing for watchOS. For developer guidance, see Games Pathway.

Jump into gameplay

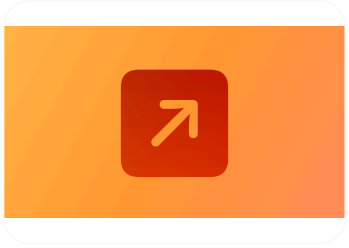
Let people play as soon as installation completes. You don't want a player's first experience with your game to be waiting for a lengthy download. Include as much playable content as you can in your game's initial installation while keeping the download time to 30 minutes or less. Download additional content in the background. For guidance, see Loading.

Provide great default settings. People appreciate being able to start playing without first having to change a lot of settings. Use information about a player's device to choose the best defaults for your game, such as the device resolution that makes your graphics look great, automatic recognition of paired accessories and game controllers, and the player's accessibility settings. Also, make sure your game supports the platform's most common interaction methods. For guidance, see Settings.

Teach through play. Players often learn better when they discover new information and mechanics in the context of your game's world, so it can work well to integrate configuration and onboarding flows into a playable tutorial that engages people quickly and helps them feel

successful right away. If you also have a written tutorial, consider offering it as a resource players can refer to when they have questions instead of making it a prerequisite for gameplay. For guidance, see Onboarding.

Defer requests until the right time. You don't want to bombard people with too many requests before they start playing, but if your game uses certain sensors on an Apple device or personalizes gameplay by accessing data like hand-tracking, you must first get the player's permission (for guidance, see Privacy). To help people understand why you're making such a request, integrate it into the scenario that requires the data. For example, you could ask permission to track a player's hands between an initial cutscene and the first time they can use their hands to control the action. Also, make sure people spend quality time with your game before you ask them for a rating or review (for guidance, see Ratings and reviews).



Launching

Onboarding

Loading

Look stunning on every display

Make sure text is always legible. When game text is hard to read, people can struggle to follow the narrative, understand important instructions and information, and stay engaged in the experience. To keep text comfortably legible on each device, ensure that it contrasts well with the background and uses at least the recommended minimum text size in each platform. For guidance, see Typography; for developer guidance, see Adapting your game interface for smaller screens.

Platform	Default text size	Minimum text size
iOS, iPadOS	17 pt	11 pt
macOS	13 pt	10 pt
tvOS	29 pt	23 pt
visionOS	17 pt	12 pt
watchOS	16 pt	12 pt

Make sure buttons are always easy to use. Buttons that are too small or too close together can frustrate players and make gameplay less fun. Each platform defines a recommended minimum button size based on its default interaction method. For example, buttons in iOS must be at least 44x44 pt to accommodate touch interaction. For guidance, see Buttons.

Platform	Default button size	Minimum button size
iOS, iPadOS	44x44 pt	28x28 pt
macOS	28x28 pt	20x20 pt
tvOS	66x66 pt	56x56 pt

Platform	Default button size	Minimum button size
visionOS	60x60 pt	28x28 pt
watchOS	44x44 pt	28x28 pt

Prefer resolution-independent textures and graphics. If creating resolution-independent assets isn't possible, match the resolution of your game to the resolution of the device. In visionOS, prefer vector-based art that can continue to look good when the system dynamically scales it as people view it from different distances and angles. For guidance, see Images.

Integrate device features into your layout. For example, a device may have rounded corners or a camera housing that can affect parts of your interface. To help your game look at home on each device, accommodate such features during layout, relying on platform-provided safe areas when possible (for developer guidance, see Positioning content relative to the safe area). For guidance, see Layout; for templates that include safe-area guides, see Apple Design Resources.

Make sure in-game menus adapt to different aspect ratios. Games need to look good and behave well at various aspect ratios, such as 16:10, 19.5:9, and 4:3. In particular, in-game menus need to remain legible and easy to use on every device — and, if you support them, in both orientations on iPhone and iPad — without obscuring other content. To help ensure your in-game menus render correctly, consider using dynamic layouts that rely on relative constraints to adjust to different contexts. Avoid fixed layouts as much as possible, and aim to create a custom, device-specific layout only when necessary. For guidance, see In-game menus.

Design for the full-screen experience. People often enjoy playing a game in a distraction-free, full-screen context. In macOS, iOS, and iPadOS, full-screen mode lets people hide other apps and parts of the system UI; in visionOS, a game running in a Full Space can completely surround people, transporting them somewhere else.

Layout

Typography

Going full screen

Enable intuitive interactions

Support each platform's default interaction method. For example, people generally use touch to play games on iPhone; on a Mac, players tend to expect keyboard and mouse or trackpad support; and in a visionOS game, people expect to use their eyes and hands while making indirect and direct gestures. As you work to ensure that your game supports each platform's default interaction method, pay special attention to control sizing and menu behavior, especially when bringing your game from a pointer-based context to a touch-based one.

Platform	Default interaction methods	Additional interaction methods
iOS	Touch	Game controller
iPadOS	Touch	Game controller, keyboard, mouse, trackpad, Apple Pencil
macOS	Keyboard, mouse, trackpad	Game controller

Platform	Default interaction methods	Additional interaction methods
tvOS	Remote	Game controller, keyboard, mouse, trackpad
visionOS	Touch	Game controller, keyboard, trackpad
watchOS	Touch	–

Support physical game controllers, while also giving people alternatives. Every platform except watchOS supports physical game controllers. Although the presence of a game controller makes it straightforward to port controls from an existing game and handle complex control mappings, recognize that not every player can use a physical game controller. To make your game available to as many players as possible, also offer alternative ways to interact with your game.

Offer touch-based game controls that embrace the touchscreen experience on iPhone and iPad. Although your game's mechanics might require you to render controls like virtual D-pads, thumbsticks, and A, B, X, and Y buttons on top of your game views, consider letting players use touch to interact directly with game elements. For guidance, see Touch controls.

Game controls

Gestures

Pointing devices

Welcome everyone

Prioritize perceivability. Make sure people can perceive your game's content whether they use sight, hearing, or touch. For example, avoid relying solely on color to convey an important detail, or providing a cutscene that doesn't include descriptive subtitles or offer other ways to read the content. For specific guidance, see:

- Text sizes
- Color and effects
- Motion
- Interactions
- Buttons

Help players personalize their experience. Players have a variety of preferences and abilities that influence their interactions with your game. Because there's no universal configuration that suits everyone, give players the ability to customize parameters like type size, game control mapping, motion intensity, and sound balance. You can take advantage of built-in Apple accessibility technologies to support accessibility personalizations, whether you're using system frameworks or Unity plug-ins.

Give players the tools they need to represent themselves. If your game encourages players to create avatars or supply names or descriptions, support the spectrum of self-identity and provide options that represent as many human characteristics as possible.

Avoid stereotypes in your stories and characters. Ask yourself whether you're depicting game characters and scenarios in a way that perpetuates real-life stereotypes. For example, does your game depict enemies as having a certain race, gender, or cultural heritage? Review your game to uncover and remove biases and stereotypes and — if references to real-life cultures and languages are necessary — be sure they're respectful.

Accessibility

Inclusion

Adopt Apple technologies

Integrate Game Center to help players discover your game across their devices and connect with their friends. Game Center is Apple's social gaming network, available on all platforms. Game Center enables achievements, leaderboards, multiplayer experiences, and real-time play, and with SharePlay integration, it helps people share gameplay within a group activity. For guidance, see Game Center and SharePlay; for developer guidance, see GameKit and Group Activities.

Let players pick up their game on any of their devices. People often have a single iCloud account that they use across multiple Apple devices. When you support CloudKit, you can help people save their game state and start back up exactly where they left off on a different device.

Support haptics to help players feel the action. When you adopt Core Haptics, you can compose and play custom haptic patterns, optionally combined with custom audio content. Core Haptics is available in iOS, iPadOS, tvOS, and visionOS, and supported on many game controllers. For guidance, see Playing haptics; for developer guidance, see Core Haptics and Playing Haptics on Game Controllers.

Use Spatial Audio to immerse players in your game's soundscape. Providing multichannel audio can help your game's audio adapt automatically to the current device, enabling an immersive Spatial Audio experience where supported. For guidance, see Playing audio > visionOS; for developer guidance, see Explore Spatial Audio.

Take advantage of Apple technologies to enable unique gameplay mechanics. For example, you can integrate technologies like augmented reality, machine learning, and HealthKit, and request access to location data and functionality like camera and microphone. For a full list of Apple technologies, features, and services, see Technologies.

Game Center

iCloud

In-app purchase

Resources

Related

Game Center

Game controls

Developer documentation

Games Pathway

Apple technologies for game developers

Videos

Design advanced games for Apple platforms

Build great games for spatial computing

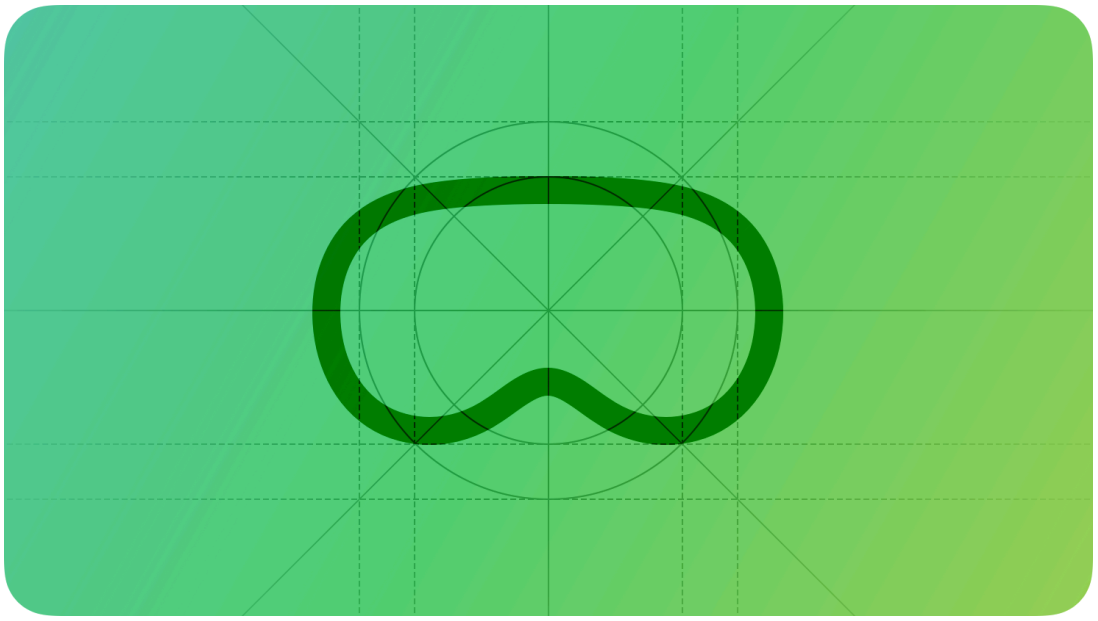
Bring your game to Mac, Part 1: Make a game plan

Change log

Date	Changes
June 10, 2024	New page.

Designing for visionOS

When people wear Apple Vision Pro, they enter an infinite 3D space where they can engage with your app or game while staying connected to their surroundings.



As you begin designing your app or game for visionOS, start by understanding the fundamental device characteristics and patterns that distinguish the platform. Use these characteristics and patterns to inform your design decisions and help you create immersive and engaging experiences.

Space. Apple Vision Pro offers a limitless canvas where people can view virtual content like windows, volumes, and 3D objects, and choose to enter deeply immersive experiences that can transport them to different places.

Immersion. In a visionOS app, people can fluidly transition between different levels of immersion. By default, an app launches in the *Shared Space* where multiple apps can run side-by-side and people can open, close, and relocate windows. People can also choose to transition an app to a *Full Space*, where it's the only app running. While in a Full Space app, people can view 3D content blended with their surroundings, open a portal to view another place, or enter a different world.

Passthrough. Passthrough provides live video from the device's external cameras, and helps people interact with virtual content while also seeing their actual surroundings. When people want to see more or less of their surroundings, they use the Digital Crown to control the amount of passthrough.

Spatial Audio. Apple Vision Pro combines acoustic and visual-sensing technologies to model the sonic characteristics of a person's surroundings, automatically making audio sound natural in their space. When an app receives a person's permission to access information about their surroundings, it can fine-tune Spatial Audio to bring custom experiences to life.

Eyes and hands. In general, people perform most actions by using their eyes to look at a virtual object and making an *indirect* gesture, like a tap, to activate it. People can also interact with a virtual object by using a *direct* gesture, like touching it with a finger.

Ergonomics. While wearing Apple Vision Pro, people rely entirely on the device's cameras for everything they see, both real and virtual, so maintaining visual comfort is paramount. The system helps maintain comfort by automatically placing content so it's relative to the wearer's head, regardless of the person's height or whether they're sitting, standing, or lying down. Because visionOS brings content to people — instead of making people move to reach the content — people can remain at rest while engaging with apps and games.

Accessibility. Apple Vision Pro supports accessibility technologies like VoiceOver, Switch Control, Dwell Control, Guided Access, Head Pointer, and many more, so people can use the interactions that work for them. In visionOS, as in all platforms, system-provided UI components build in accessibility support by default, while system frameworks give you ways to enhance the accessibility of your app or game.

> **Important**
>
> When building your app for Apple Vision Pro, be sure to consider the unique characteristics of the device and its spatial computing environment, and pay special attention to your user's safety; for more details about these characteristics, see Apple Vision Pro User Guide. For example, Apple Vision Pro should not be used while operating a vehicle or heavy machinery. The device is also not designed to be used while moving around unsafe environments such as near balconies, streets, stairs, or other potential hazards. Note that Apple Vision Pro is designed to be fit and used only by individuals 13 years of age or older.

Best practices

Great visionOS apps and games are approachable and familiar, while offering extraordinary experiences that can surround people with beautiful content, expanded capabilities, and captivating adventures.

Embrace the unique features of Apple Vision Pro. Take advantage of space, Spatial Audio, and immersion to bring life to your experiences, while integrating passthrough and spatial input from eyes and hands in ways that feel at home on the device.

Consider different types of immersion as you design ways to present your app's most distinctive moments. You can present experiences in a windowed, UI-centric context, a fully immersive context, or something in between. For each key moment in your app, find the minimum level of immersion that suits it best — don't assume that every moment needs to be fully immersive.

Use windows for contained, UI-centric experiences. To help people perform standard tasks, prefer standard windows that appear as planes in space and contain familiar controls. In visionOS, people can relocate windows anywhere they want, and the system's dynamic scaling helps keep window content legible whether it's near or far.

Prioritize comfort. To help people stay comfortable and physically relaxed as they interact with your app or game, keep the following fundamentals in mind.

- Display content within a person's field of view, positioning it relative to their head. Avoid placing content in places where people have to turn their head or change their position to interact with it.
- Avoid displaying motion that's overwhelming, jarring, too fast, or missing a stationary frame of reference.

- Support indirect gestures that let people interact with apps while their hands rest in their lap or at their sides.
- If you support direct gestures, make sure the interactive content isn't too far away and that people don't need to interact with it for extended periods.
- Avoid encouraging people to move too much while they're in a fully immersive experience.

Help people share activities with others. When you use SharePlay to support shared activities, people can view the *spatial Personas* of other participants, making it feel like everyone is together in the same space.

Resources

Related

Apple Design Resources

Developer documentation

Creating your first visionOS app

Videos

Design interactive experiences for visionOS

Design great visionOS apps

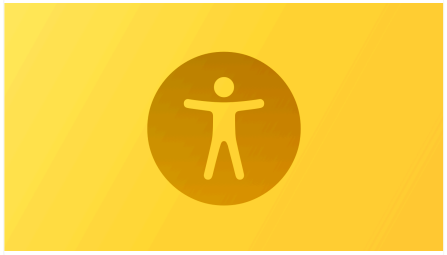
Principles of spatial design

Change log

Date	Changes
February 2, 2024	Included a link to Apple Vision Pro User Guide.
September 12, 2023	Updated intro artwork.
June 21, 2023	New page.

Foundations

Understand how fundamental design elements help you create rich experiences.



Accessibility

App icons

Branding

Color

Dark Mode

Icons

Images

Immersive experiences

Inclusion

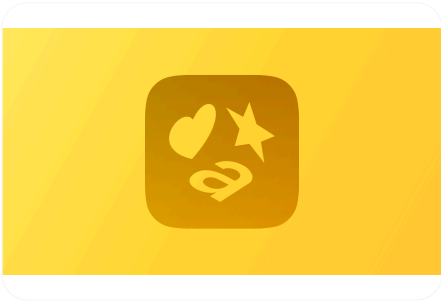
Layout

Materials

Motion

Privacy

Right to left



SF Symbols

Spatial layout

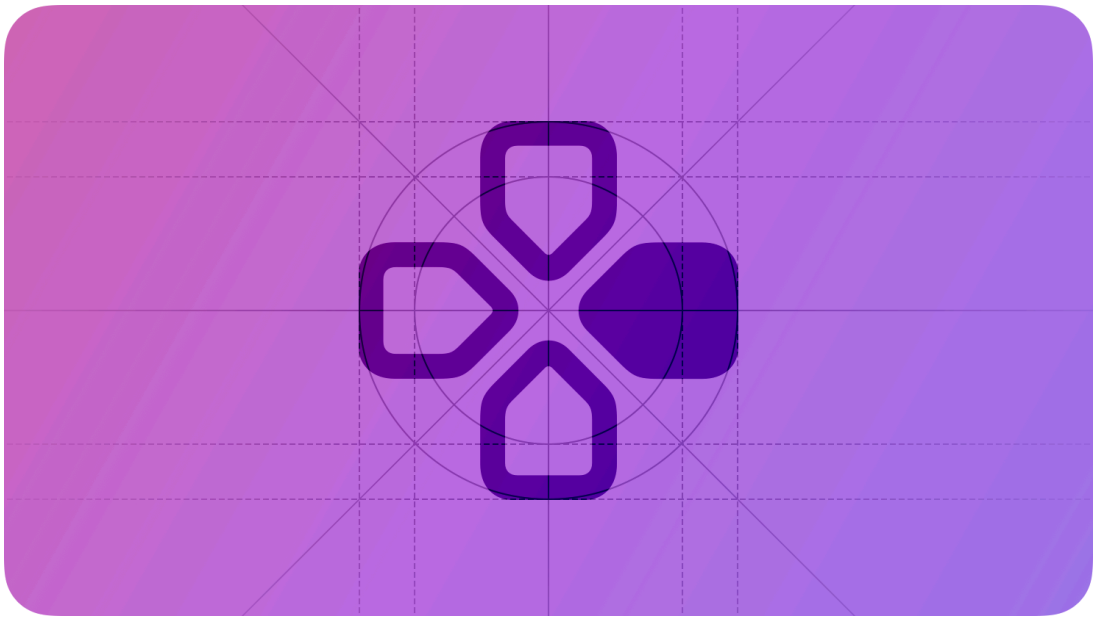
Typography

Writing

Game controls

Precise, intuitive game controls enhance gameplay and can increase a player's immersion in the game.

Supported platforms



On Apple platforms, a game can support input from physical game controllers or default system interactions, like touch, a remote, or a mouse and keyboard. Many players prefer to use physical game controllers, but there are two important reasons to also support a platform's default interaction methods:

- Even though all platforms except watchOS support physical game controllers, not every player can use one.
- Players appreciate games that let them use the platform interaction method they're most familiar with.

For games that run in iOS and iPadOS, supporting touch interaction means that you can render virtual game controls on top of game UI while also letting players interact with game elements by touching them directly.

For guidance on supporting players who use the keyboard to control your game, see Key bindings.

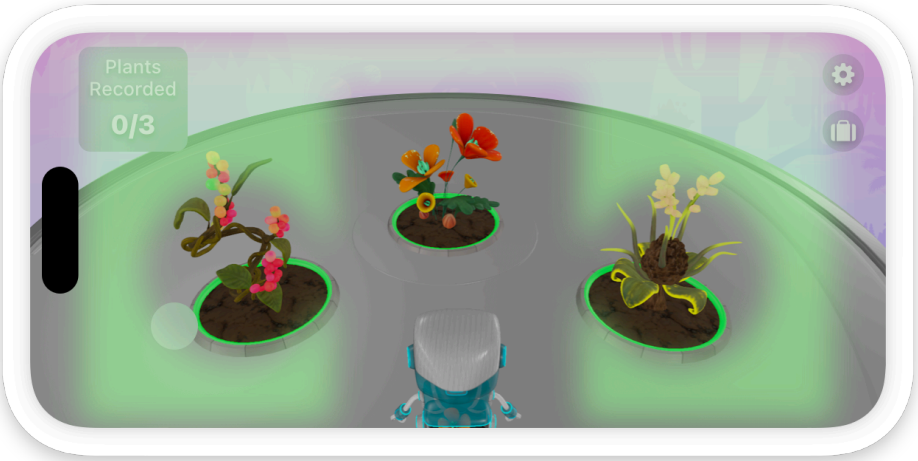
Touch controls

Determine when it makes sense to display virtual controls on top of game views. In general, a game that offers a large number of actions or requires players to control movement needs to display virtual game controls. In contrast, when a virtual control maps directly to an onscreen game element, consider removing the control and letting people interact directly with the element. For example, players often appreciate relocating a game item by dragging it or by

tapping its destination, and they expect to open an in-game menu by tapping it. For developer guidance, see Adding touch controls to games that support game controllers in iOS.

Place virtual controls where they're easy to target without letting them obstruct other controls or game content. Consider using larger controls and hit regions for actions players must make quickly or frequently, and smaller ones for other actions. If you need to offer thumbsticks, consider displaying a minimal design that hides when players don't need to use them. As you lay out virtual controls, be sure to support both landscape orientations and take advantage of system-defined safe areas to help you avoid a device's display or interactive features, like the Dynamic Island or a camera housing.

Design for two fingers. People often use their thumbs for virtual controls, so the range of possible actions is more limited than with a physical controller. Consider redesigning game mechanics that require players to press multiple buttons. For interactions that require holding a button, consider making that button a toggle or — when there are multiple actions, such as readying a weapon and aiming — combining the actions into one control.

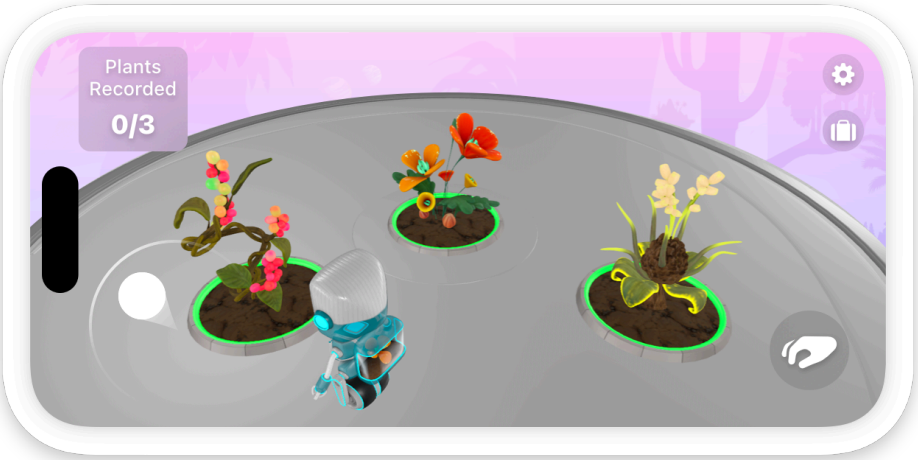


Placing virtual controls within reach of people's thumbs can make your game more comfortable to play.

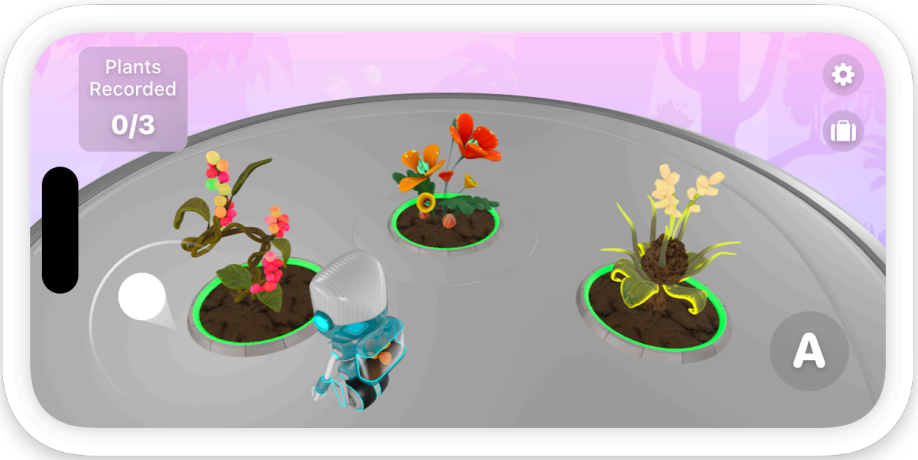
Always include a press state for each virtual control. Without a visual press state, a virtual control can feel unresponsive, making players wonder if it accepted their touch. To provide a press state, you might simply design an appearance that's darker or lighter than the default one, or you might add a glow that appears outside the bounds of the control, helping players see it even when their finger is covering the button.

Consider using haptics to enhance feedback. Subtle haptic feedback can combine with a visual press state to help players confirm that a virtual control is responding to their touch. You might also use subtle haptics to convey the range of an analog control, helping players feel when they reach its limit, or when they engage a different mode. For guidance, see Playing haptics.

Prefer designing expressive virtual controls that communicate the actions they perform. Players can more easily learn and remember what a control does when its appearance visually represents its action. For example, you might design an attack button that displays a graphic of the weapon that's currently equipped, or a movement control that uses an arrow to show the direction of motion.



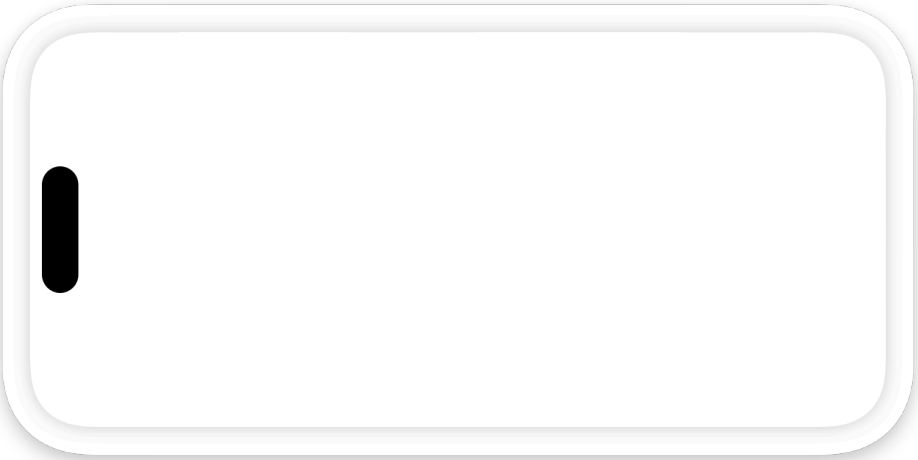
A control that communicates its function during gameplay



A control that requires players to remember what it does

Aim to design virtual controls that dynamically change to reflect gameplay. Instead of rendering a static overlay that reproduces the buttons on a physical controller, take advantage of touch interaction to give players virtual controls that adapt to gameplay. For example, if the behavior of a virtual control changes based on the current context, its appearance can update, signaling the change to players. Using adaptive virtual controls also means that you can hide a control when it isn't available or relevant, letting you reduce clutter and help players concentrate on what's important.

When the thumbstick moves to the left, the virtual control displays a leaf symbol in its label.



When the thumbstick is at rest, the virtual control hides because it isn't available.

Consider supporting direct touch interactions when it makes sense. Letting players interact directly with elements of your game can simplify control and navigation, and can increase player engagement. For example, players often prefer using direct touch interaction to control a camera, because doing so can result in greater precision than when using a virtual thumbstick. In particular, be sure to let players use touch interactions to control an in-game menu — don't require them to use virtual controls.

Physical controllers

Support the platform's default interaction method. A game controller is an optional purchase, but every iPhone and iPad has a touchscreen, every Mac has a keyboard and a trackpad or mouse, and every Apple TV has a remote. If you support game controllers, consider letting people use the platform's default interaction method to control games, too.

Tell people about game controller requirements. When necessary, you can require the use of a physical game controller. The App Store displays a "Game Controller Required" badge to help people identify such apps and may warn people if they haven't paired a compatible game controller with their device. For developer guidance, see `GCRequiresControllerUserInteraction`.

Confirm required game controller connections. People can open your game any time, even without a connected controller. If your app requires a game controller, check for its presence and gracefully prompt people to connect one if necessary.

Help people understand the advantages of using a game controller with your app. If your game is playable without a game controller, but using one is recommended, look for ways to tell players that using a game controller can make playing easier and more enjoyable.

Avoid requiring players to switch input devices. For example, make sure players can use a physical game controller to navigate all game menus and interact with all essential functions without switching to another interaction method.

Support thumbstick buttons when present. Some controllers include thumbsticks that players can press like buttons as well as rotate. For example, pressing a thumbstick that moves the player's character might change the movement speed, while pressing a thumbstick that controls camera orientation might perform a zoom or crouch. As you consider ways to support thumbstick buttons, be guided by the behaviors that people expect in various game genres.

Customize onscreen content to match the connected game controller. To simplify your game's code, the Game Controller framework assigns standard names to controller elements

based on their placement, but the colors and symbols on an actual game controller may differ. Be sure to use the connected controller's labeling scheme when referring to controls or displaying related content in your interface. For developer guidance, see `GCControllerElement`.

Prefer using symbols, not text, to refer to game controller elements. The Game Controller framework makes SF Symbols available for most elements, including the buttons on various brands of game controllers. Using symbols instead of text descriptions can be especially helpful for players who aren't experienced with controllers because it doesn't require them to hunt for a specific button label during gameplay.

Support multiple connected controllers. If there are multiple controllers connected, use labels and glyphs that match the one that the player is actively using. If your game supports multiplayer, use the appropriate labels and symbols when referring to a specific player's controller. If you need to refer to buttons on multiple controllers, consider listing them together.

Supporting a game controller in a nongame app

People sometimes use a game controller to navigate the system and use nongame apps. In this scenario, support each platform's expected behaviors for various game controller elements.

Button	Expected behavior in iOS and iPadOS	Expected behavior in macOS	Expected behavior in tvOS
D-pad	Moves focus	Moves focus	Moves focus
A	Activates a control or selects an item	Activates a control or selects an item	Activates a control or selects an item
B	Cancels an action or returns to previous screen	Cancels an action or returns to previous screen	Returns to previous screen or exits to Apple TV Home Screen from an app's root-level screen
X	—	—	Starts or pauses media playback
Y	—	—	—
Left shoulder/trigger	Navigates left or moves focus	Navigates left or moves focus	Navigates left or moves focus
Right shoulder/trigger	Navigates right or moves focus	Navigates right or moves focus	Navigates right or moves focus
Left thumbstick	Navigates or moves focus	Navigates or moves focus	Navigates or moves focus
Right thumbstick	—	—	—

A controller with a single auxiliary button needs to support the following behaviors:

Interaction with auxiliary button	Expected behavior in iOS and iPadOS	Expected behavior in macOS	Expected behavior in tvOS
Press	—	—	Returns to previous screen
Long press	—	—	Exits to Apple TV Home Screen
Double press	—	—	Reveals multitasking user interface

A controller with multiple auxiliary buttons includes a logo button in addition to the Menu button. Such a controller needs to support the following behaviors when people interact with the logo and Menu buttons:

Button	Interaction	Expected behavior in iOS and iPadOS	Expected behavior in macOS	Expected behavior in tvOS
Logo	Press	Reveals Game Center (if available)	Reveals Game Center (if available)	Reveals Game Center (if available)
	Long press	Reveals Games folder in App Library	Reveals Games folder in Launchpad	Exits to Apple TV Home Screen
	Double press	—	—	Reveals multitask-ing user interface
Menu	Press	—	—	Reveals Control Center or opens app settings

Platform considerations

No additional considerations for iOS, iPadOS, macOS, tvOS, or visionOS. Not supported in watchOS.

Resources

Related

Feedback

Developer documentation

Game Controller

Videos

Explore game input in visionOS

Supporting New Game Controllers

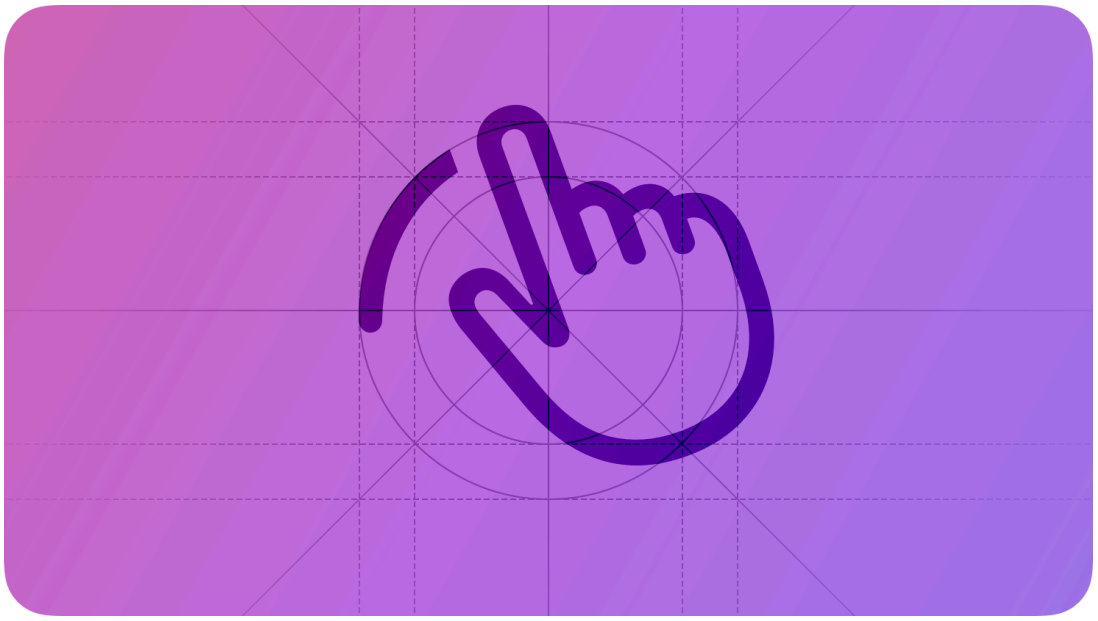
Change log

Date	Changes
June 10, 2024	Added guidance for supporting touch controls and changed title from Game controllers.

Gestures

A gesture is a physical motion that a person uses to directly affect an object in an app or game on their device.

Supported platforms



Depending on the device they're using, people can make gestures on a touchscreen, in the air, or on a range of input devices such as a trackpad, mouse, remote, or game controller that includes a touch surface.

Every platform supports basic gestures like tap, swipe, and drag. Although the precise movements that make up basic gestures can vary per platform and input device, people are familiar with the underlying functionality of these gestures and expect to use them everywhere. For a list of these gestures, see Standard gestures.

Best practices

Give people more than one way to interact with your app. People commonly prefer or need to use other inputs — such as their voice, keyboard, or Switch Control — to interact with their devices. Don't assume that people can use a specific gesture to perform a given task. For guidance, see Accessibility.

In general, respond to gestures in ways that are consistent with people's expectations. People expect most gestures to work the same regardless of their current context. For example, people expect tap to activate or select an object. Avoid using a familiar gesture like tap or swipe to perform an action that's unique to your app; similarly, avoid creating a unique gesture to perform a standard action like activating a button or scrolling a long view.

Handle gestures as responsively as possible. Useful gestures enhance the experience of direct manipulation and provide immediate feedback. As people perform a gesture in your app, provide

feedback that helps them predict its results and, if necessary, communicates the extent and type of movement required to complete the action.

Indicate when a gesture isn't available. If you don't clearly communicate why a gesture doesn't work, people might think your app has frozen or they aren't performing the gesture correctly, leading to frustration. For example, if someone tries to drag a locked object, the UI may not indicate that the object's position has been locked; or if they try to activate an unavailable button, the button's unavailable state may not be clearly distinct from its available state.

Custom gestures

Add custom gestures only when necessary. Custom gestures work best when you design them for specialized tasks that people perform frequently and that aren't covered by existing gestures, like in a game or drawing app. If you decide to implement a custom gesture, make sure it's:

- Discoverable
- Straightforward to perform
- Distinct from other gestures
- Not the only way to perform an important action in your app or game

Make custom gestures easy to learn. Offer moments in your app to help people quickly learn and perform custom gestures, and make sure to test your interactions in real use scenarios. If you're finding it difficult to use simple language and graphics to describe a gesture, it may mean people will find the gesture difficult to learn and perform.

Use shortcut gestures to supplement standard gestures, not replace them. While you may supply a custom gesture to quickly access parts of your app, people also need simple, familiar ways to navigate and perform actions, even if it means an extra tap or two. For example, in an app that supports navigation through a hierarchy of views, people expect to find a Back button in a navigation bar that lets them return to the previous view with a single tap. To help accelerate this action, many apps also offer a shortcut gesture — such as swiping from the side of a window or touchscreen — while continuing to provide the Back button.

Avoid conflicting with gestures that access system UI. Several platforms offer gestures for accessing system behaviors, like edge swiping in watchOS or rolling your hand over to access system overlays in visionOS. It's important to avoid defining custom gestures that might conflict with these interactions, as people expect these controls to work consistently. In specific circumstances within games or immersive experiences, developers can work around this area by deferring the system gesture. For more information, see the platform considerations for iOS, iPadOS, watchOS, and visionOS.

Platform considerations

iOS, iPadOS

In addition to the standard gestures supported in all platforms, iOS and iPadOS support a few other gestures that people expect.

Gesture	Common action
Three-finger swipe	Initiate undo (left swipe); initiate redo (right swipe).

Gesture	Common action
Three-finger pinch	Copy selected text (pinch in); paste copied text (pinch out).
Four-finger swipe (iPadOS only)	Switch between apps.
Shake	Initiate undo; initiate redo.

Consider allowing simultaneous recognition of multiple gestures if it enhances the experience. Although simultaneous gestures are unlikely to be useful in nongame apps, a game might include multiple onscreen controls — such as a joystick and firing buttons — that people can operate at the same time. For guidance on integrating touchscreen input with Apple Pencil input in your iPadOS app, see Apple Pencil and Scribble.

macOS

People primarily interact with macOS using a keyboard and mouse. In addition, they can make standard gestures on a Magic Trackpad, Magic Mouse, or a game controller that includes a touch surface.

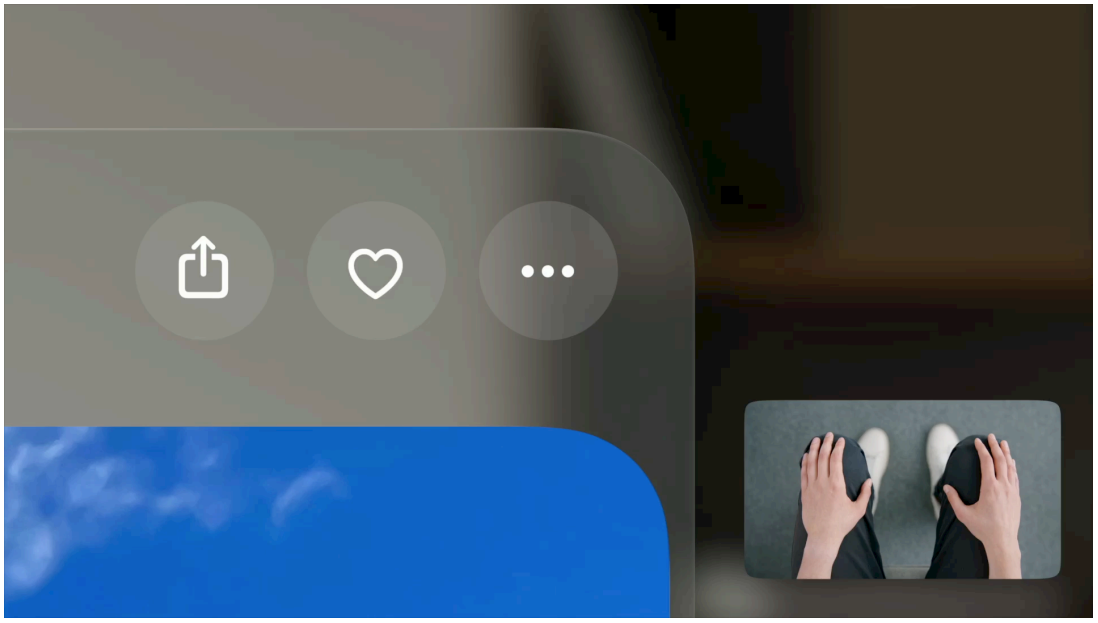
tvOS

People expect to use standard gestures to navigate tvOS apps and games with a compatible remote, Siri Remote, or game controller that includes a touch surface. For guidance, see Remotes.

visionOS

visionOS supports two categories of gestures: indirect and direct.

People use an *indirect* gesture by looking at an object to target it, and then manipulating that object from a distance — indirectly — with their hands. For example, a person can look at a button to focus it and select it by quickly tapping their finger and thumb together. Indirect gestures are comfortable to perform at any distance, and let people quickly change focus between different objects and select items with minimal movement.



Play

People use a *direct* gesture to physically touch an interactive object. For example, people can directly type on the visionOS keyboard by tapping the virtual keys. Direct gestures work best when they are within reach. Because people may find it tiring to keep their arms raised for extended periods, direct gestures are best for infrequent use. visionOS also supports direct versions of all standard gestures, allowing people the choice to interact directly or indirectly with any standard component.

Play

Here are the standard direct gestures people use in visionOS; see Specifications for a list of standard indirect gestures.

Direct gesture	Common use
Touch	Directly select or activate an object.
Touch and hold	Open a contextual menu.
Touch and drag	Move an object to a new location.
Double touch	Preview an object or file; select a word in an editing context.
Swipe	Reveal actions and controls; dismiss views; scroll.
With two hands, pinch and drag together or apart	Zoom in or out.
With two hands, pinch and drag in a circular motion	Rotate an object.

Support standard gestures everywhere you can. For example, as soon as someone looks at an object in your app or game, tap is the first gesture they're likely to make when they want to select or activate it. Even if you also support custom gestures, supporting standard gestures such as tap helps people get comfortable with your app or game quickly.

Offer both indirect and direct interactions when possible. Prefer indirect gestures for UI and common components like buttons. Reserve direct gestures and custom gestures for objects that invite close-up interaction or specific motions in a game or interactive experience.

Avoid requiring specific body movements or positions for input. Not all people can perform specific body movements or position themselves in certain ways at all times, whether due to disability, spatial constraints, or other environmental factors. If your experience requires movement, consider supporting alternative inputs to let people choose the interaction method that works best for them.

Designing custom gestures in visionOS

If you want to offer a specific interaction for your experience that people can't perform using an existing system gesture, consider designing a custom gesture. To offer this type of interaction, your app needs to be running in a Full Space, and you must request people's permission to access information about their hands. For developer guidance, see Setting up access to ARKit data.

Prioritize comfort. Continually test ergonomics of all interactions that require custom gestures. A custom interaction that requires people to keep their arms raised for even a little while can be physically tiring, and repeating very similar movements many times in succession can stress people's muscles and joints.

Carefully consider complex custom gestures that involve multiple fingers or both hands. People may not always have both hands available when using your app or game. If you require a more complex gesture for your experience, consider also offering an alternative that requires less movement.

Avoid custom gestures that require using a specific hand. It can increase someone's cognitive load if they need to remember which hand to use to trigger a custom gesture. It may also make your experience less welcoming to people with strong hand-dominance or limb differences.

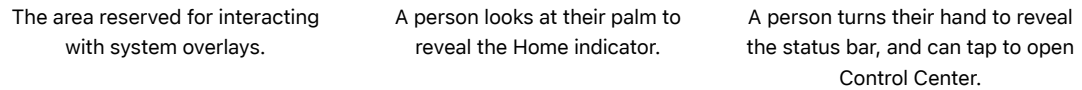
Working with system overlays in visionOS

In visionOS 2 and later, people can look at the palm of one hand and use gestures to quickly access system overlays for Home and Control Center. These interactions are available systemwide, and are reserved solely for accessing system overlays.

> **Note**
>
> The system overlay is the default method of accessing Control Center in visionOS 2 and later. The visionOS 1 behavior (looking upward) remains available as an accessibility setting.

When designing apps and games that use custom gestures or anchor content to a person's hands, it's important to take interactions with the system overlays into consideration.

Reserve the area around a person's hand for system overlays and their related gestures. If possible, don't anchor content to a person's hands or wrists. If you're designing a game that involves hand-anchored content, place it outside of the immediate area of someone's hand to avoid colliding with the Home indicator.



The area reserved for interacting with system overlays.

A person looks at their palm to reveal the Home indicator.

A person turns their hand to reveal the status bar, and can tap to open Control Center.

Consider deferring the system overlay behavior when designing an immersive app or game. In certain circumstances, you may not want the Home indicator to appear when someone looks

at the palm of their hand. For example, a game that uses virtual hands or gloves may want to keep someone within the world of the story, even if they happen to look at their hands from different angles. In such cases, when your app is running in a Full Space, you can choose to require a tap to reveal the Home indicator instead. For developer guidance, see `persistentSystemOverlays(_:)`.

Default behavior in the Shared Space | Default behavior in a Full Space | Deferred behavior in a Full Space

> **Note**
>
> Apps and games that you built for visionOS 1 defer the system overlay behavior by default. When a person looks at their palm with your app running in a Full Space, the Home indicator won't appear unless they tap first.

Use caution when designing custom gestures that involve a rolling motion of the hand, wrist, and forearm. This specific motion is reserved for revealing system overlays. Since system overlays always display on top of app content and your app isn't aware of when they're visible, it's important to test any custom gestures or content that might conflict.

watchOS

Double tap

In watchOS 11 and later, people can use the double-tap gesture to scroll through lists and scroll views, and to advance between vertical tab views. Additionally, you can specify a toggle or button as the primary action in your app, or in your widget or Live Activity when the system displays it in the Smart Stack. Double-tapping in a view with a primary action highlights the control and then performs the action. The system also supports double tap for custom actions that you offer in notifications, where it acts on the first nondestructive action in the notification.

Avoid setting a primary action in views with lists, scroll views, or vertical tabs. This conflicts with the default navigation behaviors that people expect when they double-tap.

Choose the button that people use most commonly as the primary action in a view. Double tap is helpful in a nonscrolling view when it performs the action that people use the most. For example, in a media controls view, you could assign the primary action to the play/pause button. For developer guidance, see `handGestureShortcut(_:isEnabled:)` and `primaryAction`.

Specifications

Standard gestures

The system provides APIs that support the familiar gestures people use with their devices, whether they use a touchscreen, an indirect gesture in visionOS, or an input device like a trackpad, mouse, remote, or game controller. For developer guidance, see Gestures.

Gesture	Supported in	Common action
Tap	iOS, iPadOS, macOS, tvOS, visionOS, watchOS	Activate a control; select an item.
Swipe	iOS, iPadOS, macOS, tvOS, visionOS, watchOS	Reveal actions and controls; dismiss views; scroll.
Drag	iOS, iPadOS, macOS, tvOS, visionOS, watchOS	Move a UI element.
Touch (or pinch) and hold	iOS, iPadOS, tvOS, visionOS, watchOS	Reveal additional controls or functionality.
Double tap	iOS, iPadOS, macOS, tvOS, visionOS, watchOS	Zoom in; zoom out if already zoomed in; perform a primary action on Apple Watch Series 9 and Apple Watch Ultra 2.
Zoom	iOS, iPadOS, macOS, tvOS, visionOS	Zoom a view; magnify content.
Rotate	iOS, iPadOS, macOS, tvOS, visionOS	Rotate a selected item.

For guidance on supporting additional gestures and button presses on specific input devices, see Pointing devices, Remotes, and Game controls.

Resources

Related

Feedback

Eyes

Developer documentation

`UITouch` — UIKit

Videos

Design for spatial input

Principles of spatial design

Design for spatial user interfaces

Change log

Date	Changes
September 9, 2024	Added guidance for working with system overlays in visionOS and made organizational updates.
September 15, 2023	Updated specifications to include double tap in watchOS.
June 21, 2023	Changed page title from Touchscreen gestures and updated to include guidance for visionOS.

Getting started

Create an app or game that feels at home on every platform you support.

Designing for iOS

Designing for iPadOS

Designing for macOS

Designing for tvOS

Designing for visionOS

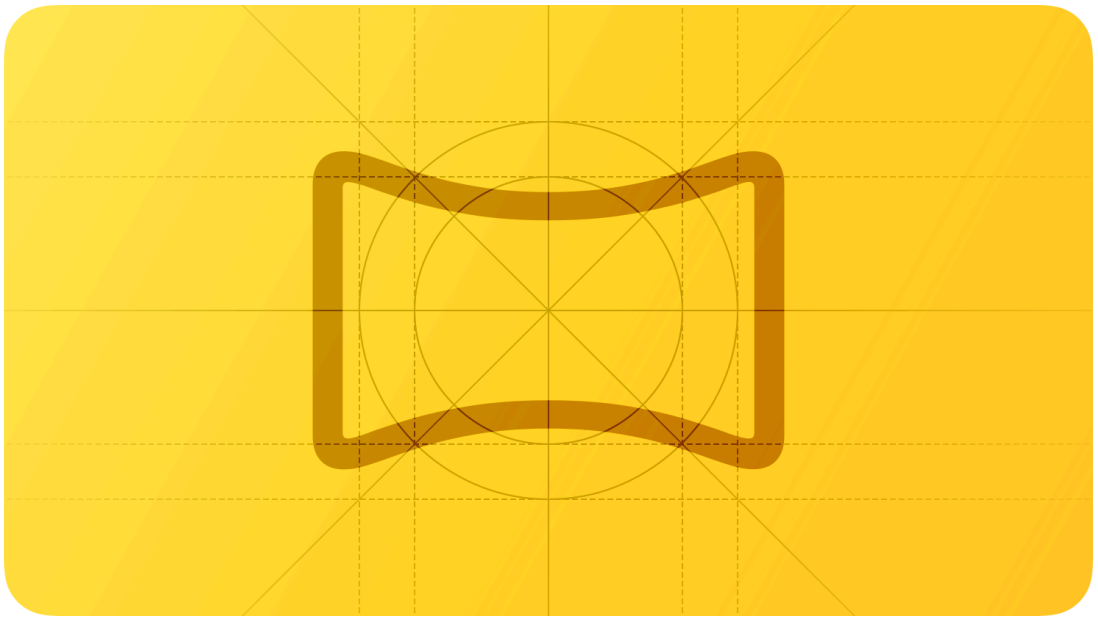
Designing for watchOS

Designing for games

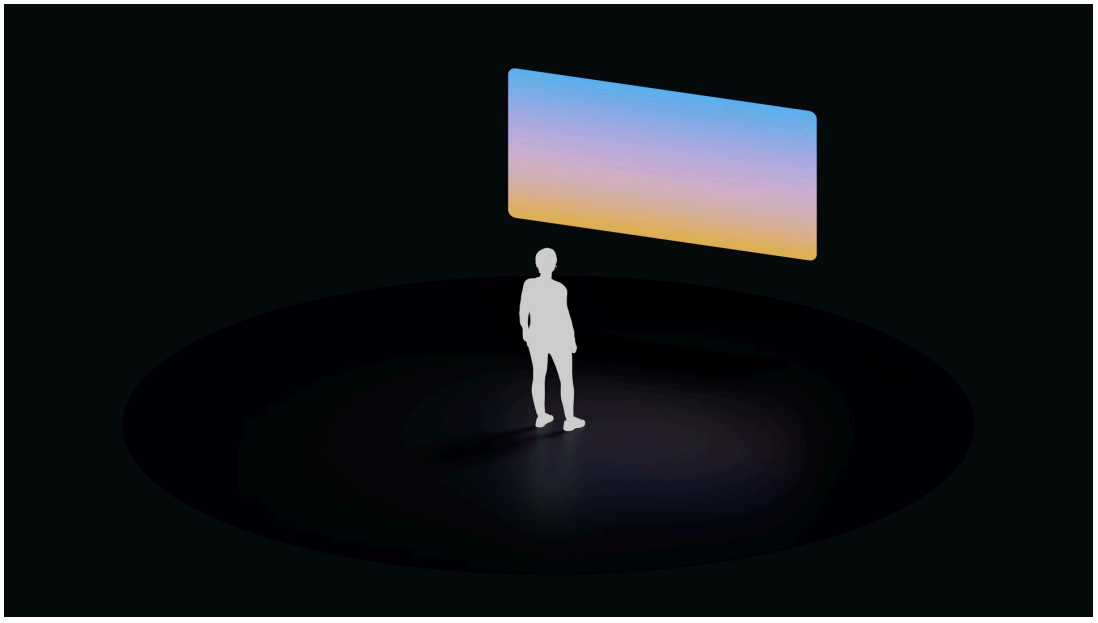
Immersive experiences

In visionOS, you can design apps and games that extend beyond windows and volumes, immersing people in your content.

Supported platforms



You can choose whether your visionOS app or game runs in the Shared Space or in a Full Space. In the *Shared Space*, your software runs alongside other experiences, and people can switch between them much as they do on a Mac; in a *Full Space*, your app or game runs alone, hiding other experiences and helping people immerse themselves in your content. Apps and games can support different types of immersion, and can transition fluidly between the Shared Space and a Full Space.



Play

Immersion and passthrough

While wearing Apple Vision Pro, people use passthrough to see their physical surroundings. *Passthrough* provides real-time video from the device's external cameras, helping people feel comfortable and connected to their physical context.

People can also use the Digital Crown to help them manage app or game content and adjust passthrough. For example, people can press and hold the Digital Crown to recenter content in their field of view or double-click it to briefly hide all content and show passthrough for a clear view of their surroundings.

Play

The system also helps people remain comfortable by automatically changing the opacity of content in certain situations. For example, if someone gets too close to a physical object in `mixed` immersion, the content in front of them dims briefly so they can see their immediate physical surroundings more clearly. In more immersive experiences — such as in the `progressive` and `full` styles described below — the system defines a boundary that extends about 1.5 meters from the initial position of the wearer's head. As their head gets close to this boundary, the entire experience begins to fade and passthrough increases. When their head moves beyond this boundary, the immersive visuals are replaced in space by the app's icon, and are restored when the wearer returns to their original location or recenters their view using the Digital Crown.

Immersion styles

visionOS offers several ways to immerse people in your content. For example, you can:

- **Use dimmed passthrough to bring attention to your content.** You can subtly dim or tint passthrough and other visible content to bring attention to your app in the Shared Space without hiding other apps and games, or create a more focused experience in a Full Space. While passthrough is tinted black by default, you can apply a custom tint color to create a dynamic experience in your app. For developer guidance, see `SurroundingsEffect`.

Without dimmed passthrough | With dimmed passthrough

- **Increase immersion in a Full Space.** When your app or game transitions to a Full Space, the system hides other apps so people can focus on yours. In a Full Space, you can display 3D content that isn't bound by a window, in addition to content in standard windows and volumes. For developer guidance, see `automatic`.
  - **Use `mixed` immersion to blend your content with passthrough.** When your app or game runs in a Full Space, you can request access to information about nearby physical objects and room layout, helping you display virtual content in a person's surroundings. The `mixed` immersion style doesn't define a boundary. Instead, when a person gets too close to a physical object, the system automatically makes nearby content semi-opaque to help them remain aware of their surroundings. For developer guidance, see `mixed` and ARKit.
  - **Use `progressive` immersion to blend your custom environment with a person's surroundings.** When your app or game runs in a Full Space, you can use the `progressive` style to display a custom environment that partially replaces passthrough. You can also define a specific range of immersion that works best with your app or game. While in your immersive experience, people can use the Digital Crown to adjust the amount of immersion within either the default range of 120- to 360-degrees or a custom range, if you specify one. The system defines the approximately 1.5-meter boundary when an experience using the `progressive` style starts. For developer guidance, see `progressive`.
  - **Use `full` immersion to create a fully immersive experience.** When your app or game runs in a Full Space, you can use the `full` style to display a 360-degree custom environment that completely replaces passthrough and transports people to a new place. As with the `progressive` style, the system defines the approximately 1.5-meter boundary when a fully immersive experience starts. For developer guidance, see `full`.

Full Space (Mixed) | Full Space (Progressive) | Full Space (Immersive)

Mixed immersion style in a Full Space blending in-app objects with real-world surroundings

Best practices

Offer multiple ways to use your app or game. In addition to giving people the freedom to choose their experiences, it's essential to design your software to support the accessibility features people use to personalize the ways they interact with their devices. For guidance, see Accessibility.

Prefer launching your app or game in the Shared Space or using the `mixed` immersion style. Launching in the Shared Space lets people reference your app or game while using other running software, and enables seamless switching between them. If your app or game provides a fully immersive or `progressive` style experience, launching in the `mixed` immersion style or with a window-based experience in the Shared Space gives people more control, letting them choose when to increase immersion.

Reserve immersion for meaningful moments and content. Not every task benefits from immersion, and not every immersive task needs to be fully immersive. Although people sometimes want to enter a different world, they often want to stay grounded in their surroundings while they're using your app or game, and they can appreciate being able to use other software and system features at the same time. Instead of assuming that your app or game needs to be fully immersive most of the time, design ways for people to immerse themselves in the individual tasks and content that make your experience unique. For example, people can browse their albums in Photos using a familiar app window in the Shared Space, but when they want to examine a single photo, they can temporarily transition to a more immersive experience in a Full Space where they can expand the photo and appreciate its details.

Help people engage with key moments in your app or game, regardless of the level of immersion. Cues like dimming, tinting, motion, scale, and Spatial Audio can help draw people's attention to a specific area of content, whether it's in a window in the Shared Space or in a completely immersive experience in a Full Space. Start with subtle cues that gently guide people's attention, strengthening the cues only when there's a good reason to do so.

Prefer subtle tint colors for passthrough. In visionOS 2 and later, you can tint passthrough to help a person's surroundings visually coordinate with your content, while also making their hands look like they belong in your experience. Avoid bright or dramatic tints that can distract people and diminish the sense of immersion. For developer guidance, see `Surroundings Effect`.

Promoting comfort

Be mindful of people's visual comfort. For example, although you can place 3D content anywhere while your app or game is running in a Full Space, prefer placing it within people's field of view. Also, make sure you display motion in comfortable ways while your software runs in a Full Space to avoid causing distraction, confusion, or discomfort. For guidance, see Motion.

Avoid encouraging people to move while they're in an immersive experience. Some people may not want to move, or are unable to move because of a disability or their physical environment. Design ways for people to interact with content without moving. For example, let people bring a virtual object closer to them instead of expecting them to move close to the object.

Choose a style of immersion that supports the movements people might make while they're in your app or game. It's essential to choose the right style for your immersive experience because it allows the system to respond appropriately when people move. Although people can make minor physical movements while in an immersive experience — such as shifting their weight, turning around, or switching between sitting and standing — making excessive movements can cause the system to interrupt some experiences. In particular, avoid using the `progressive` or `full` immersion styles if you think people might move beyond the 1.5-meter boundary.

If you use the `mixed` immersion style, avoid obscuring passthrough too much. People use passthrough to help them understand and navigate their physical surroundings, so it's important to avoid displaying virtual objects that block too much of their view. If your app or game displays virtual objects that could substantially obscure passthrough, use the `full` or `progressive` immersion styles instead of `mixed`.

Adopt ARKit if you want to blend custom content with someone's surroundings. For example, you might want to integrate virtual content into someone's surroundings or use the wearer's hand positions to inform your experience. If you need access to these types of sensitive data, you must request people's permission. For guidance, see Privacy; for developer guidance, see `SceneReconstructionProvider`.

Transitioning between immersive styles

Design smooth, predictable transitions when changing immersion. Help people prepare for different experiences by providing gentle transitions that let people visually track changes. Avoid sudden, jarring transitions that might be disorienting or uncomfortable. For developer guidance, see `CoordinateSpaceProtocol`.

Let people choose when to enter a more immersive experience. You don't want to put people into a more immersive experience when they're not expecting it, and you don't want to overwhelm them by suddenly displaying large windows or objects. Instead, provide a clear entry control so people can decide when to be more immersed in your content.

Let people choose when to exit an immersive experience. For example, Keynote provides a prominent Exit button to help people leave the Theater experience and return to the slide-viewing window. Make sure your button clarifies whether it returns people to a previous, less immersive context or quits an experience altogether. If exiting your immersive experience also quits your app or game, consider providing controls that let people pause or return to a place where they can save their progress before quitting. In particular, avoid making the close button in Control Center or pressing the Digital Crown the only ways people can leave your immersive experience.

Displaying virtual hands

When running in a Full Space, your immersive app or game can ask permission to hide a person's hands and instead show virtual hands that represent them.

Prefer virtual hands that match familiar characteristics. For example, match the positions and gestures of the viewer's hands so they can continue to interact with your app or game in ways that feel natural. Hands that work in familiar ways help people stay immersed in the experience when in fully virtual worlds.

Use caution if you create virtual hands that are larger than the viewer's hands. Virtual hands that are significantly bigger than human hands can prevent people from seeing the content they're interested in and can make interactions feel clumsy. Also, large virtual hands can seem out of proportion with the space, appearing to be too close to the viewer's face.

If there's an interruption in hand-tracking data, fade out virtual hands and reveal the viewer's own hands. Don't let the virtual hands become unresponsive and appear frozen. When hand-tracking data returns, fade the virtual hands back in.

Creating an environment

When your app or game runs in a Full Space, you can replace passthrough with a custom environment that partially or completely surrounds a person, transporting them to a new place. The following guidelines can help you design a beautiful environment that people appreciate.

> Note
>
> You can only display a custom environment in a Full Space.

Minimize distracting content. To help immerse people in a primary task like watching a video, avoid displaying a lot of movement or high-contrast details in your environment. Alternatively, when you want to draw people's attention to certain areas of your environment, consider techniques like using the highest quality textures and shapes in the important area while using lower quality assets and dimming in less important areas.

Help people distinguish interactive objects in your environment. People often use an object's proximity to help them decide if they can interact with it. For example, when you place a 3D object far away from people, they often don't try to touch or move toward it, but when you place a 3D object close to people, they're more likely to try interacting with it.

Keep animation subtle. Small, gentle movements, like clouds drifting or transforming, can enrich your custom environment without distracting people or making them uncomfortable. Always avoid displaying too much movement near the edges of a person's field of view. For guidance, see Motion.

Create an expansive environment, regardless of the place it depicts. A small, restrictive environment can make people feel uncomfortable and even claustrophobic.

Use Spatial Audio to create atmosphere. In visionOS, you use Spatial Audio to play sound that people can perceive as coming from specific locations in space, not just from speakers (for guidance, see Playing audio). As you design a soundscape that enhances your custom environment, keep the experience fresh and captivating by avoiding too much repetition or looping. If people can play other audio while they're in your environment — for example, while watching a movie — be sure to lower the volume of the soundscape or stop it completely.

In general, avoid using a flat 360-degree image to create your environment. A 360-degree image doesn't tend to give people a sense of scale when they view it in an environment, so it can

reduce the immersiveness of the experience. Prefer creating object meshes that include lighting, and use shaders to implement subtle animations like the movements of clouds or leaves or the reflections of objects.

Help people feel grounded. Always provide a ground plane mesh so people don't feel like they're floating. If you must use a flat 360-degree image in your environment, adding a ground plane mesh can help it feel more realistic.

Minimize asset redundancy. Using the same assets or models too frequently tends to make an environment feel less realistic.

Platform considerations

Not supported in iOS, iPadOS, macOS, tvOS, or watchOS.

Resources

Related

Spatial layout

Motion

Developer documentation

Creating fully immersive experiences in your app — visionOS

Incorporating real-world surroundings in an immersive experience — visionOS

`ImmersionStyle` — visionOS

Immersive spaces — SwiftUI

Videos

Create custom environments for your immersive apps in visionOS

Principles of spatial design

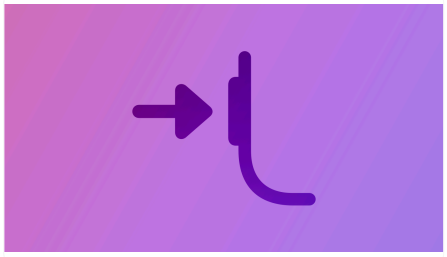
Design spatial SharePlay experiences

Change log

Date	Changes
November 19, 2024	Refined immersion style guidance and added artwork.
June 10, 2024	Added guidance for tinting passthrough and specifying initial, minimum, and maximum immersion levels.
May 7, 2024	Added guidance for creating an environment.
February 2, 2024	Clarified guidance for choosing an immersion style that matches the experience your app provides.
October 24, 2023	Updated artwork.
June 21, 2023	New page.

Inputs

Learn about the various methods people use to control your app or game and enter data.



Action button

Apple Pencil and Scribble

Camera Control

Digital Crown

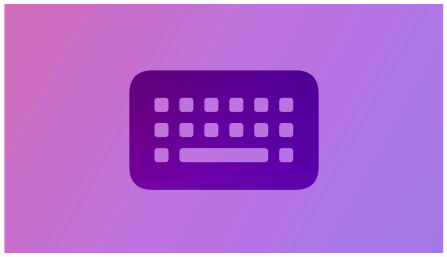
Eyes

Focus and selection

Game controls

Gestures

Gyroscope and accelerometer



Keyboards

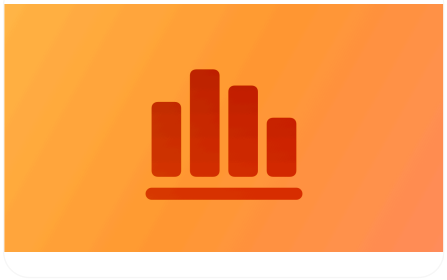
Nearby interactions

Pointing devices

Remotes

Patterns

Get design guidance for supporting common user actions, tasks, and experiences.



Charting data

Collaboration and sharing

Drag and drop

Entering data

Feedback

File management

Going full screen

Launching

Live-viewing apps

Loading

Managing accounts

Managing notifications

Modality

Multitasking

Offering help

Onboarding

Playing audio

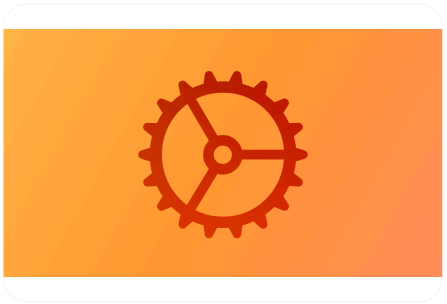
Playing haptics

Playing video

Printing

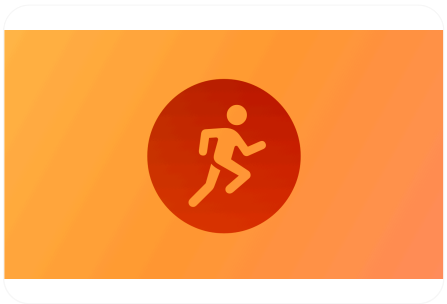
Ratings and reviews

Searching



Settings

Undo and redo

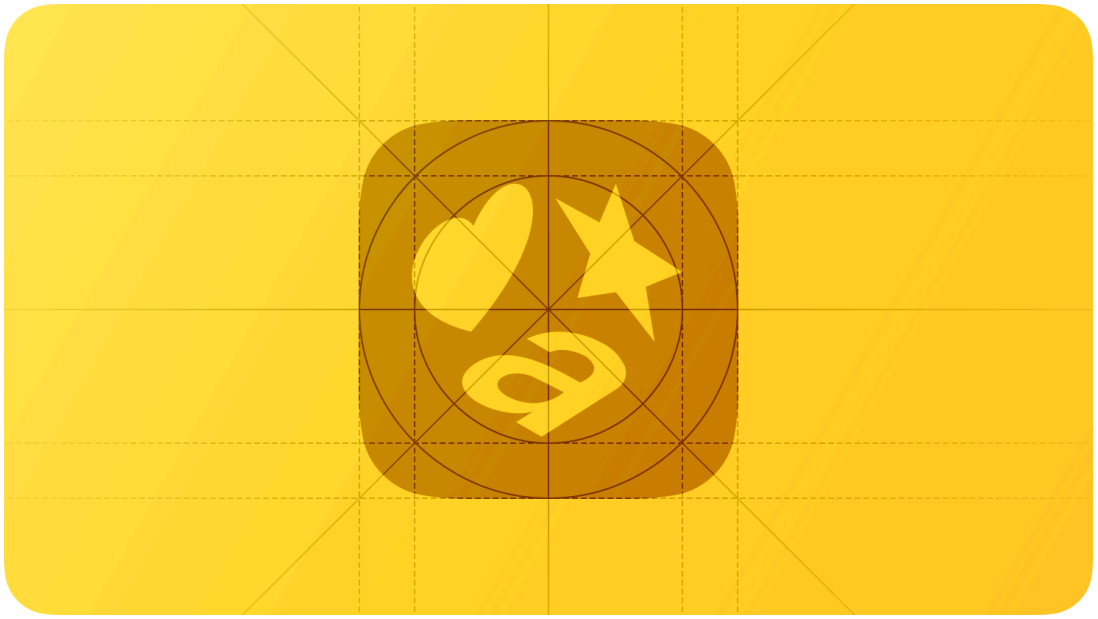


Workouts

SF Symbols

SF Symbols provides thousands of consistent, highly configurable symbols that integrate seamlessly with the San Francisco system font, automatically aligning with text in all weights and sizes.

Supported platforms



You can use a symbol to convey an object or concept wherever interface icons can appear, such as in navigation bars, toolbars, tab bars, context menus, and within text.

Availability of individual symbols and features varies based on the version of the system you're targeting. Symbols and symbol features introduced in a given year aren't available in earlier operating systems.

Visit SF Symbols to download the app and browse the full set of symbols. Be sure to understand the terms and conditions for using SF Symbols, including the prohibition against using symbols — or images that are confusingly similar — in app icons, logos, or any other trademarked use. For developer guidance, see Configuring and displaying symbol images in your UI.

Rendering modes

SF Symbols provides four rendering modes — monochrome, hierarchical, palette, and multicolor — that give you multiple options when applying color to symbols. For example, you might want to use multiple opacities of your app's accent color to give symbols depth and emphasis, or specify a palette of contrasting colors to display symbols that coordinate with various color schemes.

To support the rendering modes, SF Symbols organizes a symbol's paths into distinct layers. For example, the `cloud.sun.rain.fill` symbol consists of three layers: the primary layer contains the cloud paths, the secondary layer contains the paths that define the sun and its rays, and the tertiary layer contains the raindrop paths.

Primary

Secondary

Tertiary

Depending on the rendering mode you choose, a symbol can produce various appearances. For example, Hierarchical rendering mode assigns a different opacity of a single color to each layer, creating a visual hierarchy that gives depth to the symbol.

To learn more about supporting rendering modes in custom symbols, see Custom symbols.

SF Symbols supports the following rendering modes.

Monochrome — Applies one color to all layers in a symbol. Within a symbol, paths render in the color you specify or as a transparent shape within a color-filled path.

Hierarchical — Applies one color to all layers in a symbol, varying the color's opacity according to each layer's hierarchical level.

Palette — Applies two or more colors to a symbol, using one color per layer. Specifying only two colors for a symbol that defines three levels of hierarchy means the secondary and tertiary layers use the same color.

Multicolor — Applies intrinsic colors to some symbols to enhance meaning. For example, the `leaf` symbol uses green to reflect the appearance of leaves in the physical world, whereas the `trash.slash` symbol uses red to signal data loss. Some multicolor symbols include layers that can receive other colors.

Regardless of rendering mode, using system-provided colors ensures that symbols automatically adapt to accessibility accommodations and appearance modes like vibrancy and Dark Mode. For developer guidance, see renderingMode(_:).

Confirm that a symbol's rendering mode works well in every context. Depending on factors like the size of a symbol and its contrast with the current background color, different rendering modes can affect how well people can discern the symbol's details. You can use the automatic setting to get a symbol's preferred rendering mode, but it's still a good idea to check the results for places where a different rendering mode might improve a symbol's legibility.

Variable color

With variable color, you can represent a characteristic that can change over time — like capacity or strength — regardless of rendering mode. To visually communicate such a change, variable color applies color to different layers of a symbol as a value reaches different thresholds between zero and 100 percent.

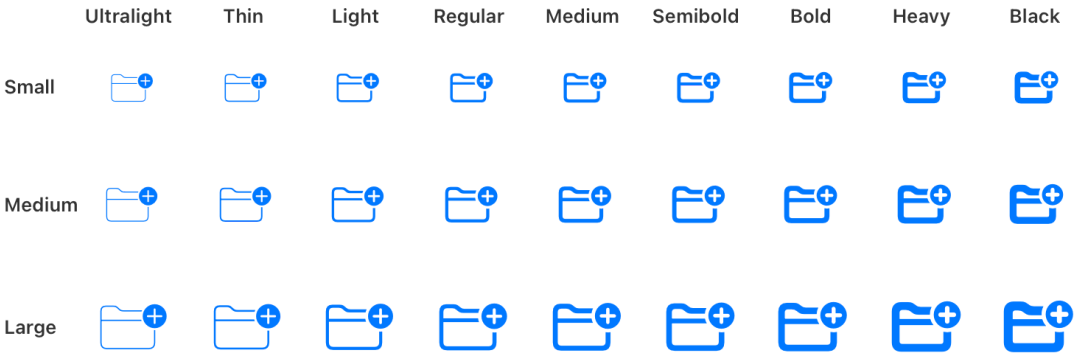
For example, you could use variable color with the `speaker.wave.3` symbol to communicate three different ranges of sound — plus the state where there's no sound — by mapping the layers that represent the curved wave paths to different ranges of decibel values. In the case of no sound, no wave layers get color. In all other cases, a wave layer receives color when the sound reaches a threshold the system defines based on the number of nonzero states you want to represent.

Sometimes, it can make sense for some of a symbol's layers to opt out of variable color. For example, in the `speaker.wave.3` symbol shown above, the layer that contains the speaker path doesn't receive variable color because a speaker doesn't change as the sound level changes. A symbol can support variable color in any number of layers.

Use variable color to communicate change — don't use it to communicate depth. To convey depth and visual hierarchy, use Hierarchical rendering mode to elevate certain layers and distinguish foreground and background elements in a symbol.

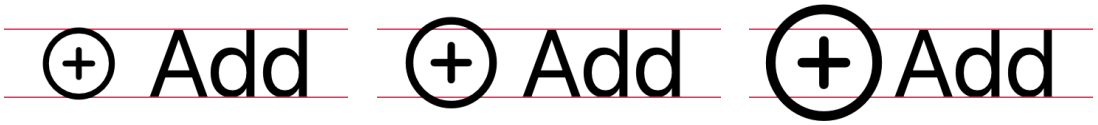
Weights and scales

SF Symbols provides symbols in a wide range of weights and scales to help you create adaptable designs.



Each of the nine symbol weights — from ultralight to black — corresponds to a weight of the San Francisco system font, helping you achieve precise weight matching between symbols and adjacent text, while supporting flexibility for different sizes and contexts.

Each symbol is also available in three scales: small, medium (the default), and large. The scales are defined relative to the cap height of the San Francisco system font.



Small Medium Large

Specifying a scale lets you adjust a symbol's emphasis compared to adjacent text, without disrupting the weight matching with text that uses the same point size. For developer guidance, see `imageScale(_:)` (SwiftUI), `UIImage.SymbolScale` (UIKit), and `NSImage.SymbolConfiguration` (AppKit).

Design variants

SF Symbols defines several design variants — such as fill, slash, and enclosed — that can help you communicate precise states and actions while maintaining visual consistency and simplicity in your UI. For example, you could use the slash variant of a symbol to show that an item or action is unavailable, or use the fill variant to indicate selection.

Outline is the most common variant in SF Symbols. An outlined symbol has no solid areas, resembling the appearance of text. Most symbols are also available in a fill variant, in which the areas within some shapes are solid.

In addition to outline and fill, SF Symbols also defines variants that include a slash or enclose a symbol within a shape like a circle, square, or rectangle. In many cases, enclosed and slash variants can combine with outline or fill variants.

SF Symbols provides many variants for specific languages and writing systems, including Latin, Arabic, Hebrew, Hindi, Thai, Chinese, Japanese, Korean, Cyrillic, Devanagari, and several Indic numeral systems. Language- and script-specific variants adapt automatically when the device language changes. For guidance, see Images.

Symbol variants support a range of design goals. For example:

- The outline variant works well in toolbars, navigation bars, lists, and other places where you display a symbol alongside text.
- Symbols that use an enclosing shape — like a square or circle — can improve legibility at small sizes.
- The solid areas in a fill variant tend to give a symbol more visual emphasis, making it a good choice for iOS tab bars and swipe actions and places where you use an accent color to communicate selection.

In many cases, the view that displays a symbol determines whether to use outline or fill, so you don't have to specify a variant. For example, an iOS tab bar prefers the fill variant, whereas a navigation bar takes the outline variant.

Animations

SF Symbols provides a collection of expressive, configurable animations that enhance your interface and add vitality to your app. Symbol animations help communicate ideas, provide feedback in response to people's actions, and signal changes in status or ongoing activities.

Animations work on all SF Symbols in the library, in all rendering modes, weights, and scales, and on custom symbols. For considerations when animating custom symbols, see Custom symbols. You can control the playback of an animation, whether you want the animation to run from start to finish, or run indefinitely, repeating its effect until a condition is met. You can customize behaviors, like changing the playback speed of an animation or determining whether to reverse an animation before repeating it. For developer guidance, see Symbols and `Symbol Effect`.

SF Symbols 5 and later support the following animations.

Appear — Causes a symbol to gradually emerge into view.

Play

Disappear — Causes a symbol to gradually recede out of view.

Play

Bounce — Briefly scales a symbol with an elastic-like movement that goes either up or down and then returns to the symbol's initial state. The bounce animation plays once by default and can help communicate that an action occurred or needs to take place.

Play

Scale — Changes the size of a symbol, increasing or decreasing its scale. Unlike the bounce animation, which returns the symbol to its original state, the scale animation persists until you set a new scale or remove the effect. You might use the scale animation to draw people's attention to a selected item or as feedback when people choose a symbol.

Play

Pulse — Varies the opacity of a symbol over time. This animation automatically pulses only the layers within a symbol that are annotated to pulse, and optionally can pulse all layers within a symbol. You might use the pulse animation to communicate ongoing activity, playing it continuously until a condition is met.

Play

Variable color — Incrementally varies the opacity of layers within a symbol. This animation can be cumulative or iterative. When cumulative, color changes persist for each layer until the animation cycle is complete. When iterative, color changes occur one layer at a time. You might use variable color to communicate progress or ongoing activity, such as playback, connecting, or broadcasting. You can customize the animation to autoreverse — meaning reverse the animation to the starting point and replay the sequence — as well as hide inactive layers rather than reduce their opacity.

The arrangement of layers within a symbol determines how variable color behaves during a repeating animation. Symbols with layers that are arranged linearly where the start and end points don't meet are annotated as *open loop*. Symbols with layers that follow a complete shape where the start and end points do meet, like in a circular progress indicator, are annotated as *closed loop*.

In SF Symbols 6 and later, variable color animations for symbols with closed loop designs feature seamless, continuous playback.

Play

Replace — Replaces one symbol with another. The replace animation works between arbitrary symbols and across all weights and rendering modes. This animation features three configurations:

- Down-up, where the outgoing symbol scales down and the incoming symbol scales up, communicating a change in state.
- Up-up, where both the outgoing and incoming symbols scale up. This configuration communicates a change in state that includes a sense of forward progression.
- Off-up, where the outgoing symbol hides immediately and the incoming symbol scales up. This configuration communicates a state change that emphasizes the next available state or action.

Play

From left to right: down-up, up-up, off-up

Magic Replace — In SF Symbols 6 and later, performs a smart transition between two symbols with related shapes. For example, slashes can draw on and off, and badges can appear or disappear, or you can replace them independently of the base symbol. Magic Replace is the new default replace animation, but doesn't occur between unrelated symbols; the default down-up animation occurs instead. You can choose a custom direction for the fallback animation in these situations if you prefer one other than the default.

Play

Additionally, SF Symbols 6 and later support the following animations.

Wiggle — Moves the symbol back and forth along a directional axis. You might use the wiggle animation to highlight a change or a call to action that a person might overlook. Wiggle can also add a dynamic emphasis to an interaction or reinforce what the symbol is representing, such as when an arrow points in a specific direction.

Play

Breathe — Smoothly increases and decreases the presence of a symbol, giving it a living quality. You might use the breathe animation to convey status changes, or signal that an activity is taking place, like an ongoing recording session. Breathe is similar to pulse; however pulse

animates by changing opacity alone, while breathe changes both opacity and size to convey ongoing activity.

Play

Rotate — Rotates the symbol to act as a visual indicator or imitate an object's behavior in the real world. For example, when a task is in progress, rotation confirms that it's working as expected. The rotate animation causes some symbols to rotate entirely, while in others only certain parts of the symbol rotate. Symbols like the desk fan, for example, use the By Layer rotation option to spin only the fan blades.

Play

Apply symbol animations judiciously. While there's no limit to how many animations you can add to a view, too many animations can overwhelm an interface and distract people.

Make sure that animations serve a clear purpose in communicating a symbol's intent. Each type of animation has a discrete movement that communicates a certain type of action or elicits a certain response. Consider how people might interpret an animated symbol and whether the animation, or combination of animations, might be confusing.

Use symbol animations to communicate information more efficiently. Animations provide visual feedback, reinforcing that something happened in your interface. You can use animations to present complex information in a simple way and without taking up a lot of visual space.

Consider your app's tone when adding animations. When animating a symbol, think about what the animation can convey and how that might align with your brand identity and your app's overall style and tone. For guidance, see Branding.

Custom symbols

If you need a symbol that SF Symbols doesn't provide, you can create your own. To create a custom symbol, first export the template for a symbol that's similar to the design you want, then use a vector-editing tool to modify it. For developer guidance, see Creating custom symbol images for your app.

> **Important**
>
> SF Symbols includes copyrighted symbols that depict Apple products and features. You can display these symbols in your app, but you can't customize them. To help you identify a noncustomizable symbol, the SF Symbols app badges it with an Info icon; to help you use the symbol correctly, the inspector pane describes its usage restrictions.

Using a process called *annotating*, you can assign a specific color — or a specific hierarchical level, such as primary, secondary, or tertiary — to each layer in a custom symbol. Depending on the rendering modes you support, you can use a different mode in each instance of the symbol in your app.

Use the template as a guide. Create a custom symbol that's consistent with the ones the system provides in level of detail, optical weight, alignment, position, and perspective. Strive to design a symbol that is:

- Simple
- Recognizable
- Inclusive
- Directly related to the action or content it represents

For guidance, see Icons.

Assign negative side margins to your custom symbol if necessary. SF Symbols supports negative side margins to aid optical horizontal alignment when a symbol contains a badge or other elements that increase its width. For example, negative side margins can help you horizontally align a stack of folder symbols, some of which include a badge. The name of each margin includes the relevant configuration — such as "left-margin-Regular-M" — so be sure to use this naming pattern if you add margins to your custom symbols.

Optimize layers to use animations with custom symbols. If you want to animate your symbol by layer, make sure to annotate the layers in the SF Symbols app. The Z-order determines the order that you want to apply colors to the layers of a variable color symbol, and you can choose whether to animate those changes from front-to-back, or back-to-front. SF Symbols 5 introduces layer groups, which allows related layers to move together when animated.

Test animations for custom symbols. It's important to test your custom symbols with all of the animation presets because the shapes and paths might not appear how you expect when the layers are in motion. To get the most out of this feature, consider drawing your custom symbols with whole shapes. For example, a custom symbol similar to the `person.2.fill` symbol doesn't need to create a cutout for the shape representing the person on the left. Instead, you can draw the full shape of the person, and in addition to that, draw an offset path of the person on the right to help represent the gap between them. You can later annotate this offset path as an erase layer to render the symbol as you want. This method of drawing helps preserve additional layer information that allows for animations to perform as you expect.

Avoid making custom symbols that include common variants, such as enclosures or badges. The SF Symbols app offers a component library for creating variants of your custom symbol. Using the component library allows you to create commonly used variants of your custom symbol while maintaining design consistency with the included SF Symbols.

Provide alternative text labels for custom symbols. Alternative text labels — or accessibility descriptions — let VoiceOver describe visible UI and content, making navigation easier for people with visual disabilities. For guidance, see VoiceOver.

Don't design replicas of Apple products. Apple products are copyrighted and you can't reproduce them in your custom symbols. Also, you can't customize a symbol that SF Symbols identifies as representing an Apple feature or product.

Platform considerations

No additional considerations for iOS, iPadOS, macOS, tvOS, visionOS, or watchOS.

Resources

Related

SF Symbols

Typography

Developer documentation

Configuring and displaying symbol images in your UI — UIKit

Creating custom symbol images for your app — UIKit

Symbols

Videos

What’s new in SF Symbols 6

What’s new in SF Symbols 5

Create animated symbols

Change log

Date	Changes
June 10, 2024	Updated with guidance for new animations and features of SF Symbols 6.
June 5, 2023	Added a new section on animations. Included animation guidance for custom symbols.
September 14, 2022	Added a new section on variable color. Removed instructions on creating custom symbol paths, exporting templates, and layering paths, deferring to developer articles that cover these topics.

Technologies

Discover the Apple technologies, features, and services you can integrate into your app or game.

AirPlay

Always On

App Clips

Apple Pay

Augmented reality

CareKit

CarPlay

Game Center

HealthKit

HomeKit

iCloud

ID Verifier

iMessage apps and stickers

In-app purchase

Live Photos

Mac Catalyst

Machine learning

Maps

Messages for Business

NFC

Photo editing

ResearchKit

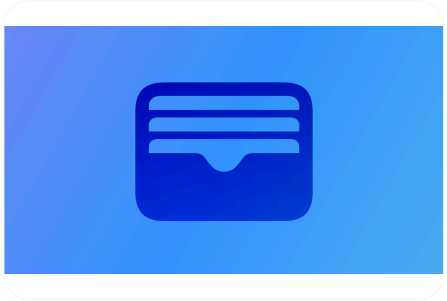
SharePlay

ShazamKit

Sign in with Apple

Siri

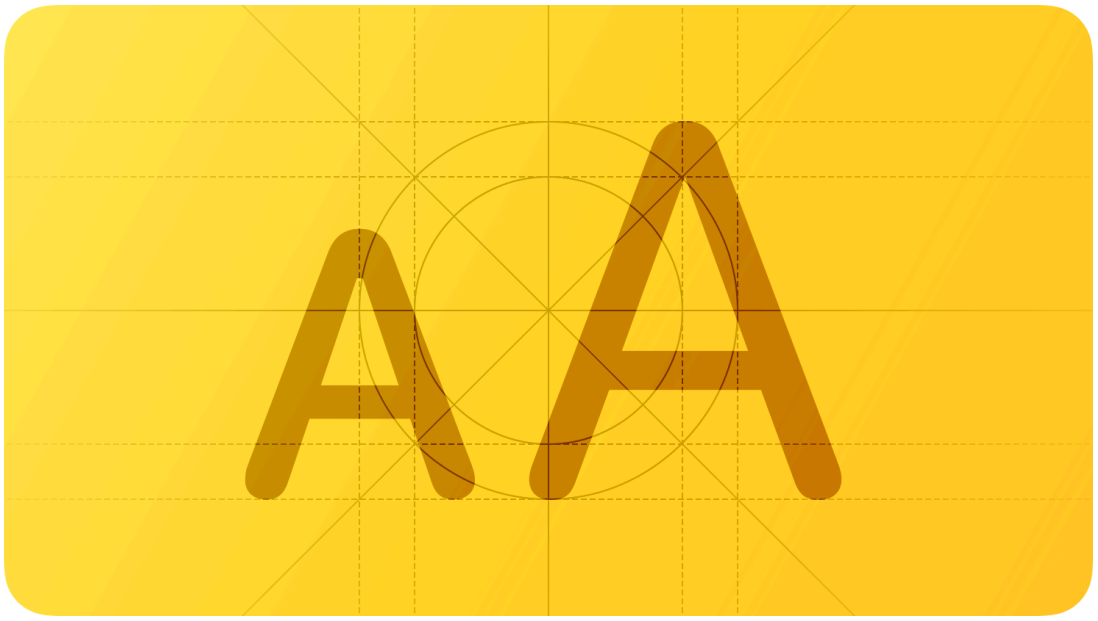
Tap to Pay on iPhone



Wallet

Typography

Your typographic choices can help you display legible text, convey an information hierarchy, communicate important content, and express your brand or style.



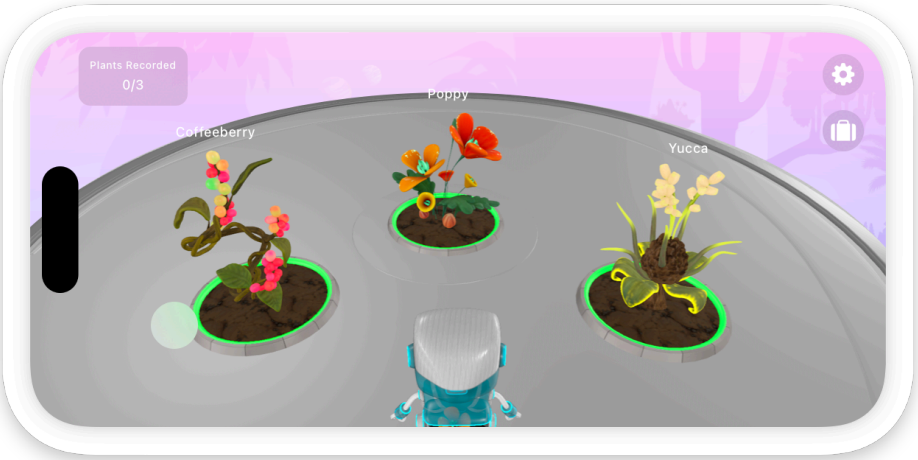
Supported platforms

Ensuring legibility

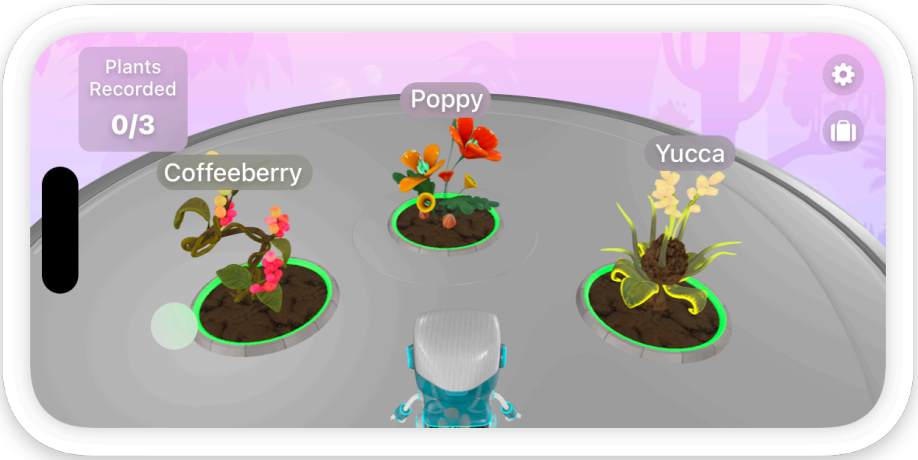
Use font sizes that most people can read easily. For example, aim to use the minimum font size of 11 pt in iOS and iPadOS, 10 pt in macOS, 23 pt in tvOS, and 12 pt in visionOS and watchOS. It's important to recognize that differences in device displays, including pixel density and brightness, in addition to other factors — such as ambient lighting, the reader's proximity to the text, their visual acuity, and whether they're in motion — can all impact legibility.

Support Dynamic Type. When you support Dynamic Type — a system feature that lets people choose the size of visible text on their device — your app or game can respond appropriately when people adjust text to a size that works for them. For available Dynamic Type sizes, see Specifications; for developer guidance, see Text input and output. To support Dynamic Type in your Unity-based game, use Apple's Unity plug-ins (for developer guidance, see Apple Unity Plug-Ins).

Test legibility in different contexts. For example, you need to test game text for legibility on each platform on which your game runs. If testing shows that some of your text is difficult to read, consider using a larger type size, increasing contrast by modifying the text or background colors, or using typefaces designed for optimized legibility, like the system fonts.



Testing a game on a new platform can show where text is hard to read.



Increasing text size and adding visible background shapes can help make text easier to read.

In general, avoid light font weights. For example, if you're using system-provided fonts, prefer Regular, Medium, Semibold, or Bold font weights, and avoid Ultralight, Thin, and Light font weights, which can be difficult to see, especially when text is small.

Conveying hierarchy

Adjust font weight, size, and color as needed to emphasize important information and help people visualize hierarchy. Be sure to maintain the relative hierarchy and visual distinction of text elements when people adjust text sizes.

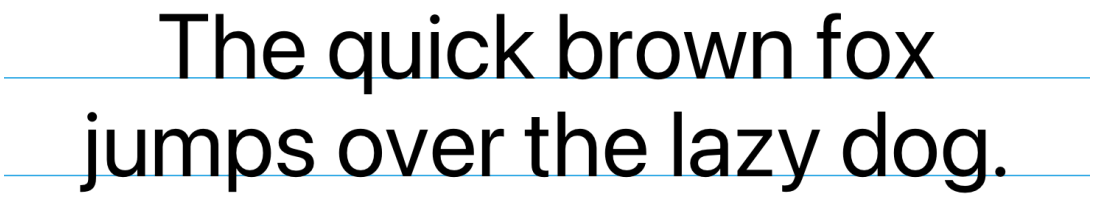
Minimize the number of typefaces you use, even in a highly customized interface. Mixing too many different typefaces can obscure your information hierarchy and hinder readability, in addition to making an interface feel internally inconsistent or poorly designed.

Prioritize important content when responding to text-size changes. Not all content is equally important. When someone chooses a larger text size, they typically want to make the content they care about easier to read; they don't always want to increase the size of every word on the screen. For example, when people increase text size to read the content in a tabbed window, they don't expect the tab titles to increase in size. Similarly, in a game, people are often more interested in a character's dialog than in transient hit-damage values.

Using system fonts

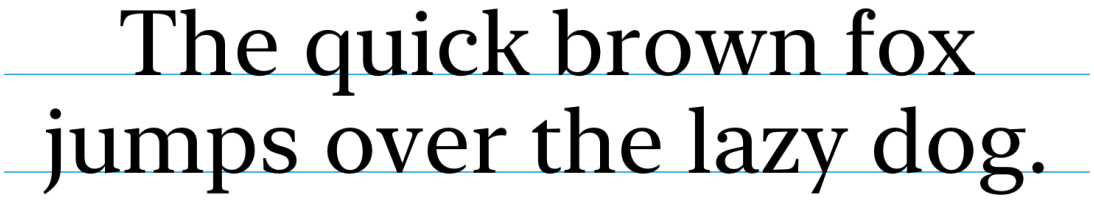
Apple provides two typeface families that support an extensive range of weights, sizes, styles, and languages.

San Francisco (SF) is a sans serif typeface family that includes the SF Pro, SF Compact, SF Arabic, SF Armenian, SF Georgian, SF Hebrew, and SF Mono variants.



The system also offers SF Pro, SF Compact, SF Arabic, SF Armenian, SF Georgian, and SF Hebrew in rounded variants you can use to coordinate text with the appearance of soft or rounded UI elements, or to provide an alternative typographic voice.

New York (NY) is a serif typeface family designed to work well by itself and alongside the SF fonts.



You can download the San Francisco and New York fonts here.

The system provides the SF and NY fonts in the *variable* font format, which combines different font styles together in one file, and supports interpolation between styles to create intermediate ones.

> **Note**
>
> Variable fonts support *optical sizing*, which refers to the adjustment of different typographic designs to fit different sizes. On all platforms, the system fonts support *dynamic optical sizes*, which merge discrete optical sizes (like Text and Display) and weights into a single, continuous design, letting the system interpolate each glyph or letterform to produce a structure that's precisely adapted to the point size. With dynamic optical sizes, you don't need to use discrete optical sizes unless you're working with a design tool that doesn't support all the features of the variable font format.

To help you define visual hierarchies and create clear and legible designs in many different sizes and contexts, the system fonts are available in a variety of weights, ranging from Ultralight to Black, and — in the case of SF — several widths, including Condensed and Expanded. Because SF Symbols use equivalent weights, you can achieve precise weight matching between symbols and adjacent text, regardless of the size or style you choose.

	Ultralight	Thin	Light	Regular	Medium	Semibold	Bold	Heavy	Black
Upright	Text	Text	Text	Text	Text	Text	Text	Text	Text
Italics	*Text*	*Text*	*Text*	*Text*	*Text*	*Text*	*Text*	*Text*	*Text*

> **Note**
>
> SF Symbols provides a comprehensive library of symbols that integrate seamlessly with the San Francisco system font, automatically aligning with text in all weights and sizes. Consider using symbols when you need to convey a concept or depict an object, especially within text.

The system defines a set of typographic attributes — called text styles — that work with both typeface families. A *text style* specifies a combination of font weight, point size, and leading values for each text size. For example, the *body* text style uses values that support a comfortable reading experience over multiple lines of text, while the *headline* style assigns a font size and weight that help distinguish a heading from surrounding content. Taken together, the text styles form a typographic hierarchy you can use to express the different levels of importance in your content. Text styles also allow text to scale proportionately when people change the system's text size or make accessibility adjustments, like turning on Larger Text in Accessibility settings.

Consider using the built-in text styles. The system-defined text styles give you a convenient and consistent way to convey your information hierarchy through font size and weight. Using text styles with the system fonts also supports Dynamic Type and the larger accessibility type sizes (where available), which let people choose the text size that works for them.

Modify the built-in text styles if necessary. System APIs define font adjustments — called *symbolic traits* — that let you modify some aspects of a text style. For example, the bold trait adds weight to text, letting you create another level of hierarchy. You can also use symbolic traits to adjust leading if you need to improve readability or conserve space. For example, when you display text in wide columns or long passages, more space between lines (*loose leading*) can make it easier for people to keep their place while moving from one line to the next. Conversely, if you need to display multiple lines of text in an area where height is constrained — for example, in a list row — decreasing the space between lines (*tight leading*) can help the text fit well. If you need to display three or more lines of text, avoid tight leading even in areas where height is limited. For developer guidance, see leading(_:).

> **Developer note**
>
> You can use the constants defined in `Font.Design` to access all system fonts — don't embed system fonts in your app or game. For example, use `Font.Design.default` to get the system font on all platforms; use `Font.Design.serif` to get the New York font.

If necessary, adjust tracking in interface mockups. In a running app, the system font dynamically adjusts tracking at every point size. To produce an accurate interface mockup of an interface that uses the variable system fonts, you don't have to choose a discrete optical size at certain point sizes, but you might need to adjust the tracking. For guidance, see Tracking values.

Using custom fonts

Make sure custom fonts are legible. People need to be able to read your custom font easily at various viewing distances and under a variety of conditions. While using a custom font, be guided by the recommended minimum font sizes for various styles and weights in Specifications.

Implement accessibility features for custom fonts. System fonts automatically support Dynamic Type (where available) and respond when people turn on accessibility features, such as Bold Text. If you use a custom font, make sure it implements the same behaviors. For developer guidance, see Applying custom fonts to text. In a Unity-based game, you can use Apple's Unity

plug-ins to support Dynamic Type. If the plug-in isn't appropriate for your game, be sure to let players adjust text size in other ways.

Platform considerations

iOS, iPadOS

SF Pro is the system font in iOS and iPadOS. iOS and iPadOS apps can also use NY.

macOS

SF Pro is the system font in macOS. NY is available for Mac apps built with Mac Catalyst. macOS doesn't support Dynamic Type.

When necessary, use dynamic system font variants to match the text in standard controls. Dynamic system font variants give your text the same look and feel of the text that appears in system-provided controls. Use the variants listed below to achieve a look that's consistent with other apps on the platform.

Dynamic font variant	API
Control content	`controlContentFont(ofSize:)`
Label	`labelFont(ofSize:)`
Menu	`menuFont(ofSize:)`
Menu bar	`menuBarFont(ofSize:)`
Message	`messageFont(ofSize:)`
Palette	`paletteFont(ofSize:)`
Title	`titleBarFont(ofSize:)`
Tool tips	`toolTipsFont(ofSize:)`
Document text (user)	`userFont(ofSize:)`
Monospaced document text (user fixed pitch)	`userFixedPitchFont(ofSize:)`
Bold system font	`boldSystemFont(ofSize:)`
System font	`systemFont(ofSize:)`

tvOS

SF Pro is the system font in tvOS, and apps can also use NY.

visionOS

SF Pro is the system font in visionOS. If you use NY, you need to specify the type styles you want.

visionOS uses bolder versions of the Dynamic Type body and title styles and it introduces Extra Large Title 1 and Extra Large Title 2 for wide, editorial-style layouts. For guidance using vibrancy to indicate hierarchy in text and symbols, see Materials > visionOS.

In general, prefer 2D text. The more visual depth text characters have, the more difficult they can be to read. Although a small amount of 3D text can provide a fun visual element that draws people's attention, if you're going to display content that people need to read and understand, prefer using text that has little or no visual depth.

Make sure text looks good and remains legible when people scale it. Use a text style that makes the text look good at full scale, then test it for legibility at different scales.

Maximize the contrast between text and the background of its container. By default, the system displays text in white, because this color tends to provide a strong contrast with the default system background material, making text easier to read. If you want to use a different text color, be sure to test it in a variety of contexts.

If you need to display text that's not on a background, consider making it bold to improve legibility. In this situation, you generally want to avoid adding shadows to increase text contrast. The current space might not include a visual surface on which to cast an accurate shadow, and you can't predict the size and density of shadow that would work well with a person's current Environment.

Keep text facing people as much as possible. If you display text that's associated with a point in space, such as a label for a 3D object, you generally want to use *billboarding* — that is, you want the text to face the wearer regardless of how they or the object move. If you don't rotate text to remain facing the wearer, the text can become impossible to read because people may view it from the side or a highly oblique angle. For example, imagine a virtual lamp that appears to be on a physical desk with a label anchored directly above it. For the text to remain readable, the label needs to rotate around the y-axis as people move around the desk; in other words, the baseline of the text needs to remain perpendicular to the person's line of sight.

watchOS

SF Compact is the system font in watchOS, and apps can also use NY. In complications, watchOS uses SF Compact Rounded.

Specifications

iOS, iPadOS Dynamic Type sizes

xSmall | Small | Medium | Large (default) | xLarge | xxLarge | xxxLarge

xSmall

Style	Weight	Size (points)	Leading (points)
Large Title	Regular	31	38
Title 1	Regular	25	31

Style	Weight	Size (points)	Leading (points)
Title 2	Regular	19	24
Title 3	Regular	17	22
Headline	Semibold	14	19
Body	Regular	14	19
Callout	Regular	13	18
Subhead	Regular	12	16
Footnote	Regular	12	16
Caption 1	Regular	11	13
Caption 2	Regular	11	13

Point size based on image resolution of 144 ppi for @2x and 216 ppi for @3x designs.

iOS, iPadOS larger accessibility type sizes

AX1 AX2 AX3 AX4 AX5

AX1

Style	Weight	Size (points)	Leading (points)
Large Title	Regular	44	52
Title 1	Regular	38	46
Title 2	Regular	34	41
Title 3	Regular	31	38
Headline	Semibold	28	34
Body	Regular	28	34
Callout	Regular	26	32
Subhead	Regular	25	31
Footnote	Regular	23	29
Caption 1	Regular	22	28
Caption 2	Regular	20	25

Point size based on image resolution of 144 ppi for @2x and 216 ppi for @3x designs.

macOS built-in text styles

Text style	Weight	Size (points)	Line height (points)	Emphasized weight
Large Title	Regular	26	32	Bold
Title 1	Regular	22	26	Bold
Title 2	Regular	17	22	Bold
Title 3	Regular	15	20	Semibold
Headline	Bold	13	16	Heavy
Body	Regular	13	16	Semibold
Callout	Regular	12	15	Semibold
Subheadline	Regular	11	14	Semibold
Footnote	Regular	10	13	Semibold
Caption 1	Regular	10	13	Medium
Caption 2	Medium	10	13	Semibold

Point size based on image resolution of 144 ppi for @2x designs.

tvOS built-in text styles

Text style	Weight	Size (points)	Leading (points)	Emphasized weight
Title 1	Medium	76	96	Bold
Title 2	Medium	57	66	Bold
Title 3	Medium	48	56	Bold
Headline	Medium	38	46	Bold
Subtitle 1	Regular	38	46	Medium
Callout	Medium	31	38	Bold
Body	Medium	29	36	Bold
Caption 1	Medium	25	32	Bold
Caption 2	Medium	23	30	Bold

Point size based on image resolution of 72 ppi for @1x and 144 ppi for @2x designs.

watchOS Dynamic Type sizes

xSmall Small Large xLarge xxLarge xxxLarge

xSmall

Style	Weight	Size (points)	Leading (points)
Large Title	Regular	30	32.5
Title 1	Regular	28	30.5
Title 2	Regular	24	26.5
Title 3	Regular	17	19.5
Headline	Semibold	14	16.5
Body	Regular	14	16.5
Caption 1	Regular	13	15.5
Caption 2	Regular	12	14.5
Footnote 1	Regular	11	13.5
Footnote 2	Regular	10	12.5

watchOS larger accessibility type sizes

AX1 AX2 AX3

AX1

Style	Weight	Size (points)	Leading (points)
Large Title	Regular	44	46.5
Title 1	Regular	42	44.5
Title 2	Regular	34	41
Title 3	Regular	24	26.5
Headline	Semibold	21	23.5
Body	Regular	21	23.5
Caption 1	Regular	18	20.5
Caption 2	Regular	17	19.5
Footnote 1	Regular	16	18.5
Footnote 2	Regular	15	17.5

Tracking values

iOS, iPadOS, visionOS tracking values

SF Pro SF Pro Rounded New York

SF Pro

Size (points)	Tracking (1/1000 em)	Tracking (points)
6	+41	+0.24
7	+34	+0.23
8	+26	+0.21
9	+19	+0.17
10	+12	+0.12
11	+6	+0.06
12	0	0.0
13	-6	-0.08
14	-11	-0.15
15	-16	-0.23
16	-20	-0.31
17	-26	-0.43
18	-25	-0.44
19	-24	-0.45
20	-23	-0.45
21	-18	-0.36
22	-12	-0.26
23	-4	-0.10
24	+3	+0.07
25	+6	+0.15
26	+8	+0.22
27	+11	+0.29
28	+14	+0.38
29	+14	+0.40
30	+14	+0.40
31	+13	+0.39
32	+13	+0.41
33	+12	+0.40
34	+12	+0.40
35	+11	+0.38
36	+10	+0.37
37	+10	+0.36
38	+10	+0.37

Size (points)	Tracking (1/1000 em)	Tracking (points)
39	+10	+0.38
40	+10	+0.37
41	+9	+0.36
42	+9	+0.37
43	+9	+0.38
44	+8	+0.37
45	+8	+0.35
46	+8	+0.36
47	+8	+0.37
48	+8	+0.35
49	+7	+0.33
50	+7	+0.34
51	+7	+0.35
52	+6	+0.33
53	+6	+0.31
54	+6	+0.32
56	+6	+0.30
58	+5	+0.28
60	+4	+0.26
62	+4	+0.24
64	+4	+0.22
66	+3	+0.19
68	+2	+0.17
70	+2	+0.14
72	+2	+0.14
76	+1	+0.07
80	0	0
84	0	0
88	0	0
92	0	0
96	0	0

Not all apps express tracking values as 1/1000 em. Point size based on image resolution of 144 ppi for @2x and 216 ppi for @3x designs.

macOS tracking values

Size (points)	Tracking (1/1000 em)	Tracking (points)
6	+41	+0.24
7	+34	+0.23
8	+26	+0.21
9	+19	+0.17
10	+12	+0.12
11	+6	+0.06
12	0	0.0
13	-6	-0.08
14	-11	-0.15
15	-16	-0.23
16	-20	-0.31
17	-26	-0.43
18	-25	-0.44
19	-24	-0.45
20	-23	-0.45
21	-18	-0.36
22	-12	-0.26
23	-4	-0.10
24	+3	+0.07
25	+6	+0.15
26	+8	+0.22
27	+11	+0.29
28	+14	+0.38
29	+14	+0.40
30	+14	+0.40
31	+13	+0.39
32	+13	+0.41
33	+12	+0.40
34	+12	+0.40
35	+11	+0.38
36	+10	+0.37
37	+10	+0.36

Size (points)	Tracking (1/1000 em)	Tracking (points)
38	+10	+0.37
39	+10	+0.38
40	+10	+0.37
41	+9	+0.36
42	+9	+0.37
43	+9	+0.38
44	+8	+0.37
45	+8	+0.35
46	+8	+0.36
47	+8	+0.37
48	+8	+0.35
49	+7	+0.33
50	+7	+0.34
51	+7	+0.35
52	+6	+0.31
53	+6	+0.33
54	+6	+0.32
56	+6	+0.30
58	+5	+0.28
60	+4	+0.26
62	+4	+0.24
64	+4	+0.22
66	+3	+0.19
68	+2	+0.17
70	+2	+0.14
72	+2	+0.14
76	+1	+0.07
80	0	0
84	0	0
88	0	0
92	0	0
96	0	0

Not all apps express tracking values as 1/1000 em. Point size based on image resolution of 144 ppi for @2x and 216 ppi for @3x designs.

tvOS tracking values

Size (points)	Tracking (1/1000 em)	Tracking (points)
6	+41	+0.24
7	+34	+0.23
8	+26	+0.21
9	+19	+0.17
10	+12	+0.12
11	+6	+0.06
12	0	0.0
13	-6	-0.08
14	-11	-0.15
15	-16	-0.23
16	-20	-0.31
17	-26	-0.43
18	-25	-0.44
19	-24	-0.45
20	-23	-0.45
21	-18	-0.36
22	-12	-0.26
23	-4	-0.10
24	+3	+0.07
25	+6	+0.15
26	+8	+0.22
27	+11	+0.29
28	+14	+0.38
29	+14	+0.40
30	+14	+0.40
31	+13	+0.39
32	+13	+0.41
33	+12	+0.40
34	+12	+0.40
35	+11	+0.38

Size (points)	Tracking (1/1000 em)	Tracking (points)
36	+10	+0.37
37	+10	+0.36
38	+10	+0.37
39	+10	+0.38
40	+10	+0.37
41	+9	+0.36
42	+9	+0.37
43	+9	+0.38
44	+8	+0.37
45	+8	+0.35
46	+8	+0.36
47	+8	+0.37
48	+8	+0.35
49	+7	+0.33
50	+7	+0.34
51	+7	+0.35
52	+6	+0.31
53	+6	+0.33
54	+6	+0.32
56	+6	+0.30
58	+5	+0.28
60	+4	+0.26
62	+4	+0.24
64	+4	+0.22
66	+3	+0.19
68	+2	+0.17
70	+2	+0.14
72	+2	+0.14
76	+1	+0.07
80	0	0
84	0	0
88	0	0
92	0	0

Size (points)	Tracking (1/1000 em)	Tracking (points)
96	0	0

Not all apps express tracking values as 1/1000 em. Point size based on image resolution of 144 ppi for @2x and 216 ppi for @3x designs.

watchOS tracking values

SF Compact | SF Compact Rounded

SF Compact

Size (points)	Tracking (1/1000 em)	Tracking (points)
6	+50	+0.29
7	+30	+0.21
8	+30	+0.23
9	+30	+0.26
10	+30	+0.29
11	+24	+0.26
12	+20	+0.23
13	+16	+0.20
14	+14	+0.19
15	+4	+0.06
16	0	0.00
17	-4	-0.07
18	-8	-0.14
19	-12	-0.22
20	0	0.00
21	-2	-0.04
22	-4	-0.09
23	-6	-0.13
24	-8	-0.19
25	-10	-0.24
26	-11	-0.28
27	-12	-0.30
28	-12	-0.34
29	-14	-0.38

Size (points)	Tracking (1/1000 em)	Tracking (points)
30	-14	-0.42
31	-15	-0.45
32	-16	-0.50
33	-17	-0.55
34	-18	-0.60
35	-18	-0.63
36	-20	-0.69
37	-20	-0.72
38	-20	-0.74
39	-20	-0.76
40	-20	-0.78
41	-20	-0.80
42	-20	-0.82
43	-20	-0.84
44	-20	-0.86
45	-20	-0.88
46	-20	-0.92
47	-20	-0.94
48	-20	-0.96
49	-21	-1.00
50	-21	-1.03
51	-21	-1.05
52	-21	-1.07
53	-22	-1.11
54	-22	-1.13
56	-22	-1.20
58	-22	-1.25
60	-22	-1.32
62	-22	-1.36
64	-23	-1.44
66	-24	-1.51
68	-24	-1.56
70	-24	-1.64

Size (points)	Tracking (1/1000 em)	Tracking (points)
72	-24	-1.69
76	-25	-1.86
80	-26	-1.99
84	-26	-2.13
88	-26	-2.28
92	-28	-2.47
96	-28	-2.62

Not all apps express tracking values as 1/1000 em. Point size based on image resolution of 144 ppi for @2x designs.

Resources

Related

Fonts

SF Symbols

Developer documentation

`Font` — SwiftUI

`UIFont` — UIKit

`NSFont` — AppKit

Videos

Meet the expanded San Francisco font family

Meet TextKit 2

The details of UI typography

Change log

Date	Changes
June 10, 2024	Added guidance for using Apple's Unity plug-ins to support Dynamic Type in a Unity-based game and enhanced guidance on billboarding in a visionOS app or game.
September 12, 2023	Added artwork illustrating system font weights, and clarified tvOS specification table descriptions.
June 21, 2023	Updated to include guidance for visionOS.

Writing

The words you choose within your app are an essential part of its user experience.

Supported platforms

Whether you're building an onboarding experience, writing an alert, or describing an image for accessibility, designing through the lens of language will help people get the most from your app or game.

Getting started

Determine your app's voice. Think about who you're talking to, so you can figure out the type of vocabulary you'll use. What types of words are familiar to people using your app? How do you want people to feel? The words for a banking app might convey trust and stability, for example, while the words in a game might convey excitement and fun. Create a list of common terms, and reference that list to keep your language consistent. Consistent language, along with a voice that reflects your app's values, helps everything feel more cohesive.

Match your tone to the context. Once you've established your app's voice, vary your tone based on the situation. Consider what people are doing while they're using your app — both in the physical world and within the app itself. Are they exercising and reached a goal? Or are they trying to make a payment and received an error? Situational factors affect both what you say and how you display the text on the screen.

Compare the tone of these two examples from Apple Watch. In the first, the tone is straightforward and direct, reflecting the seriousness of the situation. In the second, the tone is light and congratulatory.

Be clear. Choose words that are easily understood and convey the right thing. Check each word to be sure it needs to be there. If you can use fewer words, do so. When in doubt, read your writing out loud.

Write for everyone. For your app to be useful for as many people as possible, it needs to speak to as many people as possible. Choose simple, plain language and write with accessibility and localization in mind, avoiding jargon and gendered terminology. For guidance, see Writing inclusively and VoiceOver; for developer guidance, see Localization.

Best practices

Consider each screen's purpose. Pay attention to the order of elements on a screen, and put the most important information first. Format your text to make it easy to read. If you're trying to convey more than one idea, consider breaking up the text onto multiple screens, and think about the flow of information across those screens.

Be action oriented. Active voice and clear labels help people navigate through your app from one step to the next, or from one screen to another. When labeling buttons and links, it's almost always best to use a verb. Prioritize clarity and avoid the temptation to be too cute or clever with your labels. For example, just saying "Next" often works better than "Let's do this!" For links, avoid using "Click here" in favor of more descriptive words or phrases, such as "Learn more about UX Writing." This is especially important for people using screen readers to access your app.

Build language patterns. Consistency builds familiarity, helping your app feel cohesive, intuitive, and thoughtfully designed. It also makes writing for your app easier, as you can return to these patterns again and again. Here are a few language patterns to consider establishing:

- **Title or sentence case.** Decide whether you want to use title case or sentence case for alerts, page titles, headlines, button labels, and links. Throughout the HIG, you'll find guidelines for specific components, but how you format your text is a reflection of your app's voice. Title case is more formal, while sentence case is more casual. Choose a style that fits your app.
- **First or second person.** If your app allows people to save favorites or bookmark items, for example, those items could be found in "My Favorites" or "Your Saved Items," but don't use both. Choose first or second person and stick with it.
- **Continue or Next.** If your app has a flow that spans multiple screens, decide how you want to label the button or link that takes you from one step to the next. "Continue" or "Next" works well, but choose one and be consistent with how you use it. Make sure you indicate a difference at the end of the flow, like a button labeled "Done."

Write for how people use each device. People may use your app on several types of devices. While your language needs to be consistent across them, think about where it would be helpful to adjust your text to make it suitable for different devices. Make sure you describe gestures

correctly on each device — for example, not saying “click” for a touch device like iPhone or iPad where you mean “tap.”

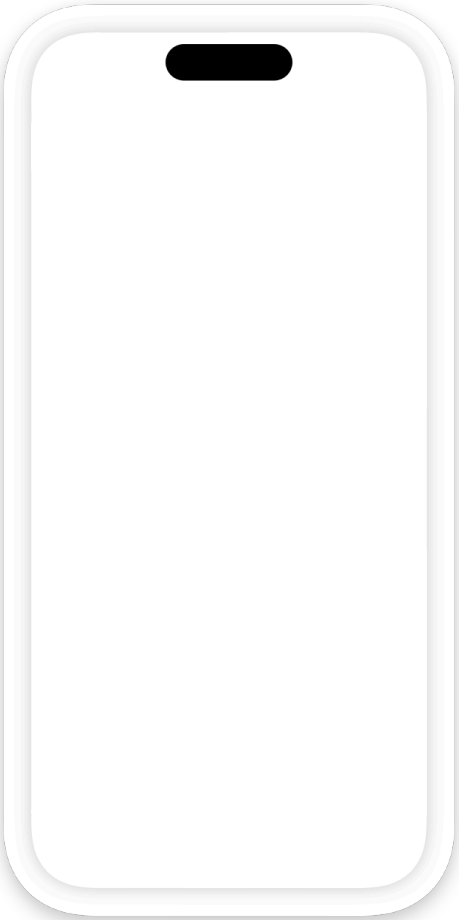
Where and how people use a device, its screen size, and its location all affect how you write for your app. iPhone and Apple Watch, for example, offer opportunities for personalization, but their small screens require brevity. TVs, on the other hand, are often in common living spaces, and several people are likely to see anything on the screen, so consider who you’re addressing. Bigger screens also require brevity, as the text must be large for people to see it from a distance.

Provide clear next steps on any blank screens. An empty state, like a completed to-do list or bookmarks folder with nothing in it, can provide a good opportunity to make people feel welcome and educate them about your app. Empty states can also showcase your app’s voice, but make sure that the content is useful and fits the context. An empty screen can be daunting if it isn’t obvious what to do next, so guide people on actions they can take, and give them a button or link to do so if possible. Remember that empty states are usually temporary, so don’t show crucial information that could then disappear.

Write clear error messages. It’s always best to help people avoid errors. When an error message is necessary, display it as close to the problem as possible, avoid blame, and be clear about what someone can do to fix it. For example, “That password is too short” isn’t as helpful as “Choose a password with at least 8 characters.” Remember that errors can be frustrating. Interjections like “oops!” or “uh-oh” are typically unnecessary and can sound insincere. If you find that language alone can’t address an error that’s likely to affect many people, use that as an opportunity to rethink the interaction.

Choose the right delivery method. There are many ways to get people’s attention, whether or not they are actively using your app. When there’s something you want to communicate, consider the urgency and importance of the message. Think about the context in which someone might see the message, whether it requires immediate action, and how much supporting information someone might need. Choose the correct delivery method, and use a tone appropriate for the situation. For guidance, see Notifications, Alerts, and Action sheets.

Keep settings labels clear and simple. Help people easily find the settings they need by labeling them as practically as possible. If the setting label isn’t enough, add an explanation. Describe what it does when turned on, and people can infer the opposite. In the Handwashing Timer setting for Apple Watch, for example, the description explains that a timer can start when you’re washing your hands. It isn’t necessary to tell you that a timer won’t start when this setting is off.



If you need to direct someone to a setting, provide a direct link or button, rather than trying to describe its location. For guidance, see Settings.

Show hints in text fields. If your app allows people to enter their own text, like account or contact information, label all fields clearly, and use hint or placeholder text so people know how to format the information. You can give an example in hint text, like "name@example.com," or describe the information, such as "Your name." Show errors right next to the field, and instruct people how to enter the information correctly, rather than scolding them for not following the rules. "Use only letters for your name" is better than "Don't use numbers or symbols." Avoid robotic error messages with no helpful information, like "Invalid name." For guidance, see Text fields.

Platform considerations

No additional considerations for iOS, iPadOS, macOS, tvOS, visionOS, or watchOS.

Resources

Related

Apple Style Guide

Writing inclusively

Inclusion

Accessibility

Color

Videos

Writing for interfaces

Change log

Date	Changes
February 27, 2023	New page.