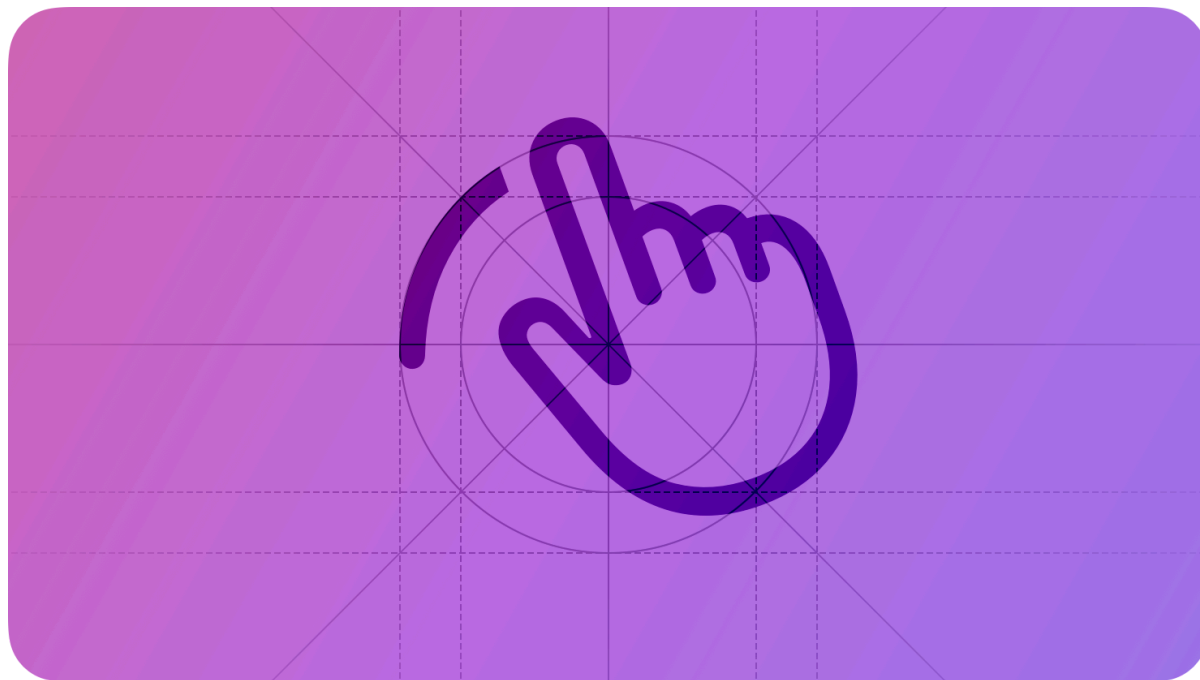**Design**

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

Gestures
Best practices
Custom gestures
Platform considerations
Specifications
Resources
Change log

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).
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.
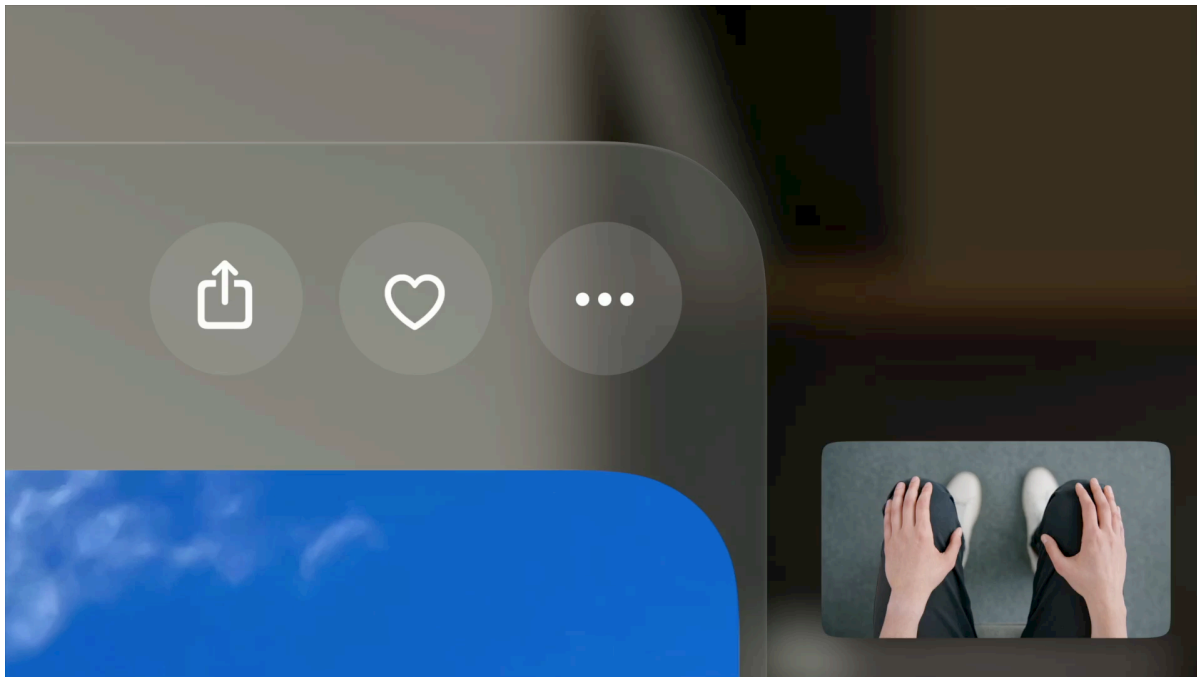
Play ⊙

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.

Direct gesture	Common use
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 `persistent SystemOverlays(_:)`.

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 underlined 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 `primary Action`.

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

Eyes

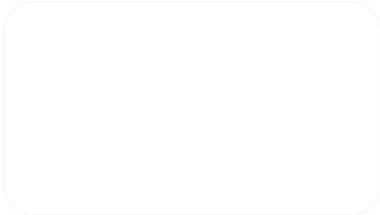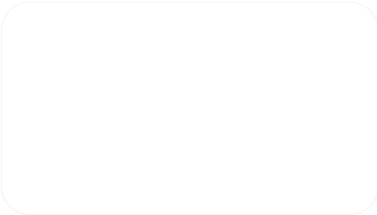# Developer documentation

`UITouch` — UIKit

Videos

Design for spatial input

Principles of spatial design
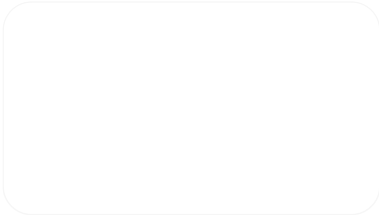
Design for spatial user interfaces

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.