Article

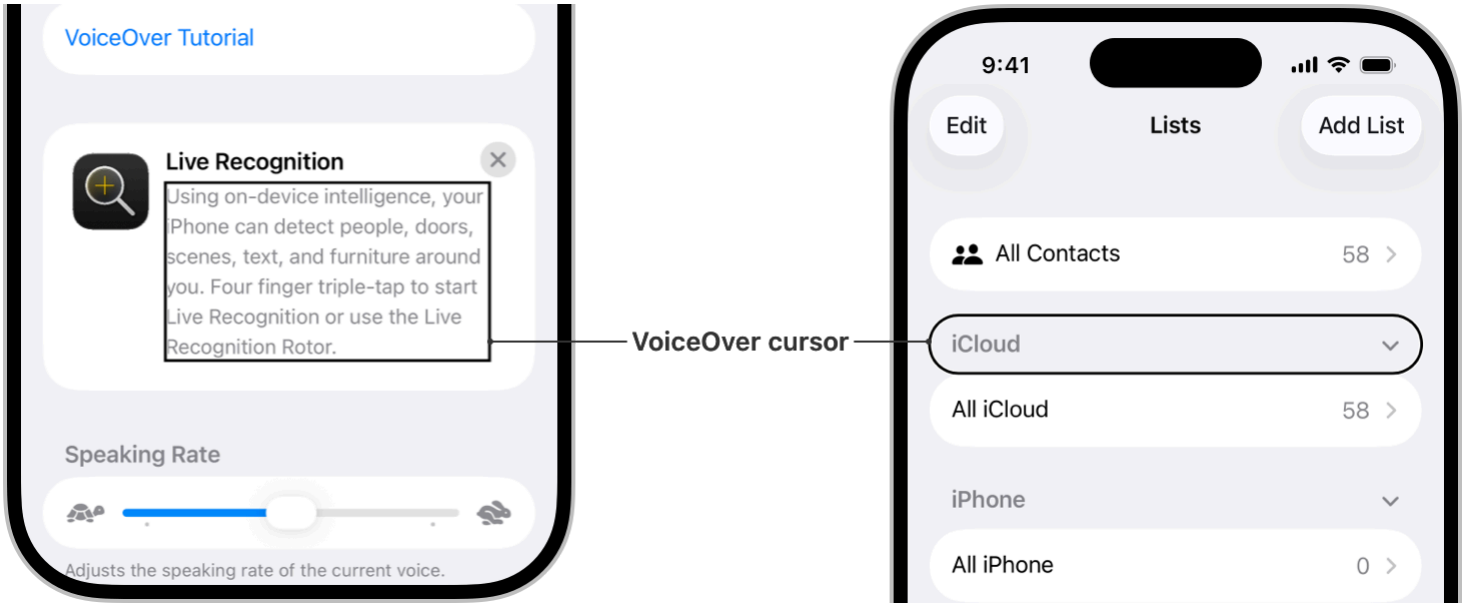
Supporting VoiceOver in your app

Add VoiceOver support to make your iOS app more accessible to users who are blind or have low vision.

Overview

VoiceOver is a gesture-based screen reader that enables people to experience the interface on their devices without having to see the screen. People who are blind depend on VoiceOver to provide auditory feedback while using their iOS devices, but VoiceOver isn't exclusively for users who are blind or have low vision. For example, someone prone to motion sickness might choose to turn VoiceOver on while they're in a moving vehicle. VoiceOver assists all types of people, but those who are blind rely on VoiceOver whenever they use their devices.

With a few steps, you can make your app VoiceOver-accessible in Xcode or programmatically. By increasing accessibility, you open your app to a wider audience, and make it easier for everyone to use.



Audit your app with VoiceOver on

To test the accessibility of your app, turn on VoiceOver and navigate through the interface. By using your app with VoiceOver on, you can establish a baseline of its accessibility. To start auditing, choose Settings > Accessibility > VoiceOver, and enable VoiceOver. Then, open your app and use the specified action whenever you want to test VoiceOver.

Auditing reveals which elements are accessible with VoiceOver and which aren't, and shows you if VoiceOver navigation is clear and logical. Keep track of which elements aren't accessible, and create a list of improvements for adding better VoiceOver support.

Navigate with VoiceOver on

To audit your app with VoiceOver on, you use VoiceOver's unique set of gestures to navigate your app. For testing, there are five key gestures that you may use:

- Swipe left or right to navigate to the next or previous UI element.
- One-finger double-tap to activate the selected element.
- Two-finger tap to stop and resume speaking.
- Swipe up with two fingers to read everything onscreen.
- Three-finger triple-tap to turn Screen Curtain on and off.

To learn more, see Operate iPhone using VoiceOver gestures.

To replicate the experience of someone who is solely relying on VoiceOver, test your app using Screen Curtain. As the name implies, Screen Curtain blacks out the entire screen. You can still navigate using VoiceOver gestures, but you can't see the elements on the screen.

Identify common accessibility issues

To audit your app, check that you can access every element and that the ordering of those elements is what you intend. Take note of which elements VoiceOver can or can't access. Also, pay attention when you find it difficult to perform a task because you assume it's reliant on visuals. For example, when you're navigating back to your app's initial view, or you're sharing content with another app or user, how can you use VoiceOver to make it accessible? While auditing your app, here are some common issues to look for:

- **Add accessibility information for your app's elements.** VoiceOver doesn't recognize custom UI elements by default. You need to add additional accessibility information to those elements.
- **Group elements so that VoiceOver navigates through them in the correct order.** VoiceOver reads from the leading to the trailing edge. If you want VoiceOver to read your elements in a

different order, use groups to facilitate navigation that makes sense for your app.

- **Include descriptive text for VoiceOver to read.** A UI that depends on visual cues may look nice, but it can be unusable for a VoiceOver user. For example, VoiceOver doesn't detect if a confirmation button turns from gray to green when the user selects it. VoiceOver may only describe the element and not it's current state. Make sure that VoiceOver says whether the button is in a selected state.

When you know which areas need improvements, start adding greater VoiceOver support to your app.

Update your app's accessibility

For elements that aren't accessible to VoiceOver, start by improving their accessibility labels and hints. The `accessibilityLabel` property provides descriptive text that VoiceOver reads when the user selects an element, and the `accessibilityHint` property provides additional context (or actions) for the selected element.

Accessibility labels are very important because they provide the text that VoiceOver reads. A good accessibility label is short and informative. It's important to note that a `UILabel` and an `accessibilityLabel` are different things. By default, VoiceOver reads the text for standard UIKit controls, such as `UILabel` and `UIButton`. However, these controls can also have corresponding `accessibilityLabel` properties to add more detail about the label or button.

Depending on the context, hints aren't always necessary. In some cases, the label provides enough context. If you feel like you're saying too much in an accessibility label, consider moving that text into a hint.

To ensure that users understand the intent of your interface, you might need to set some accessibility labels manually. You can set accessibility labels and hints in Xcode's Identity inspector, or programmatically.

Add accessibility labels and hints using the Identity inspector

When using standard UIKit controls, assign accessibility labels and hints in Xcode using the Identity inspector's Accessibility pane. To improve accessibility, you make an element accessible by selecting the Accessibility Enabled option. For example, the play button in a music app might include the following label and hint:

Add accessibility labels and hints programmatically

There are times when adding accessibility labels and hints in Xcode isn't enough, such as when you're working with custom UI elements that VoiceOver doesn't automatically recognize, or if you're using variables as part of your accessibility labels. In those situations, you need to set the accessibility labels or hints programmatically. You specify that an element is an accessibility element, and then create a corresponding accessibility label and hint.

To make your element accessible to VoiceOver programmatically, define it as an accessibility element.

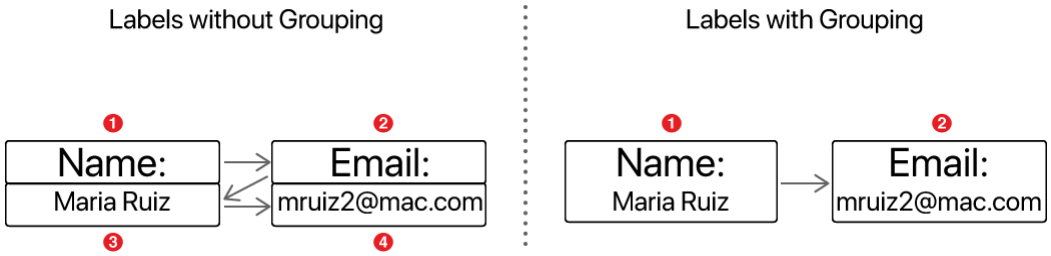
```
score.isAccessibilityElement = true
```

An element's label might not stay the same throughout the entire life cycle of your app. For example, for a counter that keeps score as you play a game, you want to change the label as the score changes. Do this programmatically by setting the accessibility label and hint.

```
score.accessibilityLabel = "score: \(currentScore)"
score.accessibilityHint = "Your current score"
```

Simplify your accessibility information

VoiceOver reads in the direction of the device's language. For example, VoiceOver reads English from left-to-right and it reads Arabic and Farsi from right-to-left. If you vertically stack labels in a UI, or display text in a table, VoiceOver may not read the labels in the correct order. You can programmatically group accessibility elements to ensure that VoiceOver reads them as you intend. For example, if you're creating an app that stacks a title and a value to display a person's name and email address, depending on their order in the interface, VoiceOver may not read those elements together. Group the elements to ensure that you're creating a clear context.



In the image above, there are four labels on the left that VoiceOver reads from the leading to the trailing edge, in this case, left-to-right. Although every element is accessible to VoiceOver, this doesn't provide the best user experience. On the right, VoiceOver reads the grouped labels in the intended order, which allows clear navigation.

To group the labels, create a `UIAccessibilityElement` and add the information you want to group together.

```
var elements = [UIAccessibilityElement]()
let groupedElement = UIAccessibilityElement(accessibilityContainer: self)
groupedElement.accessibilityLabel = "\(nameTitle.text!), \(nameValue.text!)"
groupedElement.accessibilityFrameInContainerSpace = nameTitle.frame.union(nameValue.
elements.append(groupedElement)
```

Including an accessibility label for each element and grouping elements allows people who depend on VoiceOver to navigate their devices and use your app.

See Also

Essentials

UIAccessibility

A set of methods that provides accessibility information about views and controls in an app's user interface.

UIAccessibilityContainer

Provide a set of methods that view subclasses use to make subcomponents accessible as separate elements.