Supporting Accessibility

On This Page

An accessible app is one that can be used by everyone—including those with a disability or physical impairment—while retaining its functionality and usability as a helpful tool. To be accessible, an iOS app must supply information about its user interface elements to VoiceOver, so that vision-impaired users can interact with those elements. UIKit objects are accessible by default, but there are still things you can do from the view controller's perspective to address accessibility, including the following:

- Ensure that every user element in your interface is accessible, including controls and static elements such as labels.
- Ensure that accessible elements supply accurate and helpful information.

You can enhance the VoiceOver user's experience in your app by setting the position of the VoiceOver focus ring programmatically, by responding to special VoiceOver gestures, and by observing accessibility notifications.

Moving the VoiceOver Cursor to a Specific Element

When your app presents new views onscreen, consider setting the position of the VoiceOver cursor. When the layout of a screen changes, the VoiceOver focus ring, also known as the *VoiceOver cursor*, resets its position to the first element displayed on the screen from left to right and top to bottom. Placing the cursor over a more appropriate element can speed the user's navigation of your interface. For example, when pushing a new view controller onto a navigation controller's stack, the VoiceOver cursor falls on the Back button of the navigation bar. You might want to move that cursor to the heading of the navigation bar or to an element on the newly pushed page.

To change the position of the cursor, post a `UIAccessibilityScreenChangedNotification` notification using the `UIAccessibilityPostNotification` function. The notification tells VoiceOver that the contents of the screen changed. When posting the notification, specify the element to which you want to assign the focus, as shown in Listing 6-1.

Listing 6-1 Posting an accessibility notification can change the first element read aloud

```
@implementation MyViewController
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];

    // The second parameter is the new focus element.
    UIAccessibilityPostNotification(UIAccessibilityScreenChangedNotification,
                                    self.myFirstElement);
}
@end
```

Layout changes, including those caused by rotations, reset the position of the VoiceOver cursor. When the layout of your view controller changes, post the notification `UIAccessibilityLayoutChangedNotification`. Like the `UIAccessibilityScreenChangedNotification` notification, you can specify the object that you want to become the new first element for VoiceOver.

Responding to Special VoiceOver Gestures

VoiceOver defines five special gestures for triggering app-specific actions.

- **Escape.** A two-finger Z-shaped gesture that dismisses a modal dialog, or goes back one level in a navigation hierarchy.
- **Magic Tap.** A two-finger double-tap that performs the most-intended action.
- **Three-Finger Scroll.** A three-finger swipe that scrolls content vertically or horizontally.
- **Increment.** A one-finger swipe up that increments a value in an element.
- **Decrement.** A one-finger swipe down that decrements a value in an element.

Use these gestures to perform tasks that are specific to your views and view controllers. UIKit looks for a method that implements the method corresponding to the gesture. It searches for the method using the responder chain, starting with the element that has the VoiceOver focus. If no object implements the appropriate method, UIKit performs the default system action for that gesture. For example, the Magic Tap gesture plays and pauses music playback from the Music app if no Magic Tap implementation is found from the current view to the app delegate.

On This Page

Although you can take any actions you want in your handlers, VoiceOver users expect your app's actions to follow certain guidelines. Like any gesture, your implementation of a VoiceOver gesture should follow a pattern so that interaction with an accessible app remains intuitive.

> NOTE
>
> All special VoiceOver gesture methods return a Boolean value that determine whether to propagate through the responder chain. To halt propagation, return `YES`; otherwise, return `NO`.

Escape

Use the `accessibilityPerformEscape` method to handle the Escape gesture. For a view that overlays content—such as a modal dialog or an alert—use the method to dismiss the overlay. The function of the Escape gesture is like the function of the `Esc` key on a computer keyboard; it cancels a temporary dialog or sheet to reveal the main content. You might also use the Escape gesture to navigate back one level in a custom navigation hierarchy. You do not need to implement this gesture if you are already using a `UINavigationController` object, which already handles this gesture.

Magic Tap

Use the `accessibilityPerformMagicTap` method to handle the Magic Tap gesture. The Magic Tap gesture performs an often-used or most-intended action quickly. For example, in the Phone app, a Magic Tap picks up or hangs up a phone call, and in the Clock app, a Magic Tap starts and stops the stopwatch. You might use this gesture to trigger actions that are not necessarily relevant to the element that the VoiceOver cursor is highlighting. To handle Magic Tap gestures from anywhere in your app, implement the `accessibilityPerformMagicTap` method in your app delegate.

Three-Finger Scroll

Use the `accessibilityScroll:` method to scroll the content of a custom view when a VoiceOver user performs a three-finger scroll gesture. A custom view that displays the pages of a book might use this gesture to turn the page. The parameter passed to the method indicates the direction in which to scroll.

Increment and Decrement

Use the `accessibilityIncrement` and `accessibilityDecrement` methods to increment or decrement a value in the element. Elements with the `UIAccessibilityTraitAdjustable` trait must implement this method.

Observing Accessibility Notifications

UIKit sends accessibility notifications to inform your app about relevant events. The objects of your app can observe any relevant notifications and use them to perform appropriate tasks. For example, the iBooks app uses the `UIAccessibilityAnnouncementDidFinishNotification` notification to turn the page and continue reading when VoiceOver finishes speaking the last line on a page. This behavior provides a seamless, uninterrupted reading experience.

Use the default notification center to register as an observer for accessibility notifications. Listing 6-2 shows an example of a view that records whether the reading of an announcement was successful or was interrupted by the user.

Listing 6-2 Registering as an observer for accessibility notifications

```
@implementation MyViewController
- (void)viewDidLoad
{
    [super viewDidLoad];

    [[NSNotificationCenter defaultCenter]
        addObserver:self
           selector:@selector(didFinishAnnouncement:)

               object:nil];
}

- (void)didFinishAnnouncement:(NSNotification *)dict
{
    NSString *valueSpoken = [[dict userInfo]
  objectForKey:UIAccessibilityAnnouncementKeyStringValue];
    NSString *wasSuccessful = [[dict userInfo]
  objectForKey:UIAccessibilityAnnouncementKeyWasSuccessful];
    // ...
}
@end
```

On This Page

Another helpful notification to subscribe to is the `UIAccessibilityVoiceOverStatusChanged` notification. You can use that notification to detect when VoiceOver is toggled on or off. If this notification occurs while your app is suspended, you receive the notification when your app returns to the foreground.

For a list of accessibility notifications you can observe, see *UIAccessibility Protocol Reference*.