Sample Code

Delivering an exceptional accessibility experience

Make improvements to your app's interaction model to support assistive technologies such as VoiceOver.

Download

iOS 12.0+ | iPadOS 12.0+ | Xcode 11.3+

Overview

To provide an exceptional accessibility experience, you need to think about how users with accessibility needs interact with your app. For example, your app should:

- Be easy to navigate using assistive technologies such as VoiceOver.
- Make the most frequently used features easily available to the user.
- Convey different elements in the proper context.
- Ensure the predictability of user interactions, which is to say, if a gesture works with VoiceOver one way in other apps, the gesture should work the same way in your app.

This sample code project shows how to provide an exceptional accessibility experience for VoiceOver. The sample app contains a gallery of dogs available for adoption. With VoiceOver enabled, the system reads aloud information about each dog as you swipe through the list. You'll also hear details about the dog—such as the age, gender, and attitude—mark the dog as a favorite, and view a gallery with more photos of the lovable canine.

Get started

To see the sample app in action, use Xcode to build and run the app on your iOS device. Be sure to turn on VoiceOver, which you can do from Settings under *General > Accessibility > VoiceOver*.

Improve carousel navigation

To preview each pup available for adoption, the user swipes left and right across the collection view at the top of the screen. There is, however, a problem with this interaction when VoiceOver is on, the assistive technology doesn't recognize visible elements, such as the favorite and gallery buttons, and it doesn't know the details of the dog until the user reaches the end of the list. This behavior is far from being an exceptional accessibility experience.

Navigating through the collection of dogs is central to the app. It's the primary feature that users use, and as such, it should be easy for the user to perform. To provide a better accessibility experience, the sample defines the custom class `CarouselAccessibilityElement` with its `accessibilityTraits` set to `adjustable`.

```
override var accessibilityTraits: UIAccessibilityTraits {
    get {
        return .adjustable
    }
    set {
        super.accessibilityTraits = newValue
    }
}
```

View in Source

The `adjustable` trait tells VoiceOver that the element behaves like a picker by responding to the `accessibilityIncrement()` and `accessibilityDecrement()` callback methods.

```
override func accessibilityIncrement() {
    // This causes the picker to move forward one if the user swipes up.
    _ = accessibilityScrollForward()
}

override func accessibilityDecrement() {
    // This causes the picker to move back one if the user swipes down.
    _ = accessibilityScrollBackward()
}
```

View in Source

The sample then creates an instance of CarouselAccessibilityElement, settings its frame to the collection view's frame.

```
let carouselAccessibilityElement: CarouselAccessibilityElement
if let theCarouselAccessibilityElement = self.carouselAccessibilityElement {
    carouselAccessibilityElement = theCarouselAccessibilityElement
} else {
    carouselAccessibilityElement = CarouselAccessibilityElement(
        accessibilityContainer: self,
        dog: currentDog
    )

    carouselAccessibilityElement.accessibilityFrameInContainerSpace = dogCollectionV
    self.carouselAccessibilityElement = carouselAccessibilityElement
}
```

View in Source

With VoiceOver on, the carousel accessibility element adds two new gestures on top of the collection view:

- Swipe up to move forward through the collection.
- Swipe down to move backward through the collection.

The sample's use of CarouselAccessibilityElement also makes it possible for the user to swipe to the favorite and gallery buttons for every dog in the list, rather than having the buttons reachable only at the end of the list. Although the gestures to swipe through the list of dogs is different when VoiceOver is on, this difference makes it easier for users needing assistance to preview each dog.

Display non-modal modal views

When the user taps the gallery button, the app displays a full-screen modal view. However, in the sample this modal view isn't from a view controller that the app presents modally. Instead, it's a view with a transparent background displayed on top of the app's main view. Because of the transparent background, parts of the underlying view are still available to VoiceOver, and VoiceOver doesn't know that those visible parts should not be available.

To tell VoiceOver that the underlying view is not available, and to treat the top view as a modal view, the top view returns `true` for the accessibilityViewIsModal property.

```
override var accessibilityViewIsModal: Bool {
    get {
        return true
    }

    set {}
}
```

View in Source

Group labels

When referring to separate elements belonging in the same context, it's essential to convey a single set of information about the items. For example, the sample app displays a title label, such as *NAME*, and a value label, such as the dog's name. These are two separate elements—*title* and *value*—but they are part of the same context.

To place these two elements into the same accessibility context, use `UIAccessibilityElement` to encapsulate the information from each label. For example, the sample project's `DogStatsView` groups each *title* and *value* label combination into accessibility elements. The project also sets the `accessibilityLabel` with the text from the two `UILabel` objects, and sets `accessibilityFrameInContainerSpace` to a frame containing both labels. This allows VoiceOver to recognize each *title-value* label pairing as a single accessibility element. In turn, the user hears the title and value each time a *title-value* accessibility element has focus.

```
var elements = [UIAccessibilityElement]()
let nameElement = UIAccessibilityElement(accessibilityContainer: self)
nameElement.accessibilityLabel = "\(nameTitleLabel.text!), \(nameLabel.text!)"

/*
    This tells VoiceOver where the object should be onscreen. As the user
    touches around with their finger, we can determine if an element is below
    their finger.
*/
nameElement.accessibilityFrameInContainerSpace = nameTitleLabel.frame.union(nameLabe
elements.append(nameElement)
```

View in Source

Add custom actions

The sample app also displays the name of the animal shelter housing the dog. Next to the shelter name are two buttons: one initiates a call to the shelter and the other displays the location of the shelter. To make the buttons available to VoiceOver users, the app uses `UIAccessibilityCustomAction` to specify a name for the action along with the object and selector to use when performing the action. With the custom action, VoiceOver users can access the action using the VoiceOver rotor.

```
shelterInfoView.accessibilityCustomActions = [
    UIAccessibilityCustomAction(
        name: "Call",
        target: self,
        selector: #selector(activateCallButton)
    ),
    UIAccessibilityCustomAction(
        name: "Open address in Maps",
        target: self,
        selector: #selector(activateLocationButton)
    )
]
```

View in Source

See Also

Sample code

{} Enhancing the accessibility of your SwiftUI app

Support advancements in SwiftUI accessibility to make your app accessible to everyone.

{} Creating accessible views

Make your app accessible to everyone by applying accessibility modifiers to your SwiftUI views.

{} Integrating accessibility into your app

Make your app more accessible to users with disabilities by adding accessibility features.

{} Accessibility design for Mac Catalyst

Improve navigation in your app by using keyboard shortcuts and accessibility containers.