Sample Code

Integrating accessibility into your app

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

Download

macOS 13.1+ | Xcode 14.2+

Overview

By adding accessibility features to your app, you make it available to a wider range of users. This sample code project shows how to implement accessibility for several common UI controls. The examples make the controls accessible by using accessibility properties, accessibility protocols, and the `NSAccessibilityElement` class.

In macOS 10.10, the Accessibility API moved to a protocol-based approach, in contrast to the key-based API from macOS 10.9 and earlier. With the protocol-based API, you can:

- Simplify the implementation of accessibility
- More closely align macOS accessibility features with those of iOS
- Ensure compatibility with existing macOS apps and code

Accessibility API features in macOS versions earlier than 10.10 are deprecated, but can coexist with the current API. No changes are required for apps or accessibility clients that use earlier versions of the Accessibility API. If you implement both APIs on the same class, the current API takes precedence for that class. For cell-based controls, you need to provide Accessibility API implementations on the cell.

Add accessibility attributes

Most accessibility attributes from macOS 10.9 and earlier are now properties in the following AppKit classes:

- `NSApplication`
- `NSWindow`
- `NSView`
- `NSDrawer`
- `NSPopover`
- `NSCell`

To set an accessibility attribute value on an instance of one of these classes (or a subclass), simply assign the value to the property.

```
button.setAccessibilityLabel(NSLocalizedString("My label", comment: "label to use fo
```

View in Source

Alternatively, you can override the getter in the subclass's implementation.

```
override func accessibilityLabel() -> String? {
    return NSLocalizedString("Play", comment: "accessibility label of the Play butto
}
```

View in Source

`NSAccessibility` contains the full list of accessibility properties.

Add accessibility protocols to custom controls

The Accessibility API protocols define the required accessibility functions for many common accessibility elements. Conformance to an accessibility protocol isn't required to use the API, but it's recommended when making custom controls accessible. Conforming to an accessibility protocol results in a warning for each unimplemented required function and allows the system to automatically infer the `accessibilityRole` and `isAccessibilityElement` properties.

Standard AppKit controls conform to the related accessibility protocol (for example, `NSButton` conforms to the `NSAccessibilityButton` protocol, and `NSSlider` conforms to the `NSAccessibilitySlider` protocol). Whenever possible, subclass from the appropriate AppKit control to leverage the built-in accessibility.

To add accessibility to a custom control:

1. Conform to the appropriate protocol.
2. Implement all the required functions. A warning appears for each unimplemented required function.
3. Test using VoiceOver and the Accessibility Inspector.

For example, the following code sample creates a custom control that subclasses `NSView` and draws and behaves like a button:

```
class CustomButtonView: NSView {
```

View in Source

If a custom control doesn't conform to an accessibility protocol, you need to implement the `accessibilityRole` and `isAccessibilityElement` functions.

```
override func accessibilityRole() -> NSAccessibility.Role? {
    return NSAccessibility.Role.button
}

override func isAccessibilityElement() -> Bool {
    return true
}
```

View in Source

Create an accessibility element

For objects that don't have a backing view — for example, a single view that draws several images, each of which is individually accessible — create an instance of `NSAccessibilityElement` for each object, and return an array of the instances from the containing view's `accessibilityChildren` function.

Simplify your accessibility code

`NSAccessibilityElement` has two convenience methods that simplify its use:

- `accessibilityAddChildElement` — This function sets the specified element as a subelement of the receiver's `accessibilityChildren` and the receiver as the container to the specified element. This behavior is useful when you create hierarchies of accessibility elements.

- `accessibilityFrameInParentSpace` — This property allows the accessibility element to specify its frame relative to its accessibility container, so that the system can automatically recalculate the `accessibilityFrame` property value (given in screen coordinates) whenever the element or any of its containing views changes location.

The Accessibility API includes two convenience methods in `AppKit/NSAccessibility.h` to simplify common accessibility tasks:

- `NSAccessibilityFrameInView` — This convenience method converts `frame` from the `parentView` coordinate space to the screen coordinate space. This is useful when you set an object's accessibility frame using the `accessibilityFrame()` method.
- `NSAccessibilityPointInView` — This convenience method converts `point` from the `parentView` coordinate space to the screen coordinate space. This is useful when you calculate an object's `accessibilityActivationPoint` coordinates.

Test the accessibility features on your app

The Accessibility Inspector is an Xcode tool that displays all accessibility information for the element currently beneath the cursor, including the accessibility hierarchy, accessibility attributes, and accessibility actions. It also shows warnings for common accessibility problems, such as a missing accessibility label. Launch the Accessibility Inspector from the Xcode > Open Developer Tool menu.

To test VoiceOver, choose System Settings > Accessibility > VoiceOver and click the toggle to enable it, or press Command-F5. To learn how to use VoiceOver, choose System Settings > Accessibility > VoiceOver > Open VoiceOver Training.

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.

{} Delivering an exceptional accessibility experience

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

{ } Accessibility design for Mac Catalyst

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