Sample Code

Restoring your app's state

Provide continuity for the user by preserving current activities.

Download

iOS 15.0+ | iPadOS 15.0+ | Xcode 13.0+

Overview

This sample project demonstrates how to preserve your app's state information and restore the app to that previous state on subsequent launches. During a subsequent launch, restoring your interface to the previous interaction point provides continuity for the user, and lets them finish active tasks quickly.

When using your app, the user performs actions that affect the user interface. For example, the user might view a specific page of information, and after the user leaves the app, the operating system might terminate it to free up the resources it holds. The user should be able to return to where they left off — and UI state restoration is a core part of making that experience seamless.

This sample app demonstrates the use of state preservation and restoration for scenarios where the system interrupts the app. The sample project manages a set of products. Each product has a title, an image, and other metadata you can view and edit. The project shows how to preserve and restore a product in its `DetailParentViewController`.

The sample supports two state preservation approaches. In iOS 13 and later, apps save the state for each window scene using `NSUserActivity` objects. In iOS 12 and earlier, apps preserve the state of their user interfaces by saving and restoring the configuration of view controllers.

For scene-based apps, UIKit asks each scene to save its state information using an `NSUserActivity` object. `NSUserActivity` is a core part of modern state restoration with `UIScene` and `UISceneDelegate`. In your own apps, you use the activity object to store information needed to recreate your scene's interface and restore the content of that interface. If your app

doesn't support scenes, use the view-controller-based state restoration process to preserve the state of your interface instead.

For additional information about state restoration, see Preserving your app's UI across launches.

Configure the sample code project

In Xcode, select your development team on the iOS target's General tab.

Enable state preservation and restoration

To provide the necessary activity object, the sample implements the `stateRestorationActivity(for:)` method of its scene delegate as shown in the example below. Implementing this method tells the system that the sample supports user-activity-based state restoration. The implementation of this method returns the activity object from the scene's `userActivity` property, which the sample populates when the scene becomes inactive.

```
func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
    return scene.userActivity
}
```

For view-controller-based state restoration, this sample opts in to state preservation and restoration using the app delegate's `application(_:shouldSaveApplicationState:)` and `application(_:shouldRestoreApplicationState:)` methods. Both methods return a `Bool` value that indicates whether the step should occur. This sample returns `true` for both functions.

This example enables the preservation of state for the app:

```
func application(_ application: UIApplication, shouldSaveSecureApplicationState code
    return true
}
```

This example enables the restoration of state for the app:

```
func application(_ application: UIApplication, shouldRestoreSecureApplicationState c
    return true
}
```

Restore the app state with an activity object

Scene-based state restoration is the recommended way to restore the app's user interface. An `NSUserActivity` object captures the app's state at the current moment in time. For this sample, the app preserves and restores the product information as the user displays or edits it. The sample app saves the product's data in an `NSUserActivity` object when the user closes the app or the app enters the background. When the user launches the app again, the sample's `scene(_:willConnectTo:options:)` method checks for the presence of an activity object. If one is present, the method configures the detail view controller that the activity object specifies.

Restore the app state using view controllers

This sample preserves its state by saving the state of its view controller hierarchy. View controllers adopt the `UIStateRestoring` protocol, which defines methods for saving custom state information to an archive and restoring that information later.

The sample specifies which of its view controllers to save, and assigns a restoration identifier to that view controller. A restoration identifier is a string that UIKit uses to identify a view controller or other user interface element. The identifier for each view controller must be unique. The sample assigns the identifiers in Interface Builder, but this can also occur in code.

The sample assigns a restoration ID for each view controller in the storyboard file. This information is available by selecting the view controller and looking at the Identity Inspector. The Storyboard ID for that view controller is usually the same as the Restoration ID.

This sample saves the state information in the detail view controller's `encodeRestorableState(with:)` method, and it restores that state in the `restoreState(with:)` method. Because it already encapsulates the view controller's state in an `NSUserActivity` object, the implementations of these methods operate on the existing activity object. The sample then calls the required `super` object from these methods, which allows UIKit to restore the rest of the view controller's inherited state.

This example enables the state preservation of `InfoViewController`:

```
override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)

    coder.encode(product?.identifier.uuidString, forKey: InfoViewController.restoreP
}
```

This example enables the state restoration of `InfoViewController`:

```
override func decodeRestorableState(with coder: NSCoder) {
    super.decodeRestorableState(with: coder)

    guard let decodedProductIdentifier =
        coder.decodeObject(forKey: InfoViewController.restoreProductKey) as? String
        fatalError("A product did not exist in the restore. In your app, handle this
    }
    product = DataModelManager.sharedInstance.product(fromIdentifier: decodedProduct
}
```

Test state restoration on a device

This sample restores the following user interface:

- Detail View Controller — Tap a product in the collection view to open its detail information. The app restores the selected product and selected tab.
- Detail View Controller's Edit State — In the detail view, tap Edit. The app restores the edit view and its content.
- Secondary Window — (iPad only) Drag a product from the collection view over to the left or right of the device screen to create a second scene window. The app restores that scene and its product. Another way to create the secondary window is to tap and hold a product from the collection view through its contextual menu and select Open New Window.

When debugging the sample project, the system automatically deletes its preserved state when the user force quits the app. Deleting the preserved state information is a safety precaution. In addition, the system also deletes the preserved state if this app crashes at launch time. To test the sample app's ability to restore the sample's state, do not use the app switcher to force quit it during debugging. Instead, use Xcode to stop the app, or stop the app programmatically. One technique is to suspend the sample app using the Home button, and then stop the debugger in Xcode. Launch the sample app again using Xcode, and UIKit initiates the state restoration process.

See Also

Interface restoration

{} Restoring your app's state with SwiftUI

Provide app continuity for users by preserving their current activities.

Preserving your app's UI across launches

Return your app to its previous state after the system terminates it.

`protocol` `UIViewControllerRestoration`

The methods that objects adopt so that they can act as a restoration class for view controllers during state restoration.

`protocol` `UIObjectRestoration`

The interface that restoration classes use to restore preserved objects.

`protocol` `UIStateRestoring`

Methods for adding objects to your state restoration archives.