Sample Code

Supporting multiple windows on iPad

Support side-by-side instances of your app's interface and create new windows.

Download

iOS 15.0+ | iPadOS 15.0+ | Mac Catalyst 15.0+ | Xcode 13.3+

Overview

This sample shows how to create multiple windows that give users the ability to create separate parts of your application with similar or varying content. Windows are managed by a scene or `UISceneSession` class. Your application uses `UISceneDelegate` and `UIScene Configuration` to manage the life cycle of a window. Scenes have their own dedicated lifecycle that are managed separate from `UIApplication`.

When you adopt a multi-window architecture, the `UIApplicationDelegate` class that manages your application manages newly created scenes. Then, `UISceneDelegate` replaces the code in the delegate functions of `UIApplicationDelegate`.

UIKit provides a subclass of `UISceneDelegate` called `UIWindowSceneDelegate` designed specifically to help manage your windows. When adopting the multi-window architecture in an existing application running on iOS 12 or earlier, you move more responsibility from `UIApplicationDelegate` to `UIWindowSceneDelegate`.

For more information on multiple windows in an iPadOS app, refer to Human Interface Guidelines for iOS.

Note

This sample code project was discussed and demonstrated at WWDC 2019 session 212: Introducing Multiple Windows on iPad.

Configure the sample code project

In Xcode, select your development team on the iOS target's Signing & Capabilities tab.

Add multiple scene support

To support multiple windows, the app's `Info.plist` requires a manifest or `UIApplicationSceneManifest`, which contains information about the app's scene-based life-cycle support. The presence of this key indicates that the app supports scenes and doesn't use an app delegate object to manage transitions to and from the foreground or background. Include the key `UIApplicationSupportsMultipleScenes`, with its Boolean value set to `true`, which indicates that the app support two or more scenes simultaneously.

Add a scene delegate

This sample provides a `UIWindowScene` subclass called `SceneDelegate` to manage the app's primary window scene. The `scene(_:willConnectTo:options:)` delegate function sets up the window and content.

```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connecti
    if let userActivity = connectionOptions.userActivities.first ?? session.stateRes
        if !configure(window: window, with: userActivity) {
            Swift.debugPrint("Failed to restore from \(userActivity)")
        }
    }
    // The 'window' property will automatically be loaded with the storyboard's init

    // Set the activation predicates, which operate on the 'targetContentIdentifier'
    let conditions = scene.activationConditions
    let prefsPredicate = NSPredicate(format: "self == %@", mainSceneTargetContentIde
    // The main predicate, which expresses to the system what kind of content the sc
    conditions.canActivateForTargetContentIdentifierPredicate = prefsPredicate
    // The secondary predicate, which expresses to the system that this scene is esp
    conditions.prefersToActivateForTargetContentIdentifierPredicate = prefsPredicate
}
```

By using the `UISceneConfigurations` key in the `Info.plist` scene manifest, the sample's window for this scene is automatically configured and its root view controller is loaded from the storyboard.

Restore a scene

When it's time to restore a scene, iOS calls your delegate `scene(_:willConnectTo:options:)`. The sample app restores the scene to its previous state through the use of `NSUserActivity`.

```
func configure(window: UIWindow?, with activity: NSUserActivity) -> Bool {
    var configured = false

    guard activity.activityType == UserActivity.GalleryOpenDetailActivityType else {
    guard let navigationController = window?.rootViewController as? UINavigationCont

    if let photoID = activity.userInfo?[UserActivity.GalleryOpenDetailPhotoAssetKey]
        let photoTitle = activity.userInfo?[UserActivity.GalleryOpenDetailPhotoTitle
        // Restore the view controller with the 'photoID' and 'photoTitle'.
        if let photoDetailViewController = PhotoDetailViewController.loadFromStorybo
            photoDetailViewController.photo = Photo(assetName: photoID, title: photo

            navigationController.pushViewController(photoDetailViewController, anima
            configured = true
        }
    }
    return configured
}
```

Create multiple windows from drag and drop

This sample creates a separate window when the user drags an image from the collection view to the left or right side of the iPad screen. The sample creates a new window by implementing the `UICollectionViewDragDelegate` function `collectionView(_:itemsForBeginning:at:)` and providing a `UIDragItem` with an associated `NSItemProvider`. Then, the sample passes the photo data to the new window scene with a registered `NSUserActivity`.

```
func collectionView(_ collectionView: UICollectionView, itemsForBeginning session: U
    var dragItems = [UIDragItem]()
    let selectedPhoto = photos[indexPath.row]
    if let imageToDrag = UIImage(named: selectedPhoto.assetName) {
        let userActivity = selectedPhoto.detailUserActivity
        let itemProvider = NSItemProvider(object: imageToDrag)
        itemProvider.registerObject(userActivity, visibility: .all)
        let dragItem = UIDragItem(itemProvider: itemProvider)
        dragItem.localObject = selectedPhoto
        dragItems.append(dragItem)
    }
    return dragItems
```

The NSUserActivity activityType must be included in the app’s Info.plist under the NSUserActivityTypes array. Without it, a separate window isn’t created as the photo is dragged to the edge of the device.

```
    <key>NSUserActivityTypes</key>
    <array>
        <string>com.apple.gallery.openDetail</string>
    </array>
```

Create multiple windows programmatically

This sample also creates a separate window scene programmatically, by responding to a user action:

- **iPadOS**: Tap and hold a photo and select the menu item Inspect. This creates a form sheet window with that photo. The form sheet can then be dragged to the left or right side of the iPad screen to split the app’s interface in two.
- **macOS**: Select a photo. Click the Info toolbar button or command-click the photo and select Inspect. Both create a new window containing that photo.

Both approaches use the following code to create a new window scene.

```
class func openInspectorSceneSessionForPhoto(_ photo: Photo, requestingScene: UIWinc
    let options = UIWindowScene.ActivationRequestOptions()
    options.preferredPresentationStyle = .prominent
    options.requestingScene = requestingScene // The scene object that requested the

    // Present this scene as a secondary window separate from the main window.
    //
    // Look for an already open window scene session that matches the photo.
    if let foundSceneSession = InspectorSceneDelegate.activeInspectorSceneSessionFor
        // Inspector scene session already open, so activate it.
        UIApplication.shared.requestSceneSessionActivation(foundSceneSession, // Act
                                                           userActivity: nil, // No
                                                           options: options,
                                                          errorHandler: errorHandle
    } else {
        // No Inspector scene session found, so create a new one.
        let userActivity = photo.inspectorUserActivity

        UIApplication.shared.requestSceneSessionActivation(nil, // Pass nil means ma
                                                          userActivity: userActivit
                                                          options: options,
                                                          errorHandler: errorHandle
    }
}
```

Through the use of a unique NSUserActivity activityType, the app can distinguish which new scene to create in application(_:configurationForConnecting:options:):

```
func application(_ application: UIApplication,
                 configurationForConnecting connectingSceneSession: UISceneSession,
                 options: UIScene.ConnectionOptions) -> UISceneConfiguration {
    // It's important that each UISceneConfiguration have a unique configuration nam
    var configurationName: String!

    switch options.userActivities.first?.activityType {
    case UserActivity.GalleryOpenInspectorActivityType:
        configurationName = "Inspector Configuration" // Create a photo inspector wi
    default:
        configurationName = "Default Configuration" // Create a default gallery winc
    }

    return UISceneConfiguration(name: configurationName, sessionRole: connectingScer
}
```

See Also

Scene Management

Scenes

Manage multiple instances of your app’s UI simultaneously, and direct resources to the appropriate instance of your UI.