Sample Code

Continuing User Activities with Handoff

Define and manage which of your app's activities can be continued between devices.

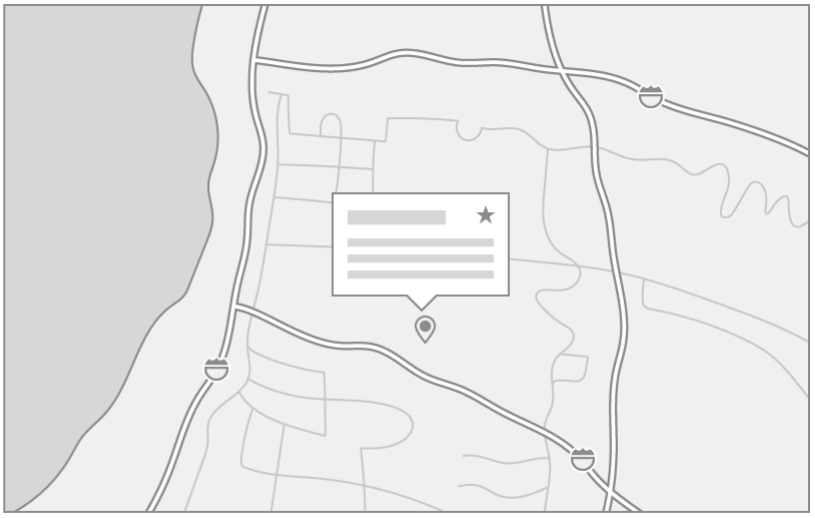
Download

iOS 12.0+ | iPadOS 12.0+ | macOS 10.14+ | Xcode 15.0+

Overview

This sample app searches for Apple Store locations and shows them on a map. The user can choose a store's map annotation to see its address and mark it as a favorite. As the user changes visible regions or inspects individual stores, the app uses Handoff to share these activities with the user's other devices. If the user changes devices, they can use Handoff to launch the app and return to what they were doing on the original device.

Mobile



Desktop

The sample project builds for both macOS and iOS, so you can run it on a Mac, iPhone, and iPad. The project does not contain a watchOS or tvOS app.

Configure the Sample Code Project

`HandoffMapViewer` must be run on actual devices; the iOS version cannot run in Simulator.

To configure your Mac to run the sample code project, open System Preferences and do the following:

1. In Bluetooth settings, click Turn Bluetooth On.
2. In iCloud settings, verify that you are signed into iCloud. If you are not, click Sign In and enter your Apple ID and password.
3. In General settings, select “Allow Handoff between this Mac and your iCloud devices”.

To configure your iOS devices to run the sample code project, open the Settings app and do the following:

1. In Bluetooth settings, tap to turn on Bluetooth.
2. In the user banner at the top of Settings, tap to sign in with your Apple ID if you haven’t already. Then tap to turn on iCloud.
3. In General settings, tap to turn on Handoff.

To configure the sample code project so that it can run on your devices, open the `HandoffMapViewer.xcodeproj` project in Xcode and do the following:

1. Select the `HandoffMapViewer` project at the top of the Project Navigator, select the `HandoffMapViewerMac` target, select the “General” tab, and change the Bundle Identifier to a unique value, such as one that uses your organization’s name instead of `com.example`.
2. With the `HandoffMapViewer` project still selected in the Project Navigator, select the `HandoffMapViewerIOS` target, select the “General” tab, and change the Bundle Identifier to the same value you used in the previous step.
3. To run the macOS version, build the `HandoffMapViewerMac` target, and run it locally, or copy the application file from the `Products` folder to another Mac and run it there.
4. To run the iOS version, build the `HandoffMapViewerIOS` target and run it on one of your connected iOS devices.

Define User Activities

You implement Handoff by determining specific activities that a user can perform in your app, and whose state you can reproduce on a second device. The sample app has two user activities:

- Viewing a map region.
- Viewing the details of a specific Apple Store and editing its “favorite” value.

> **Note**
>
> Be aware that your app usually has other persistence and synchronization strategies in addition to Handoff. For example, the sample app uses an iCloud key-value store to keep track of which stores have been marked as favorites. This way, the favorites data is available to the app if you launch it outside of Handoff, such as from the macOS Dock or the iOS Home screen.

You use the app's `Info.plist` to tell Handoff which activities your app can continue, by providing an entry with the key name `NSUserActivityTypes`. The type of this entry is `Array`, and each member is a `String` representing a supported Handoff activity. In the sample app, the macOS and iOS targets include the `map-viewing` and `store-editing` activities in their `Info.plist` files.

```
<key>NSUserActivityTypes</key>
<array>
  <string>com.example.apple-samplecode.HandoffMapViewer.map-viewing</string>
  <string>com.example.apple-samplecode.HandoffMapViewer.store-editing</string>
</array>
```

Manage User Activities

At runtime, you represent a user activity with the `NSUserActivity` type. You initialize a user activity object with a string identifier, the same one used earlier in the `Info.plist`. This object also has an `isEligibleForHandoff` property that exposes the activity to Handoff, and a `userInfo` dictionary containing data needed to recreate the app's state on the receiving device.

In the sample app, the `MapViewController` manages two `NSUserActivity` instances: one each for the `map-viewing` and `store-editing` activities. When the map region changes, it sets the `userActivity` property (defined in `NSViewController` for macOS and `UIViewController` for iOS) to the `map-viewing` activity. It makes this the current activity, replacing any other activity that may have previously been sent to Handoff, and sets `needsSave` to `true`, indicating that the activity has new data to send to remote devices.

```
userActivity = mapViewingActivity
mapViewingActivity.needsSave = true
mapViewingActivity.becomeCurrent()
```

Calling `needsSave` on the view controller's `userActivity` eventually results in a callback to the method `updateUserActivityState(_:)`, declared in `UIResponder` on iOS and

NSResponder on macOS. This is the app's opportunity to refresh the activity object's userInfo before Handoff receives the activity. The implementation in the sample app calls a convenience function updateViewingRegion(_:), defined in an extension on NSUserActivity, to encode the map view's MKCoordinateRegion into key-value entries in the userInfo dictionary.

```
func updateViewingRegion(_ region: MKCoordinateRegion) {
    let updateDict = [
        NSUserActivity.regionCenterLatitudeKeyString: region.center.latitude,
        NSUserActivity.regionCenterLongitudeKeyString: region.center.longitude,
        NSUserActivity.regionSpanLatitudeKeyString: region.span.latitudeDelta,
        NSUserActivity.regionSpanLongitudeKeyString: region.span.longitudeDelta]
    addUserInfoEntries(from: updateDict)
}
```

Receive User Activities

When you move to another device, macOS or iOS indicates that a Handoff activity is available. macOS displays a Handoff icon at the beginning of the Dock, with a badge indicating the type of source device. On iOS, the Handoff banner appears at the bottom of the screen in the app switcher, showing the app and source device name.

When you launch the app using the Handoff prompts, the system calls methods in UIApplicationDelegate (iOS) or NSApplicationDelegate (macOS) to provide the Handoff activity. The application(_:continue:restorationHandler:) method provides the activity, along with a completion handler that you call with an array of view controllers that can handle the activity. The implementation in the iOS app delegate just finds and passes the first view controller, an instance of MapViewController.

```
func application(_ application: UIApplication, continue userActivity: NSUserActivity
                 restorationHandler: @escaping ([UIUserActivityRestoring]?) -> Void)
    guard let topNav = application.keyWindow?.rootViewController as? UINavigationCon
        let mapVC = topNav.viewControllers.first as? MapViewController else {
        return false
    }

    mapVC.loadView()
    restorationHandler([mapVC])
    return true
}
```

The implementation in the macOS app delegate is similar, except that it traverses the key window's hierarchy, rather than the iOS navigation controller stack:

```
func application(_ application: NSApplication, continue userActivity: NSUserActivity
                 restorationHandler: @escaping ([NSUserActivityRestoring]) -> Void)
    guard let mapVC = application.keyWindow?.windowController?.contentViewControlle
        return false
    }

    mapVC.loadView()
    restorationHandler([mapVC])
    return true
}
```

Update the App's State

The view controllers receive the `NSUserActivity` in the `restoreUserActivityState(_:)` method. `MapViewController` inspects the activity to determine whether it is the map-viewing or the store-editing activity, and then updates the UI as needed. The map-viewing activity case resets the map region, by creating a new `MKCoordinateRegion` from the values in the `userInfo`.

```
func viewingRegion() -> MKCoordinateRegion? {
    guard let centerLatitude = userInfo?[NSUserActivity.regionCenterLatitudeKeyStrin
        let centerLongitude = userInfo?[NSUserActivity.regionCenterLongitudeKeyStrin
        let spanLatitude = userInfo?[NSUserActivity.regionSpanLatitudeKeyString] as?
        let spanLongitude = userInfo?[NSUserActivity.regionSpanLongitudeKeyString] a
            return nil
    }
    return MKCoordinateRegion(center: CLLocationCoordinate2D(latitude: centerLatitud
                                                             longitude: centerLongit
                              span: MKCoordinateSpan(latitudeDelta: spanLatitude,
                                                     longitudeDelta: spanLongitude))
}
```

In the case of the store-editing activity, the view controller also retrieves the store's URL and location coordinates from the `userInfo`. The app waits until the map adds a `MKAnnotationView` for the store being edited, so it knows where to anchor the popover.

Update the Original Device's State (Optional)

The `NSUserActivity` class has a `delegate` property of type `NSUserActivityDelegate`. This notifies the originating device when you continue an activity on another device. The originating device can use this to clean up its own UI state.

In the sample app, tapping a pin for an Apple Store shows a popover with details about the store and a switch (iOS) or checkbox (macOS) to mark the store as a favorite. The `MapViewContoller` represents this activity as the `storeEditingActivity` property, and sets itself as the activity's delegate. When you continue editing on a second device, the delegate on the originating device receives a notification that this activity has been continued, and dismisses its own popover.

```
func userActivityWasContinued(_ userActivity: NSUserActivity) {
    DispatchQueue.main.async {[weak self] in
        if let detailVC = self?.presentedViewController as? StoreDetailViewControlle
           userActivity.activityType == NSUserActivity.storeEditingActivityType {
            detailVC.dismiss(animated: true)
        }
    }
}
```

> **Note**
>
> Not all apps need to update the state of the originating app. In the sample app, dismissing the popover is meant as a gentle reminder to not make simultaneous edits from two devices. This is neither explicitly prohibited nor supported in the sample.

See Also

Activity Sharing

Implementing Handoff in Your App

Create, send, and receive user activities directly.

`class` `NSUserActivity`

A representation of the state of your app at a moment in time.

`protocol` `NSUserActivityDelegate`

The interface through which a user activity instance notifies its delegate of updates.