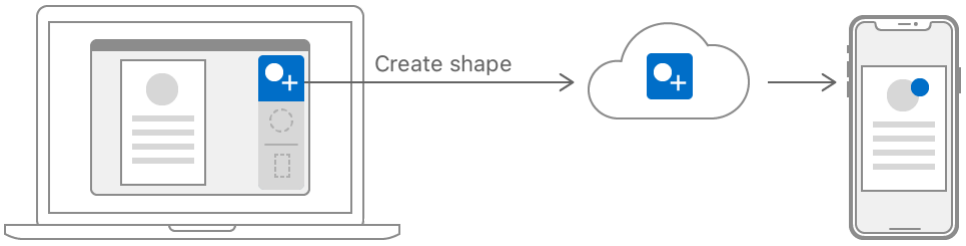
Article

Implementing Handoff in Your App

Create, send, and receive user activities directly.

Overview

Use Handoff to transfer activities the user starts on one iOS, watchOS, or macOS device to a different device. For example, a vector graphics app on macOS can send details about an in-progress editing action to the user's iPhone so that editing can continue there.



You implement Handoff in your app by:

- Representing user activities as instances of `NSUserActivity`.
- Updating the activity instances as the user performs actions in your app.
- Receiving activities from Handoff in your app on other devices.

Important

To handoff between apps on different platforms, your apps must share the same developer Team ID. This means you must either distribute your apps through the App Store, or sign them with the same credentials.

Declare Handoff Activities in Your App's Info.plist

Start by identifying which activities make sense to use with Handoff. Choose activities that represent what the user is doing at some point in time, like creating a shape or editing document properties. Choose a universally-unique identifier string for each of your activities, using a reverse-DNS pattern, like `com.example.app.activity-name`.

You use your app's `Info.plist` file to declare that your app can receive an activity from Handoff. Create a new top-level entry in this file with the key NSUserActivityTypes and with the type `Array`. Each member of the array should be a `String` whose value is one of your activity identifiers. The following example shows the `Info.plist` XML source of a `NSUserActivityTypes` entry that declares three activities that the app can continue:

```
<key>NSUserActivityTypes</key>
<array>
    <string>com.example.myapp.create-shape</string>
    <string>com.example.myapp.edit-shape</string>
    <string>com.example.myapp.edit-document-properties</string>
</array>
```

Your app doesn't need to send and receive the same set of identifiers on all platforms. For example, you might have a large macOS app and a suite of smaller iOS apps. In this case, the macOS app might handle all your activities, while each iOS app would handle a subset of these activities. Also, while watchOS can send user activities, it cannot receive them, so watchOS apps don't declare an `NSUserActivityTypes` property.

Your app can have many activities, each of which has different details to send to Handoff. Identify what information you'll need to recreate the activity on the receiving device. Be careful to only include the transient details of the user activity, and not any information that the app needs to store permanently. For example, if a user is working on a document, the activity should indicate the document—and possibly what part of the document—the user is editing. Don't include the document itself as part of the activity, since the user could launch your app without Handoff, such as by tapping or clicking its app icon. Instead, use techniques like iCloud Drive to share documents between the user's devices.

Create User Activity Objects

At runtime, create instances of `NSUserActivity` for each of your app's activities. Use the same identifier strings that you used in the `Info.plist` to indicate which activities your app can continue.

The `NSUserActivity` class contains a `userInfo` dictionary that you use to recreate the activity on other devices. The activity type also has a `requiredUserInfoKeys` property that you populate with the minimal set of dictionary keys to make the activity restorable. The activity also

contains a user-readable `title` property that you should set. If the activity also supports search, the system displays this title in the search results.

```
let activity = NSUserActivity(activityType: "com.example.myapp.create-shape")
activity?.isEligibleForHandoff = true
activity?.requiredUserInfoKeys = ["shape-type"]
activity.title = NSLocalizedString("Creating shape", comment: "Creating shape activi
```

The `NSResponder` (macOS) and `UIResponder` (iOS) classes define a `userActivity` property. Since `NSViewController` and `UIViewController` are subclasses of these responder types, you can set this property to represent the activity the controller is managing. Your app can share a single activity across multiple view controllers. Conversely, if a single view controller supports multiple activities, you can reassign the view controller's `userActivity` to different `NSUserActivity` instances as needed.

Update Activities While the User is Active

As your user interacts with your app, update the user activity to save the state of their interaction. If you have set the `userActivity` property of a responder, the system automatically calls its `updateUserActivityState(_:)` (iOS) or `updateUserActivityState(_:)` (macOS) method. Override this method to write new values to the activity's `userInfo` dictionary.

The keys and values you use in `userInfo` must be of the types `NSArray`, `NSData`, `NSDate`, `NSDictionary`, `NSNull`, `NSNumber`, `NSSet`, `NSString`, or `NSURL` (or their Swift-bridged equivalents). Create a dictionary with any data needed to recreate the activity on the other device, then call `addUserInfoEntries(from:)` to update the activity. It's also a good idea to provide a key-value pair that versions the dictionary itself. That way, you can change the activity's dictionary representation later and be able to detect incompatible versions.

```
override func updateUserActivityState(_ activity: NSUserActivity) {
    if activity.activityType == "com.example.myapp.create-shape" {
        let updateDict:  [AnyHashable : Any] = [
            "shape-type" : currentShapeType(),
            "activity-version" : 1
        ]
        activity.addUserInfoEntries(from: updateDict)
    }
}
```

Transfer as small a payload in the `userInfo` as possible, keeping the total size under 3KB. If you must transfer more data than this, use continuation streams to connect the two devices directly

(see Working with continuation streams).

Receive User Activities in the Application Delegate

When the user launches your app from Handoff on another device, the app receives callbacks to methods in its application delegate. You implement these methods to accept the activity and restore its state in your app.

After launching your app, Handoff calls the `application(_:willContinueUserActivityWithType:)` method of `UIApplicationDelegate` (iOS), or `application(_:willContinueUserActivityWithType:)` method of `NSApplicationDelegate` (macOS). Implement this method by updating your UI to indicate to your user that it is receiving the activity from the other device. If Handoff fails for some reason, the system calls `application(_:didFailToContinueUserActivityWithType:error:)` (iOS), or `application(_:didFailToContinueUserActivityWithType:error:)` (macOS).

> **Note**
>
> While watchOS can create `NSUserActivity` objects and send them to other devices, Handoff cannot launch watchOS apps.

Handoff provides the activity to your app in the `application(_:continue:restorationHandler:)` (iOS), or `application(_:continue:restorationHandler:)` (macOS) delegate method. Implement the method by creating an array of view controllers that need to update for the activity, and provide this array to the completion handler. Return `true` if your implementation successfully handled the activity, or `false` if it did not. The following example shows an iOS app delegate finding its top view controller and providing it to the completion handler.

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

Continue the Activity in Your App

Each view controller provided to the completion handler in the previous step receives a call to its `restoreUserActivityState(_:)` (iOS), or `restoreUserActivityState(_:)` (macOS) method. Use this method to update the view controller's state to match the state of the originating device. If you have several activity types, use the `activityType` to determine which activity you are handling. Then, get the values from the activity's `userInfo` dictionary to update the view controller's state.

```
override func restoreUserActivityState(_ userActivity: NSUserActivity) {    super.re
    guard userActivity.activityType == "com.example.myapp.create-shape",
        let type = userActivity.userInfo?["shape-type"] as? String,
        let version = userActivity.userInfo?["activity-version"] as? Int,
        version >= 1 else {
            return
    }

    createShape(type: type)
}
```

For URLs transferred in the `userInfo` dictionary, you must first call `startAccessingSecurityScopedResource()` and it must return `true` before you can access the URL. Call `stopAccessingSecurityScopedResource()` when you finish using the URL. Also be aware that the system modifies `file:` URLs pointing to iCloud documents, so that they point to the same document on the receiving device.

See Also

Activity Sharing

{} Continuing User Activities with Handoff

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

`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.