Sample Code

# Increasing App Usage with Suggestions Based on User Activities

Provide a continuous user experience by capturing information from your app and displaying this information as proactive suggestions across the system.

Download

iOS 14.0+ | iPadOS 14.0+ | Xcode 13.0+

## Overview

This sample app shows a list of pizza restaurants, near your current location, in a table view. Select a restaurant from the table view to donate its location information to the system as a user activity object. The system then displays the registered location information proactively as suggestions, such as in Search, used with Siri Shortcuts.

When a user interacts with a proactive suggestion, the system continues the activity by associating the object's activity type with the sample app.

When donating location information, this sample app also verifies whether the previously captured information appears, as QuickType keyboard suggestions, within other apps installed on a device.

## Declare a User Activity Type

You implement proactive suggestions by determining specific activities that a user can perform in your app, and whose state you can recreate at a later time. The sample app has one user activity, `view-location`, which represents the user viewing details of a specific restaurant location.

In the sample app, the target includes the `view-location` activity in its `Information Property List` file, as an entry with the key name . The type of this entry is `Array`, and each member is a `String` representing a supported user activity in reverse DNS notation.

```
<key>NSUserActivityTypes</key>
<array>
<string>com.example.apple-samplecode.ProactiveToolbox.view-location</string>
</array>
```

## Manage a User Activity

At runtime, you represent a user activity with the NSUserActivity object. You initialize a user activity object with a string identifier, the same one declared earlier in the `Info.plist` file.

In the sample app, `LocationViewController` manages a single `NSUserActivity` object, representing the `view-location` activity type. When you select a restaurant, the view controller sets its `userActivity` property to the `view-location` activity—registering the user activity object as the current activity, replacing any other activity previously sent to Siri or the system.

**Important**

When the user activity object is not contained within the responder chain, you must call the `becomeCurrent()` method to mark the object as current, which also registers the object with the system.

The view controller also sets `needsSave` to `true`, indicating that the activity will be updated with new data in the future, which eventually results in a callback to the `updateUserActivityState(_:)` method. This is the app's opportunity to refresh the activity object's `userInfo` property, with the minimal amount of information needed to restore the state of the app, before Siri or the system receives the activity.

```
/*
 Provide just enough information in the `userInfo` dictionary to be able to restore
 The larger the dictionary, the longer it takes to deliver that payload and resume t
 */
var userInfo = [String: Any]()
do {
    let data = try NSKeyedArchiver.archivedData(withRootObject: mapItem.placemark, r
    userInfo["placemark"] = data
} catch {
    os_log("Could not encode placemark data", type: .error)
}

if let phoneNumber = mapItem.phoneNumber {
    userInfo["phoneNumber"] = phoneNumber
}

activity.addUserInfoEntries(from: userInfo)
```

## Provide Support for Improved Search Results

If your user activity objects contain information that the user might want to search for later, you can have Search index those objects and consider them during subsequent on-device searches.

To enable support for Search on a user activity object, set the `isEligibleForSearch` property to `true`. Additionally, you can make an activity object publicly accessible to all iOS users by setting the `isEligibleForPublicIndexing` property to `true`.

```
// The following properties enable the activity to be indexed in Search.
activity.isEligibleForPublicIndexing = true
activity.isEligibleForSearch = true
activity.title = mapItem?.name
activity.keywords = ["pizza"]
```

Provide the activity object with as much rich information about the activity as possible, by configuring the `contentAttributeSet`, `keywords`, or `webpageURL` properties so that the system can index the object. The app must also maintain a strong reference to any activity objects that the system uses for search results.

```
// Provide additional searchable attributes.
activity.contentAttributeSet?.supportsNavigation = true
activity.contentAttributeSet?.supportsPhoneCall = true
activity.contentAttributeSet?.thumbnailData = #imageLiteral(resourceName: "pizza").
```

To learn more about Search, see App Search Programming Guide.

## Provide Support for Shortcuts

Siri learns about the shortcuts available to your app through donations that your app makes to Siri. You can make donations from a user activity when the action involves a view within your app, such as the sample app, which displays a list of nearby restaurants.

The sample app enables support for Siri Shortcuts on a user activity object by setting the `isEligibleForPrediction` property to `true`. The app suggests an invocation phrase, which is displayed to the user when they create a shortcut, by setting a short, memorable phrase to the `suggestedInvocationPhrase` property.

```
// The following properties enable the activity to be used as a shortcut with Siri.
activity.isEligibleForPrediction = true
activity.suggestedInvocationPhrase = "Show my favorite pizzeria"
```

During development, the system provides developer settings to modify the display behavior of shortcuts, and force sync shortcuts to an Apple Watch from an iPhone device. Go to Settings > Developer on the iPhone device to test an app's behavior with Siri Shortcuts.

For more information, see Donating Shortcuts.

## Provide Support for Sharing

If your user-activity object contains information about a webpage, your users can share that webpage with others through Siri. For example, the user can share the pizza restaurant they're viewing by asking Siri to "share this", to send a message that contains the webpage URL to another user. Siri uses the URL stored by the app in the `webpageURL` on the user activity.

```
// Enable sharing this location by telling Siri to "share this".
activity.webpageURL = mapItem?.url
```

## Handle Activities

When a user interacts with a proactive suggestion for your app, such as through Siri Shortcuts, the system restores your app to the foreground, receives data associated with its user activity, and calls `application(_:continue:restorationHandler:)`.

To give `SearchViewController` a chance to perform its state restoration operation through `restoreUserActivityState(_:)`, the sample app verifies the activity type of the user activity object, and passes the first tab's view controller hierarchy to the `restorationHandler` block parameter.

```
func application(_ application: UIApplication,
                 continue userActivity: NSUserActivity,
                 restorationHandler: @escaping ([UIUserActivityRestoring]?) -> Void)

    guard userActivity.activityType == "com.example.apple-samplecode.ProactiveToolbo
        let tabBarController = window?.rootViewController as? UITabBarController,
        let navigationController = tabBarController.viewControllers?.first as? UINav
        else { return false }

    tabBarController.selectedIndex = 0
    /*
     Calling the restoration handler is optional and is only needed
     when specific objects are capable of continuing the activity.
     */
    restorationHandler(navigationController.viewControllers)

    return true
}
```

The sample app validates the user activity, consumes any necessary information from the `userInfo` dictionary, and updates its UI to continue the requested activity from where the user left off.

```
override func restoreUserActivityState(_ activity: NSUserActivity) {
    super.restoreUserActivityState(activity)

    do {
        guard let userInfo = activity.userInfo,
            let url = activity.webpageURL,
            let phoneNumber = userInfo["phoneNumber"] as? String,
            let placemarkData = userInfo["placemark"] as? Data,
            let placemark = try NSKeyedUnarchiver.unarchivedObject(ofClass: MKPlacem
            else { return }

        let mapItem = MKMapItem(placemark: placemark)
        mapItem.name = activity.title

        if navigationController?.visibleViewController == self {
            restoreMapItem(mapItem, url: url, phoneNumber: phoneNumber)
        } else if let modalController = navigationController?.visibleViewController
            modalController.restoreMapItem(mapItem, url: url, phoneNumber: phoneNumb
        }
    } catch {
        os_log("Could not convert user activity placemark data to placemark object",
    }
}
```

## Verify QuickType Keyboard Integration

Text fields describe the intended purpose of input data, like a region's name or postal code, via the `textContentType` property. If any app on the device recently donated information matching the expected text content type, that information is suggested in the QuickType keyboard as the user types in the text field.

For example, in the sample app, a user selects a pizza restaurant, and its address is suggested in Messages or a search field in Apple Maps.

```
// These properties can also be set on the field in Interface Builder.
nameTextField.textContentType = .organizationName
streetAddressLine1TextField.textContentType = .streetAddressLine1
streetAddressLine2TextField.textContentType = .streetAddressLine2
cityTextField.textContentType = .addressCity
stateTextField.textContentType = .addressState
postalCodeTextField.textContentType = .postalCode
countryOrRegionTextField.textContentType = .countryName
```

To verify this behavior, select a point of interest in Apple Maps, then return to this sample app to see the donated information populated in the QuickType keyboard.

## 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.

{} Continuing User Activities with Handoff

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