Sample Code

Interacting with nearby points of interest

Provide automatic search completions for a partial search query, search the map for relevant locations nearby, and retrieve details for selected points of interest.

Download

iOS 18.0+ | iPadOS 18.0+ | visionOS 2.2+ | Xcode 16.2+

Overview

This sample code project demonstrates how to programmatically search for map-based addresses and points of interest using a natural language string, and how to get more information about points of interest that a person selects on the map. The search results center around the locations visible in the map view.

Request search completions

`MKLocalSearchCompleter` retrieves autocomplete suggestions for a partial search query within a map region. A person can type "cof", and a search completion suggests "coffee" as the query string. As the person types a query into a search bar, the sample app updates the query. In SwiftUI, the sample creates the search field using the `searchable(text:placement:prompt:)` modifier.

```
.searchable(text: $searchQuery, placement: .navigationBarDrawer(displayMode: .alway
```

As someone types a query into a search bar, the sample app updates the `queryFragment` for the search completion through the `searchQuery` binding.

```swift
/// Ask for completion suggestions based on the query text.
func provideCompletionSuggestions(for query: String) {
    /**
     Configure the search to return completion results based only on the options in
     someone can configure the app to exclude specific point-of-interest categories,
     */
    searchCompleter?.resultTypes = mapConfiguration.resultType.completionResultType
    searchCompleter?.regionPriority = mapConfiguration.regionPriority.localSearchReg
    if mapConfiguration.resultType == .pointsOfInterest {
        searchCompleter?.pointOfInterestFilter = mapConfiguration.pointOfInterestOpt
    } else if mapConfiguration.resultType == .addresses {
        searchCompleter?.addressFilter = mapConfiguration.addressOptions.filter
    }

    searchCompleter?.region = mapConfiguration.region
    searchCompleter?.queryFragment = query
}
```

Receive completion results

Completion results represent fully formed query strings based on the query fragment someone types. The sample app uses completion results to populate UI elements to quickly fill in a search query. The app receives the latest completion results as an array of `MKLocalSearchCompletion` objects by adopting the `MKLocalSearchCompleterDelegate` protocol.

```swift
nonisolated func completerDidUpdateResults(_ completer: MKLocalSearchCompleter) {
    Task { @MainActor in
        /**
         As a person types, new completion suggestions continuously return to this m
         results, so that the app UI can observe the change and display the updated
         */
        let suggestedCompletions = completer.results
        resultStreamContinuation?.yield(suggestedCompletions)
    }
}
```

The app uses an `AsyncStream` to deliver the completion results to the UI, which the `SidebarView` stores in its `searchCompletions` property. The app displays the search suggestions with the `searchSuggestions(_:)` modifier, which takes a binding to the `searchCompletions` property.

```
.searchSuggestions {
    // Treat each `MKMapItem` object as unique, using `\.self` for the identity. The
    // is an optional value, and the meaning of the identifier for `MKMapItem` doesn
    // the `Identifable` protocol that `ForEach` requires.
    ForEach($searchCompletions, id: \.self) { completion in
        SearchCompletionItemView(completion: completion.wrappedValue)
        .onTapGesture {
            convertSearchCompletionToSearchResults(completion.wrappedValue)
        }
    }
}
```

Highlight the relationship of a query fragment to the suggestion

Within the UI elements that represent each query result, the sample code uses the `titleHighlightRanges` on an `MKLocalSearchCompletion` to show how the query someone enters relates to the suggested result. For example, the following code applies a highlight with `NSAttributedString`:

```
private func createHighlightedString(text: String, rangeValues: [NSValue]) -> NSAttr
    let attributes = [NSAttributedString.Key.backgroundColor: UIColor(named: "sugges
    let highlightedString = NSMutableAttributedString(string: text)

    // Each `NSValue` wraps an `NSRange` that functions as a style attribute's range
    let ranges = rangeValues.map { $0.rangeValue }
    for range in ranges {
        highlightedString.addAttributes(attributes, range: range)
    }

    return highlightedString
}
```

Search for map items

An `MKLocalSearch.Request` takes either an `MKLocalSearchCompletion` or a natural language query string, and returns an array of `MKMapItem` objects. Each `MKMapItem` represents a geographic location, like a specific address, that matches the search query. The sample code asynchronously retrieves the array of `MKMapItem` objects by calling `start(completionHandler:)` on `MKLocalSearch`.

```
let search = MKLocalSearch(request: request)
currentSearch = search
defer {
    // After the search completes, the reference is no longer needed.
    currentSearch = nil
}

var results: [MKMapItem]

do {
    let response = try await search.start()
    results = response.mapItems
} catch let error {
    searchLogging.error("Search error: \(error.localizedDescription)")
    results = []
}
```

Allow someone to select points of interest on the map

If a person is exploring the map, they can get information for a point of interest by tapping it. To provide these interactions, the sample code enables selectable map features as follows:

```
// Use the standard map style, with an option to display specific point-of-interest
.mapStyle(.standard(pointsOfInterest: mapModel.searchConfiguration.pointOfInterestOp

// Only allow selection for points of interest, and disable selection of other label
.mapFeatureSelectionDisabled { feature in
    feature.kind != MapFeature.FeatureKind.pointOfInterest
}

/*
 The selection accessory allows people to tap on map features and get more detailed
 as either a sheet or a callout according to the `style` parameter. Along with the `
 which feature to display additional information for.

 This modifier differs from the `mapItemDetailSelectionAccessory(:_) modifier, which
 behaviors on annotations that the app adds to `Map` for search results.
 */
.mapFeatureSelectionAccessory(.automatic)
```

When someone taps a point of interest, the system presents the map item's details, including information like a phone number, business hours, and buttons to start navigation to the location using Apple Maps. The system presents the information using the style that the `mapFeatureSelectionAccessory(_:)` modifier configures. The sample app uses the `automatic` style, but the `MapItemDetailSelectionAccessoryStyle` structure offers several other options.

Persist and retrieve map items

If someone is exploring the map, they may want the app to store places they looked at so that they can come back to them later, including across app launches. `MKMapItem` has an `identifier` property, which the app stores in its `VisitedPlace` model using `SwiftData`.

```
guard let identifier = mapItem.identifier else { return }
let visit = VisitedPlace(id: identifier.rawValue)
```

When the app launches, it retrieves the history of visited locations from SwiftData. To get the `MKMapItem` from the previously stored identifier, the app creates an `MKMapItemRequest` with the stored identifier and calls `getMapItem(completionHandler:)`.

```
@MainActor
func convertToMapItem() async -> MKMapItem? {
    guard let identifier = MKMapItem.Identifier(rawValue: id) else { return nil }
    let request = MKMapItemRequest(mapItemIdentifier: identifier)
    var mapItem: MKMapItem? = nil
    do {
        mapItem = try await request.mapItem
    } catch let error {
        let logger = Logger(subsystem: Bundle.main.bundleIdentifier!, category: "Map
        logger.error("Getting map item from identifier failed. Error: \(error.locali
    }
    return mapItem
}
```

See Also

Local search

`enum MKLocalSearchRegionPriority`

A value that indicates the importance of the configured region.

`struct ResultType`

Options that indicate types of search results.

`class MKLocalSearch`

A utility object for initiating map-based searches and processing the results.

`struct Options`

A structure that contains options for filtering results in a search.

`class MKAddressFilter`

An object that filters which address options to include or exclude in search results.

`struct ResultType`

Options that indicate types of search completions.

`class MKLocalSearchCompleter`

A utility object for generating a list of completion strings based on a partial search string that you provide.

`class MKLocalSearchCompletion`

A fully formed string that completes a partial string.

`class MKLocalPointsOfInterestRequest`

A structured request to use when searching for points of interest.