Sample Code

Adopting App Intents to support system experiences

Create app intents and entities to incorporate system experiences such as Spotlight, visual intelligence, and Shortcuts.

Download

iOS 26.0+ | iPadOS 26.0+ | macOS 26.0+ | Xcode 26.0+

Overview

The app in this sample offers actions in the Shortcuts app that people can use to create custom shortcuts. It includes an App Shortcut to find the closest landmark and find tickets to visit the landmark, all without opening the app. Additionally, the app makes its data available to system experiences like Spotlight, Siri and Apple Intelligence, and visual intelligence.

By adopting the App Intents framework, the app provides functionality across the system, enabling people to:

- In Shortcuts, find and run the app's app intents.
- In Shortcuts, create custom shortcuts or view the provided "Find Closest" App Shortcut.
- In Shortcuts, place custom shortcuts or the App Shortcut on the Home Screen as a bookmark.
- In Spotlight, search for a landmark or the "Find Closest" App Shortcut.
- With visual intelligence, circle an object in the visual intelligence camera or onscreen and view matching results from the app.
- With the Action button, trigger a custom shortcut or the App Shortcut.
- From Siri suggestions, use custom shortcuts or the App Shortcut.

- In the app, view information about a landmark, then ask Siri something like “What’s a summary of the history of this place?” or similar to receive a content summary, and more.

Describe actions as app intents and entities

The app contains many actions and makes them available to the system as app intents, so people can use them to create custom shortcuts and invoke across system experiences. For example, the app offers key actions like finding the closest landmark or opening a landmark in the app. This app intent opens a landmark in the app:

```swift
import AppIntents

struct OpenLandmarkIntent: OpenIntent {
    static let title: LocalizedStringResource = "Open Landmark"

    @Parameter(title: "Landmark", requestValueDialog: "Which landmark?")
    var target: LandmarkEntity

    func perform() async throws -> some IntentResult {
        return .result()
    }
}
```

To use your data as input and output of app intents and make the data available to the system, you use app entities. App entities often limit the information a model object you persist to storage to what the system needs. They also add required information to understand the data or to use it in system experiences. For example, the `LandmarkEntity` of the sample app provides required `typeDisplayRepresentation` and `displayRepresentation` properties but doesn’t include every property of the `Landmark` model object:

```swift
struct LandmarkEntity: IndexedEntity {
    static var typeDisplayRepresentation: TypeDisplayRepresentation {
        return TypeDisplayRepresentation(
            name: LocalizedStringResource("Landmark", table: "AppIntents", comment:
            numericFormat: "\(placeholder: .int) landmarks"
        )
    }

    var displayRepresentation: DisplayRepresentation {
        DisplayRepresentation(
            title: "\(name)",
            subtitle: "\(continent)",
            image: .init(data: try! self.thumbnailRepresentationData)
        )
    }

    static let defaultQuery = LandmarkEntityQuery()

    var id: Int { landmark.id }

    @ComputedProperty(indexingKey: \.displayName)
    var name: String { landmark.name }

    // Maps the description variable to the Spotlight indexing key `contentDescripti
    @ComputedProperty(indexingKey: \.contentDescription)
    var description: String { landmark.description }

    @ComputedProperty
    var continent: String { landmark.continent }

    @DeferredProperty
    var crowdStatus: Int {
        get async throws { // swiftlint:disable:this implicit_getter
            await modelData.getCrowdStatus(self)
        }
    }

    var landmark: Landmark
    var modelData: ModelData

    init(landmark: Landmark, modelData: ModelData) {
        self.modelData = modelData
        self.landmark = landmark
    }
}
```

For more information about describing actions as app intents and app entities, refer to Making actions and content discoverable and widely available and Creating your first app intent.

Offer interactive snippets

The app's "Find Closest" App Shortcut performs an app intent that finds the closest nearby landmark without opening the app, and allows people to find tickets to visit it. Instead of taking them to the app, the app intent displays interactive snippets that appear as overlays at the top of the screen. To display the interactive snippet, the app's `ClosestLandmarkIntent` returns a `SnippetIntent` that presents the interactive snippet in its `perform()` method:

```
import AppIntents
import SwiftUI

struct ClosestLandmarkIntent: AppIntent {
    static let title: LocalizedStringResource = "Find Closest Landmark"

    @Dependency var modelData: ModelData

    func perform() async throws -> some ReturnsValue<LandmarkEntity> & ShowsSnippetI
        let landmark = await self.findClosestLandmark()

        return .result(
            value: landmark,
            dialog: IntentDialog(
                full: "The closest landmark is \(landmark.name).",
                supporting: "\(landmark.name) is located in \(landmark.continent)."
            ),
            snippetIntent: LandmarkSnippetIntent(landmark: landmark)
        )
    }
}
```

For more information about displaying interactive snippets, refer to Displaying static and interactive snippets.

Make your entity available to Siri and Apple Intelligence

To allow Siri to access the landmark information that's visible onscreen in the app, its `LandmarkEntity` implements the `Transferable` protocol and provides plain-text, image, and PDF representations that Siri can understand and forward to other services, including third-party services:

```
extension LandmarkEntity: Transferable {
    static var transferRepresentation: some TransferRepresentation {
        FileRepresentation(exportedContentType: .pdf) { @MainActor landmark in
            let url = URL.documentsDirectory.appending(path: "\(landmark.name).pdf")
            let renderer = ImageRenderer(content: VStack {
                Image(landmark.landmark.backgroundImageName)
                    .resizable()
                    .aspectRatio(contentMode: .fit)
                Text(landmark.name)
                Text("Continent: \(landmark.continent)")
                Text(landmark.description)
            }.frame(width: 600))

            renderer.render { size, renderer in
                var box = CGRect(x: 0, y: 0, width: size.width, height: size.height

                guard let pdf = CGContext(url as CFURL, mediaBox: &box, nil) else {
                    return
                }
                pdf.beginPDFPage(nil)
                renderer(pdf)
                pdf.endPDFPage()
                pdf.closePDF()
            }

            return .init(url)
        }

        DataRepresentation(exportedContentType: .image) {
            try $0.imageRepresentationData
        }

        DataRepresentation(exportedContentType: .plainText) {
            """
            Landmark: \($0.name)
            Description: \($0.description)
            """.data(using: .utf8)!
        }
    }
}
```

When the landmark becomes visible onscreen, the app uses the user activity annotation API to give the system access to the data:

```
HStack(alignment: .bottom) {
    Text(landmark.name)
        .font(.title)
        .fontWeight(.bold)
        .userActivity(
            "com.landmarks.ViewingLandmark"
        ) {
            $0.title = "Viewing \(landmark.name)"
            $0.appEntityIdentifier = EntityIdentifier(for: try! modelData.landmarkEr
        }
}
```

For more information about making onscreen content available to Siri and Apple Intelligence, refer to Making onscreen content available to Siri and Apple Intelligence.

Add entities to the Spotlight index

The app describes its data as app entities, so the system can use it when it performs app intents. Additionally, the app donates the entities into the semantic search index, making it possible to find the app entities in Spotlight. The following example shows how the app's `LandmarkEntity` conforms to `IndexedEntity` and uses Swift macros to add the indexing keys that Spotlight needs.

```
struct LandmarkEntity: IndexedEntity {
    // ...

    // Maps the description to the Spotlight indexing key `contentDescription`.
    @ComputedProperty(indexingKey: \.contentDescription)
    var description: String { landmark.description }

    @ComputedProperty
    var continent: String { landmark.continent }

    // ...
}
```

In a utility function, the app donates the landmark entities to the Spotlight index:

```
static func donateLandmarks(modelData: ModelData) async throws {
    let landmarkEntities = await modelData.landmarkEntities
    try await CSSearchableIndex.default().indexAppEntities(landmarkEntities)
}
```

For more information, refer to Making app entities available in Spotlight.

Integrate search results with visual intelligence

With visual intelligence, people circle items onscreen or in visual intelligence camera to search for matching results across apps that support visual intelligence. To support visual intelligence search, the sample app implements an `IntentValueQuery` to find matching landmarks:

```
@UnionValue
enum VisualSearchResult {
    case landmark(LandmarkEntity)
    case collection(CollectionEntity)
}

struct LandmarkIntentValueQuery: IntentValueQuery {

    @Dependency var modelData: ModelData

    func values(for input: SemanticContentDescriptor) async throws -> [VisualSearchF

        guard let pixelBuffer: CVReadOnlyPixelBuffer = input.pixelBuffer else {
            return []
        }

        let landmarks = try await modelData.search(matching: pixelBuffer)

        return landmarks
    }
}

extension ModelData {
    /**
     This method contains the search functionality that takes the pixel buffer that
     and uses it to find matching app entities. To keep this example app easy to unc
     returns the same landmark entity.
    */
    func search(matching pixels: CVReadOnlyPixelBuffer) throws -> [VisualSearchResul
        let landmarks = landmarkEntities.filter {
            $0.id != 1005
        }.map {
            VisualSearchResult.landmark($0)
        }.shuffled()
        let collections = userCollections
            .filter {
                $0.landmarks.contains(where: { $0.id == 1005 })
            }
            .map {
                CollectionEntity(collection: $0, modelData: self)
            }
            .map {
                VisualSearchResult.collection($0)
            }

        return [try! .landmark(landmarkEntity(id: 1005))]
            + collections
            + landmarks
    }
}
```

For more information about integrating your app with visual intelligence, refer to Visual Intelligence.

See Also

Essentials

App Intents updates

Learn about important changes in App Intents.

Making actions and content discoverable and widely available

Adopt App Intents to make your app discoverable with Spotlight, controls, widgets, and the Action button.

Creating your first app intent

Create your first app intent that makes your app available in system experiences like Spotlight or the Shortcuts app.

Accelerating app interactions with App Intents

Enable people to use your app's features quickly through Siri, Spotlight, and Shortcuts.