Article

Making app entities available in Spotlight

Allow people to find your app's content in Spotlight by donating app entities to its semantic index.

Overview

With Spotlight, people access information from within apps and the web, and use it to launch apps and perform actions. When you adopt the App Intents framework and create App Shortcuts, people can perform your app's actions from Spotlight. Additionally, you can donate your `AppEntity` objects to Spotlight, allowing people to find your content and launch your app to view more information. For example, people might search nearby landmarks in Spotlight, find a travel app's content, and tap it to launch the app for detailed information.

Create an intent that displays your entity in your app

When an app entity appears in Spotlight, people can tap it to open your app to the scene that shows detailed information matching the entity. For example, the Adopting App Intents to support system experiences sample app makes landmarks available to Spotlight. When someone taps a landmark in Spotlight, the app opens to display detailed information about that landmark.

For each entity type that you want to make available to Spotlight, create an app intent that conforms to `OpenIntent` and takes the entity as a parameter, as in the following example:

```
import AppIntents

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

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

```
}
```

Make your app entity indexable

To donate your app entities to the semantic index of Spotlight, they need to conform to the `IndexedEntity` protocol. For example, the sample app for Adopting App Intents to support system experiences extends its `LandmarkEntity` to add the protocol conformance.

```
struct LandmarkEntity: IndexedEntity {

    // ...

}
```

By stating that your entity conforms to `IndexedEntity`, you can then donate it to Spotlight. By default, the title, subtitle, and image of the entity's `DisplayRepresentation` become indexed attributes in Spotlight, and Spotlight shows your entity if a search matches one of those attributes.

However, your entity likely contains additional information that you want to be searchable and findable in Spotlight. To declare additional attributes that are searchable in Spotlight, implement a custom attribute set for your entity:

- Use the `@Property` or `@ComputedProperty` Swift property wrappers to add your entity's variables to the Spotlight index.
- In iOS 18, provide an `attributeSet` for your indexed entities that contains attributes for Spotlight indexing.
- If you already use Core Spotlight, associate your app entities with a matching `CSSearchableItem`.

For more information about each option, refer to the applicable section below.

Mark your entity's variables as indexable properties

The system can automatically extract the keys for Spotlight indexing at compile time and store them in the App Intents metadata that Xcode generates as part of your app's bundle. As a result, Spotlight indexing is faster and can find your app entities without launching your app, and without you having to explicitly donate the entities to Spotlight. You also don't need to manually update or remove entities from the Spotlight index when your app's data changes.

To add your indexed entity's variables to the Spotlight index, use the `@Property` or `@ComputedProperty` Swift macro as follows:

- Map the indexed entity's variable to an indexing key that `CSSearchableItemAttributeSet` defines, and use `@ComputedProperty(indexingKey:` or `@Property(indexingKey:)`.
- If an indexed entity's variable doesn't fit any of the indexing keys that `CSSearchableItemAttributeSet` defines, use `@ComputedProperty(customIndexingKey:)` or `@Property(customIndexingKey:)` to match the variable to a custom Spotlight indexing key.

> **Note**
>
> If your app entities conform to a schema and domain in App intent domains, donate them to the system without defining an indexing key mapping. Entities that conform to a schema use a fixed shape that maps to Spotlight indexing keys. You can't provide your own mapping for its variables.

The following example shows how the `LandmarkEntity` of the Adopting App Intents to support system experiences sample app uses the different `@ComputedProperty` Swift macros to the entity's variables as searchable attributes to the Spotlight index:

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

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

    // Maps the continent variable to a custom Spotlight indexing key.
    @ComputedProperty(
        customIndexingKey: CSCustomAttributeKey(
            keyName: "com_AppIntentsTravelTracking_LandmarkEntity_continent"
        )!
    )
    var continent: String { landmark.continent }

    // ...
}
```

On app launch, donate your app entities to the Spotlight index using `CSSearchableIndex` and its `indexAppEntities(_:priority:)`.

The following example shows the implementation of the donation in the `AppIntentsTravelTracking` app:

```
import AppIntents
import CoreSpotlight

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

If your app's data changes, delete app entities that no longer exist from the Spotlight index using `deleteAppEntities(identifiedBy:ofType:)` or `deleteAppEntities(ofType:)`.

Implement the optional searchable attribute set for your indexed entity

In iOS 18, you can't donate your app entities and their attributes to the Spotlight index using the `@ComputedProperty` or `@Property` Swift macros. Instead, your app entity needs to provide the `attributeSet` property. You need to manually donate each entity to the index, and then update the index when your data changes.

The following example shows how the sample app for Accelerating app interactions with App Intents creates the `CSSearchableItemAttributeSet`:

```
var searchableAttributes: CSSearchableItemAttributeSet {
        let attributes = CSSearchableItemAttributeSet()

        attributes.title = name
        attributes.namedLocation = regionDescription
        attributes.keywords = activities.localizedElements

        attributes.latitude = NSNumber(value: coordinate.latitude)
        attributes.longitude = NSNumber(value: coordinate.longitude)
        attributes.supportsNavigation = true

        return attributes
    }
```

On launch of your app, add your entities to the Spotlight index using `CSSearchableIndex` and its `indexAppEntities(_:priority:)`. If your app's data changes, delete app entities that no

longer exist from the Spotlight index using `deleteAppEntities(identifiedBy:ofType:)` or `deleteAppEntities(ofType:)`.

Associate existing Core Spotlight indexable items with app entities

If your app already supports Spotlight or uses Core Spotlight for its search functionality, follow these steps to create app entities and associate them with the `CSSearchableItem` that represents the entity in the Spotlight index:

1. Create the `CSSearchabileItem` that represents the entity in the Spotlight index.
2. Set the searchable attributes for `CSSearchableItem`.
3. Associate the searchable item with the app entity it represents with `associateAppEntity(_:priority:)`. Alternatively, initialize `CSSearchabileItem` using one of its initializers that takes an `IndexedEntity` as a parameter. Make sure to associate the app entity with the searchable item before adding the item to the Spotlight index.
4. Add all items to the index using `indexSearchableItems(_:completionHandler:)`.

The following code example from Accelerating app interactions with App Intents shows how the sample app's `TrailDataManager` creates searchable items, associates each item with the matching app entity, and then donates them to the Spotlight index:

```
func updateSpotlightIndex() async {
    guard CSSearchableIndex.isIndexingAvailable() else {
        Logger.spotlightLogging.info("[Spotlight] Indexing is unavailable")
        return
    }

    // Create an array of the searchable information for each `Trail`.
    let searchableItems = trails.map { trail in
        let item = CSSearchableItem(uniqueIdentifier: String(trail.id),
                                    domainIdentifier: nil,
                                    attributeSet: trail.searchableAttributes)

        let isFavorite = favoritesCollection.members.contains(trail.id)
        let weight = isFavorite ? 10 : 1
        let intent = TrailEntity(trail: trail)

        /**
         Associate `TrailEntity` with the data that the `Trail` structure provides s
         both types represent the same data. You need to create this association bet
```

```
            to `CSSearchableIndex`.
            */
            item.associateAppEntity(intent, priority: weight)
            return item
        }

        do {
            // Add the trails to the search index so people can find them through Spotli
            // You need to do this as part of the app's initial setup on launch.
            let index = CSSearchableIndex.default()
            try await index.indexSearchableItems(searchableItems)
            Logger.spotlightLogging.info("[Spotlight] Trails indexed by Spotlight")
        } catch let error {
            Logger.spotlightLogging.error("[Spotlight] Trails were not indexed by Spotli
        }
    }
```

See Also

Other system experiences

- Focus

  Adjust your app's behavior and filter incoming notifications when the current Focus changes.

- Action button on iPhone and Apple Watch

  Enable people to run your App Shortcuts with the Action button on iPhone or to start your app's workout or dive sessions using the Action button on Apple Watch.

- Developing a WidgetKit strategy

  Explore features, tasks, related frameworks, and constraints as you make a plan to implement widgets, controls, watch complications, and Live Activities.