Article

Integrating actions with Siri and Apple Intelligence

Create app intents, entities, and enumerations that conform to assistant schemas to tap into the enhanced action capabilities of Siri and Apple Intelligence.

iOS 18.0+ | iPadOS 18.0+ | macOS 15.0+ | tvOS 18.0+ | visionOS 2.0+ | watchOS 11.0+ | Xcode 16.0+

Overview

Apple Intelligence is a new personal intelligence system that deeply integrates powerful generative models into the core of iPhone, iPad and Mac. Siri will draw on the capabilities of Apple Intelligence to deliver assistance that's more natural, contextually relevant and personal to users.

> **Note**
>
> Siri's personal context understanding, onscreen awareness, and in-app actions are in development and will be available with a future software update.

A big part of people's personal context are the apps they use every day. The App Intents framework gives you a means to express your app's capabilities and content to the system and integrate them with Siri and Apple Intelligence. This will unlock new ways for your users to interact with your app from anywhere on their device.

> **Note**
>
> If you're new to the App Intents framework, make sure to read Making actions and content discoverable and widely available and Creating your first app intent.

Understand assistant schemas

To integrate your app with Siri and Apple Intelligence, provide `AppIntent`, `AppEntity`, and `AppEnum` implementations that work well with the pre-trained models Apple Intelligence uses. To provide the implementation that Apple Intelligence needs, create the necessary code using Swift macros to generate additional properties, and add the relevant protocol conformance for your app intent, app entity, and app enum implementation. Xcode offers templates for the macros and verifies the conformance of your code at compile time.

To create implementations that work well with Siri and Apple Intelligence:

- For your App Intents implementation, use the `AppIntent(schema:)` macro.
- For your `AppEntity` implementation, use the `AppEntity(schema:)` macro.
- For your `AppEnum` implementation, use the `AppEnum(schema:)` macro.

Each macro requires you to provide a `schema` value to generate app intent, app entity, or app enum code that Apple Intelligence can understand. The value you provide to the macros, the *assistant schema*, has two parts:

- The *App intent domain* that describes a collection of APIs for specific functionality; for example, the `.photos` domain if an app has photos or video functionality.
- The *schema*, an action or a content type within the domain, the specific API for the app intent, app entity, or app enum you create.

For example, an app intent that opens a photo from a photo library uses `@AppIntent(schema: .photos.openAsset)` to make sure the intent provides necessary metadata that allows Apple Intelligence to understand it well.

> **Important**
>
> Only use the provided app intents domains and schemas for app actions and content that match the specific domain and schema.

For a list of available assistant schemas, see App intent domains.

Review schema requirements

Conforming to an assistant schema comes with requirements that vary depending on the domain and schema. However, all assistant schemas share the following constraints:

- App intents can't require parameters in addition to the parameters a schema expects. If your app intent uses additional parameters, make them optional.

- Optional parameters that extend the schema are only available to the app intent when it appears in the Shortcuts app.
- App entities can't use required properties in addition to the properties the schema expects. However, you can use optional properties.
- App enums don't come with requirements for their enumeration cases and offer full flexibility, but you can't use more than a total of 10 app enums that conform to assistant schemas.

Create an app intent that implements a schema

To create an app intent that integrates your app functionality with Apple Intelligence:

1. Identify an action or existing app intent implementation in your app that matches a domain listed in App intent domains.
2. Create a new Swift file for your app intent code.
3. Use Xcode code completion to generate code that conforms to an assistant schema: Type the name of the domain, followed by an underscore (`<domain>_`) and choose the schema that fits your action. For example, type `photos_` to see a list of available schemas for the `.photos` domain and choose the action to open an asset (`photos_openAsset`). To ensure assistant schema conformance for an existing app intent, add the macro — for example, `@AppIntent(schema: .photos.openAsset)` — to your existing app intent implementation.
4. Build your app to check for errors that indicate that your app intent implementation doesn't match the chosen assistant schema.
5. Make changes to meet the schema requirements and rebuild your app.

> **Tip**
>
> Xcode code completion can create `AppIntent`, `AppEntity`, and `AppEnum` code that conforms to assistant schemas.

The following code snippet shows how the Making your app's functionality available to Siri sample declares an app intent that opens a video from a device's media library:

```
@AppIntent(schema: .photos.openAsset)
struct OpenAssetIntent: OpenIntent {
    var target: AssetEntity

    @Dependency
    var library: MediaLibrary
```

```
    @Dependency
    var navigation: NavigationManager

    @MainActor
    func perform() async throws -> some IntentResult {
        let assets = library.assets(for: [target.id])
        guard let asset = assets.first else {
            throw IntentError.noEntity
        }

        navigation.openAsset(asset)
        return .result()
    }
}
```

> **Note**
>
> When an app intent conforms to an assistant schema that is known at compile time, the system no longer needs metadata you previously provided, like a `title` or `description`. Remove them to simplify your code. However, you can always supply additional metadata as needed.

Ensure app entities and app enums conform to the schema

Many actions you describe as an `AppIntent` require parameters or return results, often with custom types you describe as an `AppEntity` or `AppEnum`. If you use an app entity or app enum in an intent that conforms to an assistant schema, the entity or enum also needs to conform to the assistant schema. Ensuring conformance works similar to conformance for your app intent:

1. Annotate your app entity or app enum with the `AppEntity(schema:)` or `AppEnum(schema:)` macro or create a new entity or enum using Xcode code completion to automatically generate code that conforms to a schema. For more information, see the previous section.
2. Pass the corresponding domain and entity or enum parts to the macro.
3. Update your code to meet the requirements of the schema.

For example, the `AssetEntity` implementation from the Making your app's functionality available to Siri sample looks like this:

```
@AppEntity(schema: .photos.asset)
struct AssetEntity: IndexedEntity {

    static let defaultQuery = AssetQuery()

    let id: String
    let asset: Asset

    @Property(title: "Title")
    var title: String?

    var creationDate: Date?
    var location: CLPlacemark?
    var assetType: AssetType?
    var isFavorite: Bool
    var isHidden: Bool
    var hasSuggestedEdits: Bool

    var displayRepresentation: DisplayRepresentation {
        DisplayRepresentation(
            title: title.map { "\($0)" } ?? "Unknown",
            subtitle: assetType?.localizedStringResource ?? "Photo"
        )
    }
}
```

Consider the impact of updating existing app intents

Your existing app intents might overlap with functionality that assistant schemas provide. If you can make an existing app intent conform to a schema without making changes to parameters that the intent uses, proceed with adding schema conformance. However, changing existing app intent implementations or removing app intents can directly impact people because their custom shortcuts may no longer work.

To not break people's existing workflows, create a new app intent in addition to an existing app intent. As a result, both intents appear in the Shortcuts app as actions. To avoid them appearing as duplicates, mark your new app intent as available to Apple Intelligence only by setting `isAssistantOnly` to `true`. For example, an app intent implementation could look like this:

```
@AppIntent(schema: .photos.createAssets)
struct CreateAssetsIntent: AppIntent {
```

```
    // ...

    static let isAssistantOnly: Bool = true

    // ...

    @MainActor
    func perform() async throws -> some ReturnsValue<[AssetEntity]> {
        // ...
    }
}
```

Similarly, you can set `isAssistantOnly` to `true` for any applicable app entities and app enums that conform to an assistant schema.

After some time, you can remove the `isAssistantOnly` code and remove your old app intent. For more information about giving people time to update their custom shortcuts with new app intents, see Understand the impact of removing app intents and shortcuts.

See Also

Siri and Apple Intelligence

- Making onscreen content available to Siri and Apple Intelligence

  Enable Siri and Apple Intelligence to respond to a person's questions and action requests for your app's onscreen content.

- App intent domains

  Make your app's actions and content available to Siri and Apple Intelligence with assistant schemas.

- Making your app's functionality available to Siri

  Add app intent schemas to your app so Siri can complete requests, and integrate your app with Apple Intelligence, Spotlight, and other system experiences.