Article

Adding parameters to an app intent

Enable people to configure app intents with their custom input values.

Overview

Many of your app's actions likely require input data to perform their work. To help people provide the input that an `AppIntent` needs to perform its functionality, add parameters to the intent to tell the system about that data and whether it's required or optional. When you expose these parameters, people can configure your intents with values unique to their requirements and enable the App Intents framework to mediate with system experiences to write those values at runtime.

For example, the Accelerating app interactions with App Intents sample code project's `GetTrailInfo` intent lets people choose which hiking trail information to view when they invoke the app intent. It declares a `trail` parameter by decorating the `trail` property with the `IntentParameter` property wrapper and provides a title and a description to identify the parameter in the Shortcuts app.

```
@Parameter(title: "Trail", description: "The trail to get information on.")
var trail: TrailEntity
```

Note that the example doesn't provide localized text for the `title` and `description` fields to keep the example focused and make it easy to understand. Always provide localized strings for app intents, App Shortcuts, and their parameters.

Make a parameter optional or required

How you define your parameter variables determines whether the system treats that parameter as required or optional. If you define a variable as a non-optional type, the system knows the parameter is required and, when necessary, requests a value. Conversely, if you define a variable as an optional type, the system assumes the parameter is optional and doesn't request a value. In

this scenario, use the property wrapper's `requestValue(_:)` method to pause execution and request a value if the intent can't proceed otherwise.

```
guard let date = date else {
    throw $date.requestValue("What date would you like to use?")
}
```

Review supported parameter types

You can use the `@Parameter` property wrapper with common Swift and Foundation types:

- Primitives such as `Bool`, `Int`, `Double`, `String`, `Duration`, `Date`, `Decimal`, `Measurement`, and `URL`.
- Collections such as `Array` and `Set`. Make sure the collection's elements are of a type that's compatible with `IntentParameter`.

Additionally:

- Use framework-specific types such as `IntentPerson` and `IntentFile`. For additional types, see Common types.
- Use enumerable types that conform to the `AppEnum` protocol for parameters that have known static values at build time.
- Use custom types that adopt the `AppEntity` protocol and that the system can request at runtime.

For example, the Accelerating app interactions with App Intents sample code project makes its trail data available in an app intent through the `TrailEntity` type, which is a structure conforming to the `AppEntity` protocol.

Transform input into your intent parameter's types

When a person provides input that your app intents use, the input doesn't always match the type that your parameters require. For example, natural spoken language commands from Siri are strings, but your app intent might require an integer or floating-point value. To help you with input of various types, use Resolvers to leverage the system's ability to translate one type to another automatically so your app intent can use the input.

Restrict parameter values

To make it easy for people to provide your app intents with the right information, restrict parameter values. The system presents known values as a list and prompts the person to select one when it needs to resolve a parameter. To restrict parameter values to a list of known values:

- At compile time, use an enumeration type for the parameter that conforms to the `AppEnum` protocol.
- At runtime, specify an options provider as part of the property wrapper's declaration. An *options provider* is a type you implement that conforms to the `DynamicOptionsProvider` protocol and provides a set of permitted values at runtime.

For example, the Accelerating app interactions with App Intents sample code project uses a dynamic options provider to display a sorted list of location parameters in the Shortcuts app.

```
struct LocationOptionsProvider: DynamicOptionsProvider {

    @Dependency
    private var trailManager: TrailDataManager

    func results() async throws -> [String] {
        Logger.intentLogging.debug("Getting locations from LocationOptionsProvider")

        // Get a list of locations and return it sorted for display, such as in the
        return trailManager.uniqueLocations
                .sorted(using: KeyPathComparator(\.self, comparator: .localizedStand
    }
}
```

You can configure a parameter with additional options such as enforcing an inclusive range for number types, or specifying the capitalization style and keyboard mode for string types. For more information, see `IntentParameter`.

Provide an interactive parameter summary for your intent

A parameter summary is a visual, textual outline of your app intent that the Shortcuts app displays in the shortcut editor. The summary can include placeholders that people interact with to choose the values for the intent's parameters. Even if your intent doesn't expose any parameters, providing a summary is an opportunity to present more information about your intent in addition to its title.

To add a parameter summary to your intent, implement the protocol's `parameterSummary` requirement and use the provided `ParameterSummaryBuilder` result builder to build the

summary. Write the content using localized natural language and, where applicable, substitute words that represent parameters with the key paths to those parameters.

```
static var parameterSummary: some ParameterSummary {
        Summary("Get information on \(\.$trail)")
    }
```

The shortcut editor substitutes each key path with the corresponding parameter's title and enables a person to set the value by tapping it. The editor uses the parameter's type to determine which input controls to display.

Parameter summaries can include conditional statements such as `AppIntent.When` and `AppIntent.Switch` that let the summary update itself in response to already chosen values.

For example, the Accelerating app interactions with App Intents sample code project uses `AppIntent.Switch` in its `SuggestedTrails` app intent:

```
    static var parameterSummary: some ParameterSummary {
        Switch(\.$activity) {
            Case(.biking) {
                When(\.$location, .hasAnyValue) {
                    Summary("Ride a bike within \(\.$searchRadius) of \(\.$location)
                } otherwise: {
                    When(\.$trailCollection, .hasAnyValue) {
                        Summary("Pick a bike ride from \(\.$trailCollection)")
                    } otherwise: {
                        Summary("Suggest bike rides from \(\.$trailCollection) or ne
                    }
                }
            }
            DefaultCase() {
                When(\.$location, .hasAnyValue) {
                    Summary("Suggest \(\.$activity) trails within \(\.$searchRadius)
                } otherwise: {
                    When(\.$trailCollection, .hasAnyValue) {
                        Summary("Suggest \(\.$activity) trails from \(\.$trailCollec
                    } otherwise: {
                        Summary("Suggest \(\.$activity) trails from \(\.$trailCollec
                    }
                }
            }
        }
    }
```

For more information, see Parameter resolution.

Review the role of app entities

App entities provide the system with information about your app's data, or about concepts related to your app's data. App entities describe your app's custom data types you use for parameters, and help the system resolve parameters for app intents by letting it inspect relevant types. For example, a photo app that provides app entities for its photos and albums might also provide app entities to represent "the most recent photo" or "the default album." These specific app entities help resolve intents more quickly and with fewer verbal interactions.

Define app entities for core types and concepts that you want to make available to system experiences, and make sure to include properties for any data values that help people discover the entities using queries. For example, create an entity that describes a photo album and add a property to the entity for the name of the photo album.

For more information about expressing your app's data as entities, see Integrating custom data types into your intents.

See Also

Parameters, custom data types, and queries

- Integrating custom data types into your intents

  Provide the system with information about the types your app uses to model its data so that your intents can use those types as parameters.

- Parameter resolution

  Define the required parameters for your app intents and specify how to resolve those parameters at runtime.

- App entities

  Make core types or concepts discoverable to the system by declaring them as app entities.

- Entity queries

  Help the system find the entities your app defines and use them to resolve parameters.

- Resolvers

  Resolve the parameters of your app intents, and extend the standard resolution types to include your app's custom types.