Article

Displaying static and interactive snippets

Enable people to view the outcome of an app intent and immediately perform follow-up actions.

Overview

Using App Intents, you can integrate your app into the system, allowing people to perform actions from system experiences such as Spotlight or Control Center, the Action button, or Siri. To inform people about the outcome of an action you make available as an app intent, the intent can return a static snippet.

Many of the actions you make available to the system as app intents are simple, and happen quickly. App intents can return an interactive snippet that allows people to perform an action from a snippet instead of viewing static information. Instead of taking a person out of their current context by launching your app from the app intent if it needs further action from the person, you can change the app intent to display an interactive snippet that shows the intent's result, a way to confirm its action, or a button for a follow-up action.

For example, the Adopting App Intents to support system experiences sample app offers an app intent to view details about the landmarks nearby. People might use it to create shortcuts and perform its action from Spotlight or the Action button. When the app intent finds information about a nearby landmark, it displays an interactive snippet with the most important information, a button to add it to a list favorites, and a button to search for available tickets if the landmark requires people to pay an entrance fee.

Note

App intents that people perform from a control in Control Center can't display snippets.

Show a static snippet

If your app intent doesn't require a follow-up action, return a static snippet that enables someone to view the outcome of the app intent. To show a static snippet as a result from an app intent, return a view from your app intent's `perform()` method:

```
func perform() async throws -> some IntentResult {
    // ...

    return .result(view: Text("Some example text.").font(.title))
}
```

Return an interactive snippet

To display an interactive snippet as a result of an app intent, create an app intent for your action - or use an existing app intent. For example, the Adopting App Intents to support system experiences sample app provides landmark information might already have an app intent that finds a nearby landmark and returns information about it:

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

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

        return .result(
            value: landmarkEntity // Return information about the landmark.
        )
    }
}
```

To display a snippet instead of just returning the app entity, change your intent's `perform()` function to return a `SnippetIntent` in addition to the existing return value by adding `& ShowsSnippetIntent`. When you return a `ShowsSnippetIntent` result from the intent, you let the system know that the action displays an interactive snippet. In the Adopting App Intents to support system experiences example app, the previous example's updated `perform()` method might look like this:

```
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,
            snippetIntent: LandmarkSnippetIntent(landmark: landmark)
        )
    }
}
```

In the example, the intent returns a landmark entity by declaring `-> some ReturnsValue<LandmarkEntity>` and, additionally, returns a `LandmarkSnippetIntent`. This intent, an implementation of `SnippetIntent` , handles the snippet's layout and interactive components.

When you adopt interactive snippets, you might be able to reuse existing intents and add logic to display a snippet. Like the example above, you can return several results from an intent. By keeping an existing result type and additionally returning a snippet intent, you avoid breaking people's custom shortcuts that use the previous version of your intent.

Create the interactive snippet

As described in the previous section, the intent that performs your app's action can return a `SnippetIntent`. The snippet intent constructs your snippet's layout and returns it to the system, which then displays the interactive snippet. To return the views for your interactive snippet:

1. Create an app intent that conforms to `SnippetIntent`.

2. Make sure the intent's `perform()` method returns a `ShowsSnippetView`.

The following code continues the previous example and shows how the `AppIntentsTravelTracking` app might return a `LandmarkView` from the `SnippetIntent`:

```
import AppIntents
import SwiftUI

struct LandmarkSnippetIntent: SnippetIntent {
    static let title: LocalizedStringResource = "Landmark Snippet"

    @Parameter var landmark: LandmarkEntity
    @Dependency var modelData: ModelData

    func perform() async throws -> some IntentResult & ShowsSnippetView {
        let isFavorite = await modelData.isFavorite(landmark)
        return .result(
            view: LandmarkView(landmark: landmark, isFavorite: isFavorite)
        )
    }
}

extension LandmarkSnippetIntent {
    init(landmark: LandmarkEntity) {
        self.landmark = landmark
    }
}
```

Note the `isFavorite` parameter in the view's initializer. The `LandmarkView` indicates whether a landmark is marked as a favorite already, and includes a button to add or remove it from favorites. The `LandMarkView` also includes a button to start a search for tickets to visit the landmark.

> **Note**
>
> Snippets are great candidates to reuse views you use in your widgets, and, like widgets, you must initialize the snippet's `Button` or `Toggle` with an `AppIntent` that performs the underlying action. For more information, refer to Adding interactivity to widgets and Live Activities.

Review the lifecycle of snippet intents

A snippet remains visible until a person dismisses it, and, similar to SwiftUI views, the system and a person's actions might result in your `SnippetIntent` being created and performed multiple times during its lifecycle.

For example, the landmarks sample code project's snippet includes a Favorites button to add or remove a nearby landmark from a list of favorites. When a person taps the Favorites button, the system performs the `FavoriteLandmarkIntent` to make the change. It discards the snippet's old SwiftUI views, and performs the `SnippetIntent` again to provide a new version of the snippet to show that the person added or removed the landmark from the list of favorites.

> **Note**
>
> Treat intents that conform to `SnippetIntent` like any other intent; they aren’t limited to displaying a snippet. People can use them in their custom shortcuts, and you can reuse them. For example, the sample app might use `FavoriteLandmarkIntent` in an App Shortcut or interactive widget.

In the snippet intent’s `perform()` function, retrieve app state - for example, whether a current landmark is a favorite - and return an updated snippet as shown in the `LandmarkSnippetIntent` example code above.

Because the system creates and performs your `SnippetIntent` repeatedly, make sure calling its `perform()` method doesn’t produce side effects:

- If you pass data between intents, pass a minimum amount of immutable data.
- Avoiding long-running tasks to ensure the snippet appears responsive.
- Fetch dynamic values from a shared object instead of passing them around as parameters between intents; for example, the `LandmarkSnippetIntent` above uses an `AppDependency` for its `modelData`.

> **Note**
>
> As long as a snippet is visible, the system keeps your app active in the background, retaining all information in memory. As a result, you don’t have to persist or refresh data while your snippet is visible.

Create a sequence of snippets and request confirmation

With interactive snippets, you can create quick, fast-flowing interactions, allowing people to view content and perform a series of actions without leaving their current context. For example, the landmark example might display a sequence of three snippets:

1. When a person performs the “Find Closest” App Shortcut, the landmarks snippet described above appears. It includes a button to search for tickets for the nearby landmark.
2. When a person taps the button to search for tickets, a second snippet appears, requesting confirmation of the total number of tickets. When they’ve adjusted the number of tickets and confirmed the number of tickets, the search starts.
3. When the search finishes, a third snippet appears to display the total amount for the number of tickets and a buy button.

To create this sequence of snippets, the Adopting App Intents to support system experiences app uses a combination of regular app intents, a request for confirmation, and snippet intents.

First, the app defines the `FindTicketsIntent`, a regular app intent to perform the search. In its `perform()` method, the `requestConfirmation(conditions:actionName:dialog:showDialogAsPrompt:snippetIntent:)` API displays the interactive snippet for people to enter the tickets, using the `TicketRequestSnippetIntent`.

```
import AppIntents

struct FindTicketsIntent: AppIntent {

    // ...

    func perform() async throws -> some IntentResult & ShowsSnippetIntent {
        let searchRequest = await searchEngine.createRequest(landmarkEntity: landmar

        // Present a snippet that allows people to change
        // the number of tickets.
        try await requestConfirmation(
            actionName: .search,
            snippetIntent: TicketRequestSnippetIntent(searchRequest: searchRequest)
        )

        // ...
    }
}

// ...
```

When the person has entered the number of tickets in the snippet that `TicketRequestSnippetIntent` presents, they confirm the number of tickets and the search starts. The search results appear in a third snippet that the `TicketResultSnippetIntent` presents:

```
// ...

func perform() async throws -> some IntentResult & ShowsSnippetIntent {
    let searchRequest = await searchEngine.createRequest(landmarkEntity: landmark)

    // Present a snippet that allows people to change
    // the number of tickets.
    try await requestConfirmation(
```

```swift
        actionName: .search,
        snippetIntent: TicketRequestSnippetIntent(searchRequest: searchRequest)
    )

    // If the person has confirmed the action, perform the ticket search.
    try await searchEngine.performRequest(request: searchRequest)

    // Show the result of the ticket search.
    return .result(
        snippetIntent: TicketResultSnippetIntent(
            searchRequest: searchRequest
        )
    )
}

// ...
```

By displaying the snippet using the `requestConfirmation()` API, the snippet includes the option to cancel the action. If the person doesn't confirm the number of tickets, the app intent doesn't continue its `perform()` function, perform the search, or display another snippet.

Reload a snippet to display updated data

In the above example, three snippets appear in a sequence, each snippet replacing the previous snippet. If a snippet remains onscreen for some time; for example, if you perform a search like the in the example; reload the snippet to let people know that the search is ongoing. Similarly, reload the snippet if its underlying data changes.

To reload a snippet, use the `reload()` function defined by `SnippetIntent`. The following example adds it to the trailing closure of the search method:

```swift
// ...

func perform() async throws -> some IntentResult & ShowsSnippetIntent {
    // ...

    // If the person has confirmed the action, perform the ticket search.
    try await searchEngine.performRequest(request: searchRequest) {
        // Creates and reloads the TicketResultSnippetIntent.
        TicketResultSnippetIntent.reload()
    }
    // Show the result of the ticket search.
    return .result(
        snippetIntent: TicketResultSnippetIntent(
            searchRequest: searchRequest
        )
    )
}
```

> **Note**
>
> Calling `reload()` displays a snippet if it’s not visible and dismisses any other visible snippet. If the snippet is already visible, it reloads the snippet to display updated data.

See Also

Interactive Snippets

`protocol SnippetIntent`

An app intent that presents an interactive snippet onscreen.