Article

Increasing the visibility of widgets in Smart Stacks

Provide contextual information and donate intents to the system to make sure your widget appears prominently in Smart Stacks.

Overview

Smart Stacks efficiently organize multiple widgets to show useful content at just the right time. With input from your app and on-device intelligence, the system can identify patterns and determine a widget's relevance to a person, making sure a Smart Stack shows your widget at the right moment. Context helps the system increase your widget's visibility and gives people an quick, intelligent way to view content or perform an action. For example, if a person checks the daily weather forecast every morning or their calendar on the commute to work every morning, the system can suggest the corresponding widgets at the right moments.

On iPhone and iPad, people create Smart Stacks on their Home Screen with options for surfacing widgets in a Smart Stack:

Smart Rotate
: Automatically rotates widgets to the top of the stack when they have timely, relevant information to show.

Widget Suggestions
: Automatically propose widgets that a person doesn't already have in their stack, perhaps exposing them to widgets they don't even know exist.

On Apple Watch, people rely more on the Smart Stack to intelligently display widgets that best fit their personal context. Additionally, they can configure their favorite widgets to appear in the Smart Stack and pin them to a fixed position.

To make sure the system can accurately determine your widget's importance to someone, provide contextual and behavioral clues in your timeline provider with RelevanceKit and App Intents.

Provide relevance and contextual clues for each platform

To determine which widget is most relevant to a person in a particular context, the system's on-device intelligence processes information it gathers automatically. For example, it considers when and how often a person uses your app.

However, the system can't access in-app context — actions a person performs or content they view in your app. Only you can determine which in-app actions make your widget more important across Smart Stacks and only you can know when your widget's content is important. As a result, the system relies on additional clues from your app or widget extension to show your widget at the right time and increase its visibility.

Depending on the platform, provide the following clues to the system:

App-provided clue	iOS	iPadOS	watchOS
Add a relevance score and duration to your timeline entries	Yes	Yes	No
Donate intents to the system that conform to `PredictableIntent`	Yes	Yes	No
Implement your timeline provider's `relevance()` callback	No	No	Yes
Add app intents that conform to `RelevantIntent` to the `RelevantIntentManager`:	No	No	Yes, see Add watchOS relevance clues to your timeline provider
Create a widget that uses `RelevanceConfiguration` and a `RelevanceEntriesProvider` instead of a timeline provider	No	No	Yes, see Create a watchOS widget that uses a relevance configuration

> **Note**
>
> Smart Stacks on iPhone and iPad don't consider relevance information you provide with your timeline provider's `relevance()` callback, but the callback is available across platforms to enable code sharing. However, `RelevanceConfiguration` API is available in watchOS only.

Add a relevance score and duration to your timeline entries

On iPhone and iPad, one of the different clues on-device intelligence uses to determine the widget's position in the Smart Stack is the `relevance` property of the entries your `TimelineProvider` or `AppIntentTimelineProvider` generates. The property's `TimelineEntryRelevance` type contains a `score` and a `duration`. The score is a value you choose that indicates the relevance of the widget, relative to entries across timelines that the provider creates. When the date of an entry arrives, and for the duration you specify, WidgetKit may rotate your widget to the top of the Smart Stack, making it visible.

For example, the Emoji Rangers: Supporting Live Activities, interactivity, and animations sample code project sets an increased relevance if a configured hero hasn't recovered its full health:

```
func timeline(for configuration: EmojiRangerSelection, in context: Context) asyn
    let selectedCharacter = hero(for: configuration)
    let endDate = selectedCharacter.fullHealthDate
    let oneMinute: TimeInterval = 60
    var currentDate = Date()
    var entries: [SimpleEntry] = []

    while currentDate < endDate {
        let relevance = TimelineEntryRelevance(score: Float(selectedCharacter.he
        let entry = SimpleEntry(date: currentDate, relevance: relevance, hero: s
        currentDate += oneMinute
        entries.append(entry)
    }

    return Timeline(entries: entries, policy: .atEnd)
}
```

To determine a score for an individual entry, it's preferable to decide on an absolute scale from least relevant to most relevant. The range of the scale is up to you; for instance, 1 to 100, 50 to 5000, or any other range of positive numbers that's meaningful to you. WidgetKit calculates the relative difference of scores between entries; the absolute values don't matter.

Because WidgetKit maintains relevance information across timelines, use a consistent scale for all of them. For simplicity, you might use an enumeration with specific values for `low`, `medium`, and `high` relevance. Alternatively, if your score calculations have a continuous variation, you might use floating-point precision in your scale, with consistent minimum and maximum values.

A score of zero (`0`) or lower indicates that a widget isn't relevant, and WidgetKit won't consider rotating it to the top of the stack.

Donate app intents for increased visibility on iPhone and iPad

To help the system determine your widget's relevance, donate app intents to provide the system with *behavioral clues* about a person's current in-app actions. When you donate app intents, the system learns how and when people interact with your app and compares your donated app intents with app intents of configured widgets in the Smart Stack. If it finds a widget with a matching app intent, it can automatically rotate to it when a person typically looks for that information. If someone hasn't configured a matching widget in a Smart Stack, the system can automatically suggest your widget with the configuration that matches your donated app intent. This process increases the visibility of your widget.

For example, a tour guide app displays information for cities around the world, such as popular tourist attractions, restaurants, and upcoming festivals throughout the year. The app offers several configurable widgets that display these details. To offer configurable widgets, the app's widget configurations make use of custom intents, such as `CityInfoAppIntent`, `AttractionAppIntent`, `RestaurantAppIntent`, and `FestivalAppIntent`. As people interact with the app, the app donates these app intents to inform Smart Rotate and Widget Suggestions in Smart Stacks.

> **Note**
>
> App intent donations inform much more than Smart Stacks. The system uses donated app intents to display Spotlight search results, offer shortcuts on the Lock Screen, and more. For more information about donating app intents, refer to Intent discovery.

To enable Smart Stacks to suggest your widget or show it more prominently:

1. Describe an app action as a custom app intent that conforms to `PredictableIntent`.
2. If your widget is configurable, make sure the predictable intent contains a `predictionConfiguration` that matches the intent's values to the configurable parameters of your widget. If your widget isn't configurable, don't provide a parameter matching.
3. Donate the app intent to the system using `IntentDonationManager` or the intent's `donate()` functionality.

Continuing the example with the tour guide app, a person frequently uses the app on the weekend to plan a trip to France. Using the app, they look for festivals in various cities they may visit. Each time they search for or view the details of a festival, the app donates a `FestivalAppIntent`. If the `FestivalAppIntent` contains `location` and `type` parameters, the system can learn that the person frequently searches for art events in Paris. Based on this learned information, Smart Stacks can automatically rotate to the widget that shows information about the art festival in Paris when the person typically looks for that information. If Smart Stacks don't contain a matching

widget, the system can automatically add the configured widget as a Widget Suggestion, giving the app more visibility and the person instant access to the information.

> **Note**
>
> For more information about creating configurable widgets, refer to Making a configurable widget.

Choose how to provide relevance clues on Apple Watch

On Apple Watch, widgets automatically appear in the Smart Stack based on *contextual clues* it requests from your widget — as opposed to behavioral cues your app provides that inform the Smart Stack on iPhone and iPad. When a person's context matches one of your widget's contextual clues, such as their location or bedtime, or an inferred location such as their workplace, the Smart Stack may show your widget more prominently.

In watchOS, you can choose between two options to provide relevance information:

Using a timeline
: Provide relevance information using your timeline provider to enable watchOS to intelligently display one widget in the Smart Stack when it matches a person's context. Additionally, people can configure your widget to appear in the Smart Stack and pin it to a fixed position.

Using a relevance entries provider
: watchOS widgets can use a `RelevanceConfiguration` and a `RelevanceEntriesProvider` to provide widget data and relevance clues. Using a a relevance entries provider instead of a timeline provider, watchOS can intelligently show a widget in the Smart Stack for each relevance clue that matches a person's context, making your app's content more visible. For example, a weather widget might provide a relevance clue for someone's current location and a second relevance clue when it's time to leave for an upcoming trip. As a result, the widget might appear in the Smart Stack twice, showing different weather data. One widget shows weather data for their current physical location, the other shows weather data for the destination of their trip shortly before they have to leave. However, people can't configure widgets that use a `RelevanceConfiguration` to appear in the Smart Stack, add them to the Smart Stack, or pin them to a fixed location.

Provide relevance clues in your timeline provider if your widget's data changes on a relatively predicable schedule and always has timely data to display. Provide relevance clues using a relevance entries provider if your widget should automatically show information for one of the available `RelevantContext` clues. Additionally, choose a `RelevanceEntriesProvider` if your widget should appear automatically in the Smart Stack several times or if it shouldn't appear in the Smart Stack unless it matches a person's context. For example, a widget might only be relevant at a person's inferred work location and not at their inferred home location.

Depending on your app's functionality, you might want to support both or only one of the options:

- If your existing app is only available on Apple Watch and you want to offer an additional widget that can appear several times in the Smart Stack, create separate widgets for each option.
- If you create a new app that's only available in watchOS, create separate widgets for each option.
- If your app supports iOS and watchOS, use a timeline provider to provide relevance clues to share code across platforms and consider adding a watchOS widget that uses a `RelevanceConfiguration`.

> **Important**
>
> To provide contextual clues about a person's location, workouts, or bedtime, make sure your app and your widget extension request a person's permission to access location, workout, or sleep schedule information. For example, you need to request the `sleepAnalysis` permission to provide a clue based on a person's sleep schedule. For more information, refer to `fitness(_:)`, `sleep(_:)`, and `location(inferred:)`. For general information about accessing a person's location in widgets, refer to Accessing location information in widgets.

Add watchOS relevance clues to your timeline provider

If your widget uses a `TimelineProvider`, `AppIntentTimelineProvider`, or `IntentTimelineProvider`, watchOS queries your widget extension periodically using the timeline provider's `relevance()` requirement to gather contextual clues.

To provide the system with contextual clues using a timeline provider:

1. Create app intents that conform to `RelevantIntent` and include a `RelevantContext` that describes conditions such as time, location, workouts, sleep schedule, and more.
2. Return the relevant intents in the `relevance()` callback of your widget's `TimelineProvider` or `AppIntentTimelineProvider` implementation.
3. When you create or update your widget's timeline, set each timeline entry's `relevance` property and create or update your relevant intents using the `updateRelevantIntents(_:)` function of the `RelevantIntentManager`.
4. Call `updateRelevantIntents(_:)` to update relevance information every time it changes for your app; for example, when you receive updated data.

The following example shows how a gaming app might allow people to track the health of their hero and provide contextual clues to the system:

```swift
import SwiftUI
import WidgetKit
import AppIntents

struct AppIntentProvider: AppIntentTimelineProvider {

    typealias Entry = SimpleEntry

    typealias Intent = EmojiRangerSelection

    func placeholder(in context: Context) -> SimpleEntry {
        // Code that returns a placeholder entry.
    }

    func snapshot(for configuration: EmojiRangerSelection, in context: Context) asyn
        // Code that returns a snapshot entry.
    }

    func timeline(for configuration: EmojiRangerSelection, in context: Context) asyn
        // Code that creates timeline entries.
        // ...

        // Update relevant intents with each timeline refresh.
        await updateRangerRelevantIntents()

        // Code that returns a timeline.
    }

    func recommendations() -> [AppIntentRecommendation<EmojiRangerSelection>] {
        [] // Returns an empty array that makes the widget configurable.
    }

    // Provide contextual clues that inform the system about contexts in which
    // your widget is especially important to a person.
    func relevance() async -> WidgetRelevance<Self.Intent> {
        let relevances = EmojiRanger.allHeros.map { hero in
            let rangerIntent = EmojiRangerSelection()
            rangerIntent.hero = hero
            let relevantRangerContext = RelevantContext.date(from: hero.injuryDate,
            return WidgetRelevanceAttribute(configuration: rangerIntent, context: re
        }
        return WidgetRelevance(relevances)
    }
```

```
    // A function that updates your relevant intents.
    func updateRangerRelevantIntents() async {
        var relevantIntents: [RelevantIntent] = []

        // Create relevant intents.
        // The code below is intentionally simple to make it easy to understand.
        for hero in EmojiRanger.allHeros {
            let configuredRangerIntent = EmojiRangerSelection()
            configuredRangerIntent.hero = hero
            let relevantRangerContext = RelevantContext.date(from: hero.injuryDate,

            let relevantIntent = RelevantIntent(
                configuredRangerIntent,
                widgetKind: EmojiRanger.EmojiRangerWidgetKind,
                relevance: relevantRangerContext
            )
            relevantIntents.append(relevantIntent)
        }

        do {
            try await RelevantIntentManager.shared.updateRelevantIntents(relevantInt
        } catch {
            // Handle error cases.
        }
    }
}
```

Create a watchOS widget that uses a relevance configuration

On Apple Watch, enable the system to display relevant widgets in the Smart Stack based on relevance information you provide with a `RelevanceEntriesProvider`:

1. Configure your watchOS widget using `RelevanceConfiguration`.
2. Adopt `RelevanceEntriesProvider` and implement its `entry(configuration:context:)`, `placeholder(context:)`, and `relevance()` requirements.
3. Make sure you return each `RelevanceEntry`. If you don't provide entries, the widget won't appear in the Smart Stack.

The following code snippet shows how a gaming app that allows people to track the health level of their heroes initializes a watchOS widget with a `RelevanceConfiguration` and provides

relevance information with a `RelevanceEntriesProvider`:

```
// Create the widget with a relevance configuration.
@available(watchOS 12, *)
struct EmojiRangerWidget: Widget {
    var body: some WidgetConfiguration {
        RelevanceConfiguration(
            kind: EmojiRanger.EmojiRangerWidgetKind,
            provider: EmojiRangerRelevanceProvider()
        ) { entry in
            EmojiRangerWidgetView(entry: entry)
        }
        .configurationDisplayName("Emoji Ranger")
        .description("Healing status")
    }
}

// Implement the relevance provider and its requirements.
@available(watchOS 12, *)
struct EmojiRangerRelevanceProvider: RelevanceEntriesProvider {
    // Provide all contextual clues.
    func relevance() async -> WidgetRelevance<EmojiRangertConfigurationIntent> {
        let healingUpdates = ... // Code to retrieve healing update information
        let attributes = healingUpdates.map { update in
            // Create relevant contexts objects.
            let context = RelevantContext.date(
                interval: update.date,
                kind: .scheduled
            )
            WidgetRelevanceAttribute(
                configuration: EmojiRangertConfigurationIntent(update: update),
                context: context
            )
        }
        return WidgetRelevance(attributes)
    }

    // Provide all data entries for the widget.
    func entry(
        configuration: Configuration,
        context: Context
    ) async throws -> EmojiRangersRelevanceEntry {
```

```
        if context.isPreview {
            return EmojiRangersRelevanceEntry.previewEntry
        }
        return EmojiRangersRelevanceEntry(
            hero: configuration.update?.hero,
            update: configuration.update
        )
    }

    func placeholder(context: Context) -> EmojiRangersRelevanceEntry {
        EmojiRangersRelevanceEntry.placeholderEntry
    }
}
```

If you offer timeline-based and widgets that uses relevance clues, a person could pin the timeline widget to the Smart Stack. When several instances of the relevance-based widget become relevant, the Smart Stack could run out of space. To ensure that the Smart Stack can display the most relevant widgets, tell the system that your relevance-based widget can replace the timeline-based widget using the `associatedKind(_:)` modifier.

Test widgets in Smart Stacks

When a widget is in a Smart Stack, WidgetKit normally limits the number of times it rotates the widget to the top of the stack or when it inserts the widget as a suggestion. To bypass this limit during development, enable the WidgetKit Developer Mode switch in Settings > Developer.

For additional tips about debugging widgets, refer to Debugging widgets and Previewing widgets and Live Activities in Xcode.

Provide backward compatibility

In iOS 16, macOS 13, watchOS 9 and earlier, configurable widgets use SiriKit and the system determines your widget's relevance based on donations of SiriKit intents. For details about supporting older operating system versions, refer to Migrating widgets from SiriKit Intents to App Intents.

Donating SiriKit Intents is very similar to donating app intents. Instead of using `IntentDonationManager`, you use `INInteraction` to donate an `INIntent`. For more information about donating SiriKit Intents, refer to Donating Shortcuts and Apple Watch support.

See Also

Smart Stacks

`struct` `TimelineEntryRelevance`

An object that describes the relative importance of a timeline entry compared to other entries in the current and past timelines.

`struct` `RelevanceConfiguration`

A type that describes the content of a widget that uses relevance clues.

`protocol` `RelevanceEntriesProvider`

A type that provides the content for a widget that uses relevance clues to display information in the Smart Stack.

`protocol` `RelevanceEntry`

A type that specifies the information to render a widget at a specific relevance configuration.

`struct` `WidgetRelevance`

A type collecting the relevances for a widget kind.

`struct` `WidgetRelevanceAttribute`

A type that describes when a specific widget could be relevant.

`struct` `WidgetRelevanceGroup`

A type for configuring widget behavior in the watchOS Smart Stack.

`struct` `AppIntentRecommendation`

An object that describes a recommended intent configuration for a user-customizable widget.

`struct` `IntentConfiguration`

An object describing the content of a widget that uses a custom intent definition to provide user-configurable options.

`struct` `IntentRecommendation`

An object that describes a recommended intent configuration for a user-customizable widget.