Article

Playing haptic feedback in your app

Provide tactile feedback when people perform certain actions in your app.

Overview

Haptic feedback provides a tactile response, such as a tap, that draws attention and reinforces both actions and events. While many system-provided interface elements (for example, pickers, switches, and sliders) automatically provide haptic feedback, you can add feedback to custom views and controls in your app when it's suitable.

Use feedback intentionally

When providing feedback:

- Always use feedback for its intended purpose. Don't select a haptic because of the way it feels.
- The source of the feedback must be clear to the user. For example, the feedback must match a visual change in the user interface, or must be in response to a user action. Feedback should never come as a surprise.
- Don't overuse feedback. Overuse can cause confusion and diminish the feedback's significance.

For design guidance, read Human Interface Guidelines > Playing haptics.

Choose the type of feedback

SwiftUI and UIKit have different APIs for providing haptic feedback. Learn more about each style of haptic feedback and choose what makes sense for your app.

SwiftUI UIKit

To learn more about different types of feedback in SwiftUI, read `SensoryFeedback`.

Associate the feedback with a view

To play haptic feedback in your app, you need to add the feedback to a view.

SwiftUI UIKit

The following SwiftUI code example shows how to associate selection feedback with a view.

Add a `sensoryFeedback( :trigger:)` view modifier to your view. For the `trigger` parameter, pass a value to monitor for changes.

```
@State private var showAccessory = false

ContentView()
    .sensoryFeedback(.selection, trigger: showAccessory)
```

Define when to play feedback

Haptic feedback occurs in response to something, like an action or event. You need to define what to trigger feedback in response to.

Using the feedback APIs in SwiftUI and UIKit doesn't play haptics directly. Instead, it informs the system of the event. The system then determines whether to play the haptics based on the device, the app state, the amount of battery power remaining, and other factors.

For example, haptic feedback plays only:

- On a device with hardware for haptic feedback
- When the app is running in the foreground
- When the system Haptics setting is on

Not all types of haptic feedback play on every type of device. As a general rule, trust the system to determine whether it should play feedback. Don't check the device type or app state to conditionally trigger feedback. After you decide how you want to use feedback, always trigger it when the appropriate events occur. The system ignores any requests that it can't fulfill.

Play feedback for selection events

Use selection feedback to communicate movement through a series of discrete values. For example, you might trigger selection feedback to indicate that a UI element's values are changing.

SwiftUI UIKit

The following SwiftUI code example shows how to use a long-press gesture to toggle an accessory view, playing haptic feedback to indicate when the accessory appears or disappears.

```
struct MyView: View {
    @State private var showAccessory = false

    var body: some View {
        ContentView()
            .sensoryFeedback(.selection, trigger: showAccessory)
            .onLongPressGesture {
                showAccessory.toggle()
            }

        if showAccessory {
            MyAccessoryView()
        }
    }
}
```

Play feedback for canvas events

Use canvas feedback to indicate when a drawing event occurs, such as an object snapping to a guide or ruler. When using Apple Pencil Pro with a compatible iPad, this type of feedback can provide a tactile response.

SwiftUI UIKit

The following SwiftUI code example shows how to use a drag gesture to drag a square, playing haptic feedback to indicate when the square aligns with a gridline on the canvas.

```
// The translation of the currently active drag gesture.
@GestureState private var translation = CGSize.zero

// The offset of the square as a result of the drag gesture.
@State private var offset = CGSize.zero
```

```
// A Boolean that indicates if the square currently aligns with a gridline.
@State private var isAligned = false

var drag: some Gesture {
    DragGesture()
        .updating($translation) { drag, translation, _ in
            translation = drag.translation
        }
        .onChanged { drag in
            isAligned = aligned(at: drag.location)
        }
        .onEnded { drag in
            offset = CGSize(
                width: offset.width + drag.translation.width,
                height: offset.height + drag.translation.height
            )
        }
}

var body: some View {
    Rectangle()
        .sensoryFeedback(.alignment, trigger: isAligned) { oldValue, newValue in
            // Plays feedback only when the square aligns with a gridline, but didn'
            !oldValue && newValue
        }
        .frame(width: 100, height: 100)
        .offset(CGSize(
            width: offset.width + translation.width,
            height: offset.height + translation.height
        ))
        .gesture(drag)
}
```

See Also

Related reference in SwiftUI

```
nonisolated func sensoryFeedback<T>(_ feedback: SensoryFeedback,
trigger: T) -> some View where T : Equatable
```

Plays the specified `feedback` when the provided `trigger` value changes.

`nonisolated func sensoryFeedback<T>(trigger: T, _ feedback: @escaping () -> SensoryFeedback?) -> some View where T : Equatable`

Plays feedback when returned from the `feedback` closure after the provided `trigger` value changes.

`nonisolated func sensoryFeedback<T>(_ feedback: SensoryFeedback, trigger: T, condition: @escaping (T, T) -> Bool) -> some View where T : Equatable`

Plays the specified `feedback` when the provided `trigger` value changes and the `condition` closure returns `true`.

`struct SensoryFeedback`

Represents a type of haptic and/or audio feedback that can be played.

Related reference in UIKit

`@MainActor class UIFeedbackGenerator`

The abstract superclass for all feedback generators.

`@MainActor class UICanvasFeedbackGenerator`

A concrete feedback generator subclass that creates haptics to indicate events on a drawing canvas.

`@MainActor class UIImpactFeedbackGenerator`

A concrete feedback generator subclass that creates haptics to simulate physical impacts.

`@MainActor class UISelectionFeedbackGenerator`

A concrete feedback generator subclass that creates haptics to indicate a change in selection.

`@MainActor class UINotificationFeedbackGenerator`

A concrete feedback generator subclass that creates haptics to communicate successes, failures, and warnings.