Sample Code

Backyard Birds: Building an app with SwiftData and widgets

Create an app with persistent data, interactive widgets, and an all new in-app purchase experience.

Download

iOS 17.2+ | iPadOS 17.2+ | macOS 14.2+ | watchOS 10.2+ | Xcode 15.1+

Overview

Backyard Birds offers a rich environment in which you can watch the birds that visit your backyard garden. You can monitor their water and food supply to ensure they always have fresh water and plenty to eat, or upgrade the game using an in-app purchase to provide tastier food for the birds to eat.

The sample implements its data model using SwiftData for persistence, and integrates seamlessly with SwiftUI using the `Observable` protocol. The game's widgets implement App Intents for interactive and configurable widgets. The in-app purchase experience uses the `ProductView` and `SubscriptionStoreView` from StoreKit.

You can access the source code for this sample on GitHub.

> Note
>
> This sample code project is associated with WWDC23 session 102: State of the Union.

Configure the sample code project

To configure the Backyard Birds app to run on your devices, follow these steps:

1. Open the project in Xcode 15 or later.
2. Edit the multiplatform target’s scheme, and on the Options tab, choose the `Store.storekit` file for StoreKit configuration.
3. Repeat the previous step for the watchOS target’s scheme.
4. Select the top-level Backyard Birds project.
5. For all targets, choose your team from the Team menu in the Signing & Capabilities pane so Xcode can automatically manage your provisioning profile.

Create a data-driven app

The app defines its data model by conforming the model objects to `PersistentModel` using the `Model` macro. Using the `Attribute` macro with the `unique` option ensures that the `id` property is unique.

```
@Model public class BirdSpecies {
    @Attribute(.unique) public var id: String
    public var naturalScale: Double
    public var isEarlyAccess: Bool
    public var parts: [BirdPart]

    @Relationship(deleteRule: .cascade, inverse: \Bird.species)
    public var birds: [Bird] = []

    public var info: BirdSpeciesInfo { BirdSpeciesInfo(rawValue: id) }

    public init(info: BirdSpeciesInfo, naturalScale: Double = 1, isEarlyAccess: Boo
        self.id = info.rawValue
        self.naturalScale = naturalScale
        self.isEarlyAccess = isEarlyAccess
        self.parts = parts
    }
}
```

Construct interactive widgets

Backyard Birds displays interactive widgets by presenting a `Button` to refill a backyard’s supplies when the water and food are running low. The app does this by placing a `Button` in the widget’s view, and passing a `ResupplyBackyardIntent` instance to the `init(intent:label:)` initializer:

```
Button(intent: ResupplyBackyardIntent(backyard: BackyardEntity(from: snapshot.backya
    Label("Refill Water", systemImage: "arrow.clockwise")
        .foregroundStyle(.secondary)
        .frame(maxWidth: .infinity)
        .padding(.vertical, 8)
        .padding(.horizontal, 12)
        .background(.quaternary, in: .containerRelative)
}
```

The app allows for configuration of the widget by implementing the WidgetConfiguration Intent protocol:

```
struct BackyardWidgetIntent: WidgetConfigurationIntent {
    static let title: LocalizedStringResource = "Backyard"
    static let description = IntentDescription("Keep track of your backyards.")

    @Parameter(title: "Backyards", default: BackyardWidgetContent.all)
    var backyards: BackyardWidgetContent

    @Parameter(title: "Backyard")
    var specificBackyard: BackyardEntity?

    init(backyards: BackyardWidgetContent = .all, specificBackyard: BackyardEntity?
        self.backyards = backyards
        self.specificBackyard = specificBackyard
    }

    init() {
    }

    static var parameterSummary: some ParameterSummary {
        When(\.$backyards, .equalTo, BackyardWidgetContent.specific) {
            Summary {
                \.$backyards
                \.$specificBackyard
            }
        } otherwise: {
            Summary {
                \.$backyards
            }
        }
    }
}
```

Provide a new in-app purchase experience

The sample app uses `ProductView` to display several different bird food upgrades available for purchase on a store shelf. To prominently feature an in-app purchase item, the app uses the `.productViewStyle(.large)` modifier:

```
ProductView(id: product.id) {
    BirdFoodProductIcon(birdFood: birdFood, quantity: product.quantity)
        .bestBirdFoodValueBadge()
}
.padding(.vertical)
.background(.background.secondary, in: .rect(cornerRadius: 20))
.productViewStyle(.large)
```

The Backyard Birds Pass page displays renewable subscriptions using the `SubscriptionStoreView` view. The app uses the `PassMarketingContent` view as the content of the `SubscriptionStoreView`:

```
SubscriptionStoreView(
    groupID: passGroupID,
    visibleRelationships: showPremiumUpgrade ? .upgrade : .all
) {
    PassMarketingContent(showPremiumUpgrade: showPremiumUpgrade)
        #if !os(watchOS)
        .containerBackground(for: .subscriptionStoreFullHeight) {
            SkyBackground()
        }
        #endif
}
```

See Also

Creating an app

{} Destination Video

Leverage SwiftUI to build an immersive media experience in a multiplatform app.

{} Hello World

Use windows, volumes, and immersive spaces to teach people about the Earth.

{} Food Truck: Building a SwiftUI multiplatform app

Create a single codebase and app target for Mac, iPad, and iPhone.

{} Fruta: Building a feature-rich app with SwiftUI

Create a shared codebase to build a multiplatform app that offers widgets and an App Clip.

Migrating to the SwiftUI life cycle

Use a scene-based life cycle in SwiftUI while keeping your existing codebase.

`protocol App`

A type that represents the structure and behavior of an app.