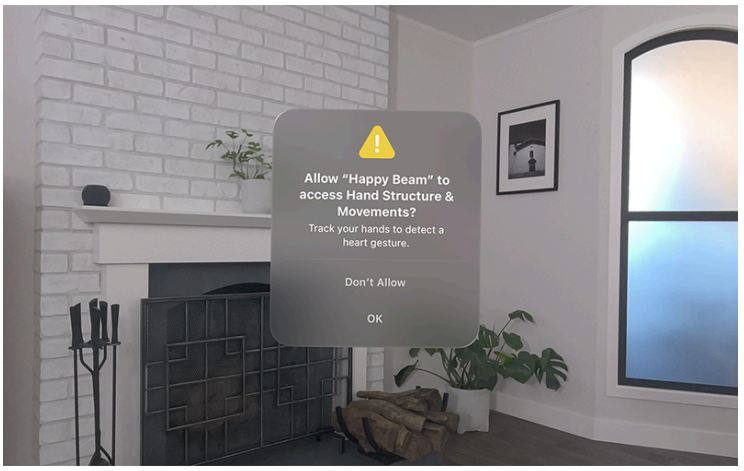
Article

Setting up access to ARKit data

Check whether your app can use ARKit and respect people's privacy.

Overview

In visionOS, ARKit can enable new kinds of experiences that leverage data such as hand tracking and world sensing. The system gates access to this kind of sensitive information. Because people can decline your app's request to use ARKit data or revoke access later, you need to provide alternative ways to use your app and to handle cases where your app loses access to data.



Add usage descriptions for ARKit data access

People need to know why your app wants to access data from ARKit. Add the following keys to your app's information property list to provide a user-facing usage description that explains how your app uses the data:

`NSHandsTrackingUsageDescription`
: Use this key if your app uses hand tracking.

`NSWorldSensingUsageDescription`
: Use this key if your app uses image tracking, plane detection, or scene reconstruction.

> **Note**
>
> World *tracking* — unlike world *sensing* — doesn't require authorization. For more information, see Tracking specific points in world space.

Choose between up-front or as-needed authorization

You can choose when someone sees an authorization request to use ARKit data. If you need precise control over when the request appears, call the `requestAuthorization(for:)` method on `ARKitSession` to explicitly authorize access at the time you call it. Otherwise, people see an authorization request when you call the `run(_:)` method. This is an implicit authorization because the timing of the request depends entirely on when you start the session.

Open a space and run a session

To help protect people's privacy, ARKit data is available only when your app presents a Full Space and other apps are hidden. Present one of these space styles before calling the `run(_:)` method.

The following shows an app structure that's set up to use a space with ARKit:

```
@main
struct MyApp: App {
    @State var session = ARKitSession()
    @State var immersionState: ImmersionStyle = .mixed
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
        ImmersiveSpace(id: "appSpace") {
            MixedImmersionView()
            .task {
                let planeData = PlaneDetectionProvider(alignments: [.horizontal])

                if PlaneDetectionProvider.isSupported {
                    do {
                        try await session.run([planeData])
                        for await update in planeData.anchorUpdates {
                            // Update app state.
                        }
                    } catch {
                        print("ARKit session error \(error)")
```

```
                    }
                }
            }
        }
        .immersionStyle(selection: $immersionState, in: .mixed)
    }
}
```

Call `openImmersiveSpace` from your app's user interface to create a space, start running an ARKit session, and kick off an immersive experience. The following shows a simple view with a button that opens the space:

```
struct ContentView: View {
    @Environment(\.openImmersiveSpace) private var openImmersiveSpace

    var body: some View {
        Button("Start ARKit experience") {
            Task {
                await openImmersiveSpace(id: "appSpace")
            }
        }
    }
}
```

Provide alternatives for declined and revoked authorizations

Someone might not want to give your app access to data from ARKit, or they might choose to revoke that access later in Settings. Handle these situations gracefully, and remove or transition content that depends on ARKit data. For example, you might fade out content that you need to remove, or recenter content to an appropriate starting position. If your app uses ARKit data to place content in a person's surroundings, consider letting people place content using the system-provided interface.

Providing alternatives is especially important if you're using ARKit for user input. People using accessibility features, trackpads, keyboards, or other forms of input might need a way to use your app without ARKit.

See Also

ARKit

{} Happy Beam

Leverage a Full Space to create a fun game using ARKit.

{} Incorporating real-world surroundings in an immersive experience

Create an immersive experience by making your app's content respond to the local shape of the world.

{} Placing content on detected planes

Detect horizontal surfaces like tables and floors, as well as vertical planes like walls and doors.

{} Tracking specific points in world space

Retrieve the position and orientation of anchors your app stores in ARKit.

Tracking preregistered images in 3D space

Place content based on the current position of a known image in a person's surroundings.

{} Exploring object tracking with ARKit

Find and track real-world objects in visionOS using reference objects trained with Create ML.

{} Object tracking with Reality Composer Pro experiences

Use object tracking in visionOS to attach digital content to real objects to create engaging experiences.

{} Building local experiences with room tracking

Use room tracking in visionOS to provide custom interactions with physical spaces.

{} Placing entities using head and device transform

Query and react to changes in the position and rotation of Apple Vision Pro.