Sample Code

Adding synthesized speech to calls

Provide a more accessible experience by adding your app's audio to a call.

Download

iOS 18.2+ | iPadOS 18.2+ | Xcode 16.1+

Overview

This sample shows how to create an accessibility app that supports augmentative and alternative communication (AAC) by adding synthesized speech to a call. This feature is available in iOS 18.2 and visionOS 2.2 and later, and is available to use with calling apps that capture microphone input using Apple's voice processing like Phone, FaceTime, and most VoIP apps.

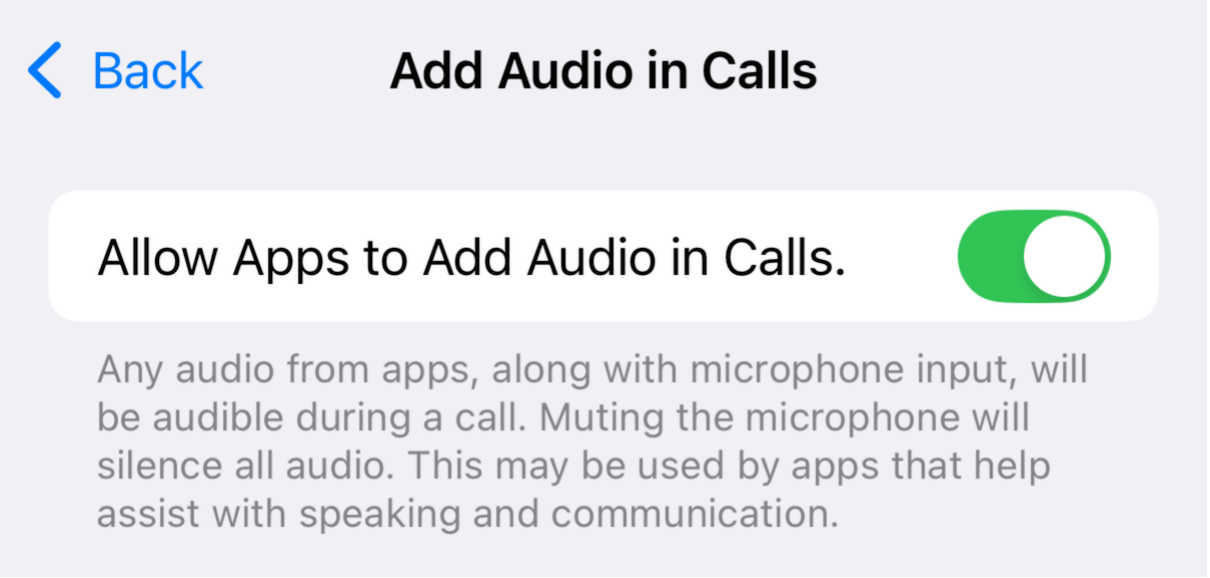
The sample app provides a basic user interface with a button to toggle the feature state and a text field. When you enter text into the field and press enter, the app speaks the phrase. If you have an active call in progress and you enable adding the app's audio to it, you'll hear the synthesized speech on the originating and receiving ends of the call.

Configure the sample code project

The sample requires running on an iOS device with iOS 18.2 or later. To test the sample, establish a phone or FaceTime call with another device.

Enable the accessibility service

Before an app can add its audio to calls, a person must turn on a system-level service in the Settings app by choosing Accessibility > Audio & Visual > Add Audio in Calls. This setting is global to the device and influences the availability of the service for all apps.



The sample determines the state of this setting by querying the shared `AVAudioApplication` object for its microphone injection permission:

```
// Retrieve the current microphone injection permission.
let permission = AVAudioApplication.shared.microphoneInjectionPermission
```

A permission value of `AVAudioApplication.MicrophoneInjectionPermission.serviceDisabled` indicates the person hasn't turned on the service, which means apps can't add audio to calls. When the app retrieves this value, it presents an alert dialog that indicates the current state and provides the person an opportunity to update their setting. When you press the dialog's Open Settings button, the app uses the Accessibility framework to directly open the Add Audio in Calls screen in the Settings app like shown below.

```
@ViewBuilder
var alertButtons: some View {
    Button("Open Settings") {
        Task {
            do {
                // Open the configuration screen for this feature in the Settings ap
                try await AccessibilitySettings.openSettings(for: .allowAppsToAddAud
            } catch {
                print("Unable to open Settings app: \(error)")
            }
        }
    }
    Button("Cancel", role: .cancel) {}
}
```

Request permission

Turning on Add Audio in Calls makes the feature available to apps on the system, but apps must explicitly request and be granted permission to use the feature. The sample determines its permission by querying for the current microphone injection permission. A value of `AVAudioApplication.MicrophoneInjectionPermission.undetermined` indicates the app hasn't yet requested permission and needs to before it can use the feature.

For an app to request a person's permission, it needs to provide an `NSMicrophoneInjectionUsageDescription` key in its `Info.plist` file with a description of why the app requests microphone access. The system displays this string when an app requests user permission. The sample app defines this entry as follows.

```
<key>NSMicrophoneInjectionUsageDescription</key>
<string>This app adds its audio to calls to support augmentative and alternative com
```

Attempting to request permission without this usage string present results in the system quitting the app.

The app requests permission by calling the `requestRecordPermission(completionHandler:)` method and awaiting a response:

```
// If undetermined, prompt the person to grant the app access, and turn on the featu
if await AVAudioApplication.requestMicrophoneInjectionPermission() == .granted {
    // If the person grants access, turn on adding app audio.
}
```

Calling this method causes the system to present a dialog that requests user permission. If a person grants the app permission, the call returns a value of `AVAudioApplication.MicrophoneInjectionPermission.granted` and the app updates its state accordingly.

If a person denies the app permission, the system sets the microphone injection permission state to `AVAudioApplication.MicrophoneInjectionPermission.denied`, and the app is unable to use the feature. The app remains in this state until a person explicitly changes the permission from the Add Audio in Calls screen in the Settings app. If you attempt to turn on the feature in the app while in this state, the app presents an alert similar to the one shown in the previous section to update the app's permission.

Enable adding audio to calls

When the app has permission to add audio and a person toggles the feature state in the user interface, the app responds by calling the shared `AVAudioSession` object's `setPreferred`

`MicrophoneInjectionMode(_:)` method. To turn on the feature, the app passes the method a value of `AVAudioSession.MicrophoneInjectionMode.spokenAudio`; to turn off the feature, it passes a value of `AVAudioSession.MicrophoneInjectionMode.none`.

```
let mode: AVAudioSession.MicrophoneInjectionMode = // spoken audio or none
try AVAudioSession.sharedInstance().setPreferredMicrophoneInjectionMode(mode)
```

When turned on during an active call, the system plays the app's audio locally and adds it to the microphone's input stream.

> **Note**
>
> The `AVAudioSession/setPreferredMicrophoneInjectionMode(_:)` method uses the word *preferred* to indicate that an app can set its preference, but a person ultimately determines whether they allow the feature's use.

Monitor the availability of calls

To determine whether a call can use this feature, the sample awaits notification of changes to the state of the audio session's microphone injection capabilities:

```
/// Monitor the active state of phone and FaceTime calls.
private func observeCallState() async {
    // Await notification of changes to the audio session's microphone injection cap
    for await notification in NotificationCenter.default.notifications(named: AVAudi
        // Inspect the user information dictionary to determine whether microphone i
        isCallActive = notification.userInfo?[AVAudioSessionMicrophoneInjectionIsAva
    }
}
```

When a call begins or ends, the system posts a notification of the change. The app queries the notification's user information dictionary for its `AVAudioSessionMicrophoneInjectionIsAvailableKey` value to determine whether there's an active call. When the value is `true`, the app updates its UI to show a pulsing phone icon in the toolbar to indicate the call is active.

See Also

System audio

- Handling audio interruptions

  Observe audio session notifications to ensure that your app responds appropriately to interruptions.

- Responding to audio route changes

  Observe audio session notifications to ensure that your app responds appropriately to route changes.

- Routing audio to specific devices in multidevice sessions

  Map audio channels to specific devices in multiroute sessions for recording and playback.

- {} Capturing stereo audio from built-In microphones

  Configure an iOS device's built-in microphones to add stereo recording capabilities to your app.

`class AVAudioSession`

An object that communicates to the system how you intend to use audio in your app.

`class AVAudioApplication`

An object that manages one or more audio sessions that belong to an app.

`class AVAudioRoutingArbiter`

An object for configuring macOS apps to participate in AirPods Automatic Switching.