Sample Code

Translating text within your app

Display simple system translations and create custom translation experiences.

Download

iOS 18.0+ | iPadOS 18.0+ | macOS 15.0+ | Xcode 16.0+

Overview

With the Translation framework, you can offer in-app translations from one language to another. To display simple system translations in your app, use the `translationPresentation(isPresented:text:attachmentAnchor:arrowEdge:replacementAction:)` translation overlay.

For a more customizable translation experience, use the `TranslationSession` object and its associated translation methods. With the customizable translation APIs, the framework asks a person for permission to download the language translation models, if necessary. You can translate strings efficiently, while checking for language availability before offering a translation.

This sample code project provides the `TranslatingText` sample app to demonstrate various methods of translating text. It consists of the following translation demos:

- Translate Text
- Replace Text
- Single String
- Batch All at Once
- Batch as a Sequence
- Language Availability
- Prepare for Translation

Note

This sample code project is associated with WWDC24 session 10117: Meet the Translation API.

Configure the sample code project

Before you run the sample code project, download and install Xcode 16 or later.

Note

This sample code project runs only in macOS and on physical iOS devices. It doesn’t translate text in iOS or iPadOS simulators.

To run the sample code project in macOS, select My Mac from the Xcode toolbar scheme menu, and then choose Project > Run.

To run the sample code on a physical iOS device:

1. Upgrade your device to iOS 18 or later.
2. Select your device from the Xcode toolbar scheme menu.
3. Select the `TranslatingText` project.
4. Click the `TranslatingText` target and select your team from the Team menu in the Signing & Capabilities pane to let Xcode automatically manage your provisioning profile. For more information, see Assign a project to a team.
5. Choose Project > Run.

Translate text with the simple translation overlay

The Translate Text demo shows how to offer a translation using the system UI. The code example below indicates what happens when a person clicks the Translate button in that demo. First, the sample defines a variable to control when the system UI appears using a `State` property wrapper. Then it defines a variable representing the text to translate. Finally, it passes the `State` property wrapper that controls when the system UI displays along with the text to translate to the `translationPresentation(isPresented:text:attachmentAnchor:arrowEdge:replacementAction:)` function that attaches to the view containing the text to translate.

```
struct ViewTranslationView: View {
    // Define the condition to display the translation UI.
    @State private var showTranslation = false

    // Define the text you want to translate.
    var originalText = "Hallo, Welt!"

    var body: some View {
        VStack {
            Text(verbatim: originalText)
            Button("Translate") {
                showTranslation.toggle()
            }
        }
        // Offer a system UI translation.
        .translationPresentation(isPresented: $showTranslation,
                                 text: originalText)
        .navigationTitle("Translate")
    }
}
```

The Replace Text demo also uses the system UI, and replaces the original text with the translated text by adding the `replacementAction` trailing closure to the function. This closure runs after the translation occurs when a person clicks the Translate button. The sample then replaces the input text with the translated text.

```
.translationPresentation(isPresented: $showTranslation, text: originalText) { transl
    originalText = translatedText
}
```

Offer a custom translation

For more control over the translation experience, you can create your own in-app translation experiences using the `TranslationSession` object, along with its associated translation functions.

The Single String demo shows how to translate a single string of text and update a view. Its code example below defines a `State` property wrapper for the `TranslationSession.Configuration` object in the view offering the translation, and passes it to a `translationTask(_:action:)` function. Then it uses the `session` instance the task returns to call `translate(_:)` and passes in the string of text to translate. Finally, it updates the view using the `targetText` property from the response.

```swift
struct SingleStringView: View {
    @State private var sourceText = "Hallo, Welt!"
    @State private var targetText = ""

    // Define a configuration.
    @State private var configuration: TranslationSession.Configuration?

    var body: some View {
        VStack {
            TextField("Enter text to translate", text: $sourceText)
                .textFieldStyle(.roundedBorder)
            Button("Translate") {
                triggerTranslation()
            }
            Text(verbatim: targetText)
        }
        // Pass the configuration to the task.
        .translationTask(configuration) { session in
            do {
                // Use the session the task provides to translate the text.
                let response = try await session.translate(sourceText)

                // Update the view with the translated result.
                targetText = response.targetText
            } catch {
                // Handle any errors.
            }
        }
        .padding()
        .navigationTitle("Single string")
    }

    private func triggerTranslation() {
        guard configuration == nil else {
            configuration?.invalidate()
            return
        }

        // Let the framework automatically determine the language pairing.
        configuration = .init()
    }
```

This pattern of defining a configuration as a `State` property wrapper and invalidating its state is recommended for triggering new translations. Storing `TranslationSession.Configuration` in a model object also works.

Translate batches of strings

To translate multiple strings between the same languages, it's most efficient to use one of the batch translation functions of `TranslationSession`. You can translate a batch of strings and get the results all at once, or you can receive them incrementally as they arrive.

To keep your UI clean and uncluttered, you can also do this work in a view model, as the Batch All at Once demo shows. Its code example defines a `State` property wrapper for a `TranslationSession.Configuration` object in the view offering the translation, and attaches the `translationTask(_:action:)` function to the view containing the text to translate. Using the `session` instance from the task, it calls the `translations(from:)` function in the view model. The view model maps all the strings to translate in an array of type `TranslationSession.Request` and sets the `sourceText` property of the request to the string to translate. Then the sample calls `translations(from:)` on the session object and passes in the array of requests to translate. Finally, it maps each response object that returns to the corresponding index in the original array of text.

View   View model

```
struct BatchOfStringsView: View {
    @Environment(ViewModel.self) var viewModel

    // Define a configuration.
    @State private var configuration: TranslationSession.Configuration?

    var body: some View {
        VStack {
            ForEach(viewModel.foodItems, id: \.self) { item in
                Text(item)
                    .padding()
            }
            HStack {
                Button("Translate") {
                    triggerTranslation()
                }
                Button("Reset") {
                    viewModel.reset()
                }
            }
        }
        .translationTask(configuration) { session in
            // Use the session the task provides to translate the text.
            await viewModel.translateAllAtOnce(using: session)
        }
        .onAppear {
            viewModel.reset()
        }
        .padding()
        .navigationTitle("Batch all at once")
    }

    private func triggerTranslation() {
        if configuration == nil {
            // Set the language pairing.
            configuration = .init(source: Locale.Language(identifier: "de"),
                                  target: Locale.Language(identifier: "en"))
        } else {
            // Invalidate the previous configuration.
            configuration?.invalidate()
        }
    }
```

When translating a group of strings using the `translations(from:)` function, the responses return in the same order you send them.

Display the results as they arrive

If you want to translate multiple strings of text and display the results as they arrive, use the `translate(batch:)` function.

The Batch as a Sequence demo is the same as the Batch All at Once demo except for the way it matches translation requests to responses.

In this process, the responses can return in a different order than you send them. So the Batch as a Sequence demo tracks which request corresponds with which response by assigning a `clientIdentifier` to each sent request — in this case, the index of the item to translate in the array.

The client identifier returns with the same value in the response. When the response returns, the sample uses the client identifier to associate the request with the response.

```swift
extension ViewModel {
    func translateSequence(using session: TranslationSession) async {
        Task { @MainActor in
            let requests: [TranslationSession.Request] = foodItems.enumerated().map
                // Assign each request a client identifier.
                    .init(sourceText: string, clientIdentifier: "\(index)")
            }

            do {
                for try await response in session.translate(batch: requests) {
                    // Use the returned client identifier (the index) to map the rec
                    guard let index = Int(response.clientIdentifier ?? "") else { cc
                    foodItems[index] = response.targetText
                }
            } catch {
                // Handle any errors.
            }
        }
    }
}
```

Check for language availability

The Language Availability demo shows how to check whether the framework supports a specific language pairing before offering it to people. A person selects a source and a target language from a picker, and the sample passes that language pairing to a function in the view model that checks whether the framework supports that pairing. The view then updates the display when the model sets the `isTranslationSupported` variable.

```swift
extension ViewModel {
    func checkLanguageSupport(from source: Locale.Language, to target: Locale.Langua
        let availability = LanguageAvailability()
        let status = await availability.status(from: source, to: target)

        switch status {
        case .installed, .supported:
            isTranslationSupported = true
        case .unsupported:
            isTranslationSupported = false
        @unknown default:
            print("Not supported")
        }
    }
}
```

> **Note**
>
> The framework doesn’t support translation between the same language, such as British English (en-GB) and U.S. English (en-US).

Prepare for translation

If you know which languages a translation needs, you can download the language models before offering the translation.

The Prepare for Translation demo shows how to initiate the download of a language pairing by creating a `TranslationSession.Configuration` instance with the source and target language to download. The sample then calls `prepareTranslation()` on the session the translation task returns. If the languages aren’t already downloaded, a view appears that requests permission to download them.

```
struct PrepareTranslationView: View {

    // Define the pairing of languages you want to download.
    @State private var configuration = TranslationSession.Configuration(
        source: Locale.Language(identifier: "pt_BR"),
        target: Locale.Language(identifier: "ko_KR")
    )

    @State private var buttonTapped = false

    var body: some View {
        VStack(spacing: 20) {
            Text("Tap the button to start downloading languages before offering a tr
            Button("Prepare") {
                configuration.invalidate()
                buttonTapped = true
            }
        }
        .translationTask(configuration) { session in
```

```
                if buttonTapped {
                    do {
                        // Display a sheet asking the user's permission
                        // to start downloading the language pairing.
                        try await session.prepareTranslation()
                    } catch {
                        // Handle any errors.
                    }
                }
            }
            .padding()
            .navigationTitle("Prepare translation")
        }
    }
```

For testing, you can delete locally downloaded models in macOS by choosing System Settings > General > Language & Region > Translation Languages. In iOS, you can manage your downloaded languages by choosing Settings > Apps > Translate > Downloaded Languages if you have the system Translate app installed locally on your device.

See Also

Essentials

`nonisolated func translationPresentation(isPresented: Binding<Bool>, text: String, attachmentAnchor: PopoverAttachmentAnchor = .rect(.bounds), arrowEdge: Edge = .top, replacementAction: ((String) -> Void)? = nil) -> some View`

Presents a translation popover when a given condition is true.

`nonisolated func translationTask(_ configuration: TranslationSession.Configuration?, action: @escaping (TranslationSession) async -> Void) -> some View`

Adds a task to perform before this view appears or when the translation configuration changes.

`nonisolated func translationTask(source: Locale.Language? = nil, target: Locale.Language? = nil, action: @escaping (TranslationSession) async -> Void) -> some View`

Adds a task to perform before this view appears or when the specified source or target languages change.

```
class TranslationSession
```

A class that performs translations between a pair of languages.