Article

Support languages and locales with Foundation Models

Generate content in the language people prefer when they interact with your app.

Overview

The on-device system language model is multilingual, which means the same model understands and generates text in any language that Apple Intelligence supports. The model supports using different languages for prompts, instructions, and the output that the model produces.

When you enhance your app with multilingual support, generate content in the language people prefer to use when they interact with your app by:

- Prompting the model with the language you prefer.
- Including the target language for your app in the instructions you provide the model.
- Determining the language or languages a person wants to use when they interact with your app.
- Gracefully handling languages that Apple Intelligence doesn't support.

For more information about the languages and locales that Apple Intelligence supports, see the "Supported languages" section in How to get Apple Intelligence.

Prompt the model in the language you prefer

Write your app's built-in prompts in the language with which you normally write code, if Apple Intelligence supports that language. Translate your prompts into a supported language if your preferred language isn't supported. In the code below, *all* inputs need to be in supported language for the model to understand, including all `Generable` types and descriptions:

```swift
@Generable(description: "Basic profile information about a cat")
struct CatProfile {
    var name: String

    @Guide(description: "The age of the cat", .range(0...20))
    var age: Int

    @Guide(description: "One sentence about this cat's personality")
    var profile: String
}

#Playground {
    let response = try await LanguageModelSession().respond(
        to: "Generate a rescue cat",
        generating: CatProfile.self
    )
}
```

Because the framework treats `Generable` types as model inputs, the names of properties like `age` or `profile` are just as important as the `@Guide` descriptions for helping the model understand your request.

Check a person's language settings for your app

People can use the Settings app on their device to configure the language they prefer to use on a per-app basis, which might differ from their default language. If your app supports a language that Apple Intelligence doesn't, you need to verify that the current language setting of your app is supported before you call the model. Keep in mind that language support improves over time in newer model and OS versions. Thus, someone using your app with an older OS may not have the latest language support.

Before you call the model, run `supportsLocale(_:)` to verify the support for a locale. By default, the method uses `current`, which takes into account a person's current language and app-specific settings. This method returns true if the model supports this locale, or if this locale is considered similar enough to a supported locale, such as en-AU and en-NZ:

```swift
if SystemLanguageModel.default.supportsLocale() {
    // Language is supported.
}
```

For advanced use cases where you need full language support details, use `supportedLanguages` to retrieve a list of languages supported by the on-device model.

Handle an unsupported language or locale errors

When you call `respond(to:options:)` on a `LanguageModelSession`, the Foundation Models framework checks the language or languages of the input prompt text, and whether your prompt asks the model to respond in any specific language or languages. If the model detects a language it doesn't support, the session throws `LanguageModelSession.GenerationError.unsupportedLanguageOrLocale(_:)`. Handle the error by communicating to the person using your app that a language in their request is unsupported.

If your app supports languages or locales that Apple Intelligence doesn't, help people that use your app by:

- Explaining that their language isn't supported by Apple Intelligence in your app.
- Disabling your Foundation Models framework feature.
- Providing an alternative app experience, if possible.

Important

Guardrails for model input and output safety are only for supported languages and locales. If a prompt contains sensitive content in an unsupported language, which typically is a short phrase mixed-in with text in a supported language, it might not throw a `LanguageModelSession.GenerationError.unsupportedLanguageOrLocale(_:)` error. If unsupported-language detection fails, the guardrails may also fail to flag that short, unsupported content. For more on guardrails, see Improving the safety of generative model output.

Use Instructions to set the locale and language

For locales other than United States English, you can improve response quality by telling the model which locale to use by detailing a set of `Instructions`. Start with the *exact* phrase in English. This special phrase comes from the model's training, and reduces the possibility of hallucinations in multilingual situations:

```
func localeInstructions(for locale: Locale = Locale.current) -> String {
    if Locale.Language(identifier: "en_US").isEquivalent(to: locale.language) {
        // Skip the locale phrase for U.S. English.
        return ""
    } else {
        // Specify the person's locale with the exact phrase format.
        return "The person's locale is \(locale.identifier)."
    }
```

After you set the locale in `Instructions`, you may need to explicitly set the model output language. By default, the model responds in the language or languages of its inputs. If your app supports multiple languages, prompts that you write and inputs from people using your app might be in different languages. For example, if you write your built-in prompts in Spanish, but someone using your app writes inputs in Dutch, the model may respond in either or both languages.

Use `Instructions` to explicity tell the model which language or languages with witch it needs to respond. You can phrase this request in different ways, for example: "You MUST respond in Italian" or "You MUST respond in Italian and be mindful of Italian spelling, vocabulary, entities, and other cultural contexts of Italy." These instructions can be in the language you prefer.

```
let session = LanguageModelSession(
    instructions: "You MUST respond in U.S. English."
)
let prompt = // A prompt that contains Spanish and Italian.
let response = try await session.respond(to: prompt)
```

Finally, thoroughly test your instructions to ensure the model is responding in the way you expect. If the model isn't following your instructions, try capitalized words like "MUST" or "ALWAYS" to strengthen your instructions.

See Also

Essentials

Generating content and performing tasks with Foundation Models

Enhance the experience in your app by prompting an on-device large language model.

Improving the safety of generative model output

Create generative experiences that appropriately handle sensitive inputs and respect people.

{} Adding intelligent app features with generative models

Build robust apps with guided generation and tool calling by adopting the Foundation Models framework.

`class SystemLanguageModel`

An on-device large language model capable of text generation tasks.

`struct UseCase`

A type that represents the use case for prompting.