Article

Improving interactions between Siri and your messaging app

Donate app-specific content, use Siri's contact suggestions, and adopt the latest platform features to create a more consistent messaging experience.

Overview

SiriKit provides a number of supplementary APIs that are specific to messaging apps. Adopt these APIs and provide the required information so your messaging app can:

- influence the choices Siri makes when routing a message-related request
- provide more accurate recipient options when sending messages with Siri
- enable someone to unsend or edit a message sent with Siri
- provide richer information about message attachments for Siri to announce or display

> **Tip**
>
> In addition to messaging apps, the content of the first two sections is applicable to apps using SiriKit's VoIP-related intents. For example, `INStartCallIntent`.

For information about how Apple protects people's privacy when your messaging or VoIP app shares information with the system, see the Siri section of Privacy - Features.

Donate contact vocabulary and app interactions to Siri

Siri integrates with the Contacts app and uses the app's contact database to suggest message recipients. If your app manages its own database of contacts, independent of the Contacts app,

donate the names of those contacts using `INVocabulary` to enable Siri to locate them when handling requests. If your messaging app supports group chats, use the same APIs to donate the in-app names of those group chats.

To donate the required vocabulary, create instances of `INSpeakableString`. Set `spokenPhrase` to the name of the contact or group chat, and `vocabularyIdenfitier` to the value of the contact's `customIdentifier` property — the value your app uses to uniquely identify that object.

```
let contacts = NSOrderedSet(array: [
    INSpeakableString(vocabularyIdentifier: "73f16d14-0207-4668-a1c1-1bd90aaea93f",
                      spokenPhrase: "Maria Ruiz",
                      pronunciationHint: nil),
    INSpeakableString(vocabularyIdentifier: "c28da32e-31a0-4ee9-b93c-9619ad254e10",
                      spokenPhrase: "Juan Chavez",
                      pronunciationHint: nil)
])

// Use INVocabularyStringType.contactGroupName for group chats.
INVocabulary.shared().setVocabulary(contacts, of: .contactName)
```

Important

Order the collection of speakable strings by importance. For more information, see `setVocabulary(_:of:)`.

When someone performs an action in your messaging app and the app includes support for the corresponding intent, such as sending or replying to a message, create and donate an instance of `INInteraction` to make Siri aware of the action. If your app uses the system's Contacts database, reduce ambiguity by setting the `contactIdentifier` property for each of the interaction's contacts to their corresponding database identifier; otherwise, set their `customIdentifier` property to the unique value your app uses to identify them.

Choose recipients from the matches Siri suggests

When Siri asks your intent handler to resolve a message's recipients, it provides a collection of `INPerson` objects — one for each recipient. Using data from the Contacts app (for example, work or family relationships), Siri populates the `siriMatches` property on each instance of `INPerson` with zero or more proposed contact matches, enabling your app to provide more accurate recipient options to the person sending the message.

Use the following guidance to determine which `INPersonResolutionResult` method to invoke based on the matches Siri provides:

- If the `siriMatches` array contains a single match and your app successfully identifies that person by their `contactIdentifier` or `customIdentifier`, use the `success(with:)` method to resolve the recipient.
- If the array contains multiple people and those people each have distinct `contactIdentifier` or `customIdentifier` values, disregard those your app can't identify and then respond according to the number of remaining matches. For example, if there are still two or more matches, use the `disambiguation(with:)` method to ask the person interacting with Siri to choose.
- If the array contains multiple people and they all have the same `contactIdentifier`, disambiguate using `personHandle` instead. If your app doesn't use handles, simply pass any match to the `success(with:)` method.
- If the array is empty or the property is `nil`, search your app's contacts for people matching those in the `recipients` property of `INSendMessageIntent` instead. Use the `INPersonResolutionResult` method that best describes the outcome of that search.

If your messaging app allows people to attach a custom image to each of their contacts, use the `image` property on `INPerson` to provide that image when returning an instance of `INPersonResolutionResult`.

Enable people to unsend or edit sent messages

In iOS 16 and later, after a person sends a message from the Messages app or Siri, within a short duration after sending, they can unsend or edit that message. To add the same functionality to your messaging app, perform the following:

1. Whenever you create an instance of `INSendMessageIntentResponse`, populate the `sentMessages` property with one or more `INMessage` objects. Set each object's `identifier` property to the value your app uses to uniquely identify the sent message. The system passes those values to your app when the app needs to take action on behalf of the sender.
2. To support message unsending, create an object that conforms to the `INUnsendMessagesIntentHandling` protocol and implement the required `handle(intent:completion:)` method. On receipt of an unsend request, Siri invokes this method and provides an instance of `INUnsendMessagesIntent`. Use the values in the object's `messageIdentifiers` property to identify those messages in your app and take the action necessary to unsend them.
3. Similarly, to support message editing, create an object that conforms to the `INEditMessageIntentHandling` protocol and implement the required `handle(intent:completion:)` method. On receipt of an edit request, Siri invokes this method and provides an instance of `INEditMessageIntent`. The object contains a single message identifier for you to identify

the corresponding message in your app. If you specify multiple `INMessage` objects when originally sending the message, the system returns the last one with a message type of `INMessageType.text`.

The following shows an example implementation of message unsending:

```
class UnsendMessagesIntentHandler: NSObject, INUnsendMessagesIntentHandling {
    func handle(intent: INUnsendMessagesIntent) async -> INUnsendMessagesIntentRespo
        guard let identifiers = intent.messageIdentifiers, identifiers.count > 0 els
            return INUnsendMessagesIntentResponse(code: .failure, userActivity: nil)
        }

        messageIdentifiers.forEach { MessageDatabase.unsendMessage($0) }
        return INUnsendMessagesIntentResponse(code: .success, userActivity: nil)
    }
}
```

SiriKit provides additional response codes specific to message unsending and editing. For more information, see `INUnsendMessagesIntentResponseCode` and `INEditMessageIntentResponseCode`.

Return attachment metadata in Siri search results

When someone asks Siri for information about their sent or received messages, Siri creates an instance of `INSearchForMessagesIntent` and requests your app to fulfill that intent. If the messages in your app's results include attachments, use the appropriate attachment-related initializers on `INMessage` to provide additional information for Siri to announce or display. For example, you can include an array of `INFile` instances that describe each file (including their content type), or if a single message has several attachments, provide the count of attachments instead.

For more information, see the following:

Action	Initializer
Provide one or more message attachments	`init(identifier:conversationIdentifier:content:dateSent:sender:recipients:groupName:messageType:serviceName:attachmentFiles:)`
Provide information about a URL in a message	`init(identifier:conversationIdentifier:content:dateSent:sender:recipients:groupName:serviceName:linkMetadata:)`

Action	Initializer
Provide the total number of attachments	`init(identifier:conversationIdentifier:content:dateSent:sender:recipients:groupName:serviceName:messageType:numberOfAttachments:)`

See Also

Articles

- Adding User Interactivity with Siri Shortcuts and the Shortcuts App

  Add custom intents and parameters to help users interact more quickly and effectively with Siri and the Shortcuts app.

- Defining Relevant Shortcuts for the Siri Watch Face

  Inform Siri when your app's shortcuts may be useful to the user.

- Deleting Donated Shortcuts

  Remove your donations from Siri.

- Dispatching intents to handlers

  Provide SiriKit with an intent handler capable of handling a specific intent.

- Improving Siri Media Interactions and App Selection

  Fine-tune voice controls and improve Siri Suggestions by sharing app capabilities, customized names, and listening habits with the system.

- Registering Custom Vocabulary with SiriKit

  Register your app's custom terminology, and provide sample phrases for how to use your app with Siri.

- Confirming the Details of an Intent

  Perform final validation of the intent parameters and verify that your services are ready to fulfill the intent.

- Handling an Intent

  Fulfill the intent and provide feedback to SiriKit about what you did.

- Resolving the Parameters of an Intent

Validate the parameters of an intent and make sure that you have the information you need to continue.

Generating a List of Ride Options

Generate ride options for Maps to display to the user.

Handling the Ride-Booking Intents

Support the different intent-handling sequences for booking rides with Shortcuts or Maps.

Donating Reservations

Inform Siri of reservations made from your app.

Specifying Synonyms for Your App Name

Provide alternative names for your app that are more familiar or easier for users to speak.

Intent Phrases

The keys that you include in your global vocabulary file to show how users engage your app from Siri.

Localizing Your Vocabulary for Chinese Dialects

Apply emphasis markers to your pronunciation tips to assist Siri with Chinese dialects.