Sample Code

Handling Communication Notifications and Focus Status Updates

Create a richer calling and messaging experience in your app by implementing communication notifications and Focus status updates.

Download

iOS 14.0+ | iPadOS 14.0+ | Xcode 13.0+

Overview

Communication notifications have a distinct user experience that features prominent avatars and group names, as well as unique breakthrough behaviors. Implement these notifications for incoming calls and messages to provide additional context to the user.

> **Note**
>
> This sample code project is associated with WWDC21 session 10091: Send communication and Time Sensitive notifications.

Configure the Sample Code Project

This sample runs on physical devices in order to meet requirements for push notifications and the Notification Service Extension. The Notification Service Extension modifies a notification's content before it's displayed on the user's device. To learn more about Notification Service Extensions, see Modifying Content in Newly Delivered Notifications.

Begin by configuring the iOS target to include your development team and a bundle identifier. The bundle identifier must support the following capabilities:

- Push notifications
- Communication notifications
- Time Sensitive notifications
- SiriKit

To support communication notifications, the app's `Info.plist` file must contain a top-level entry with the key NSUserActivityTypes and type `Array`. This array should contain `INStartCallIntent` if the app supports calling functionality, and `INSendMessageIntent` if it supports messaging.

Accessing a user's Focus status requires their authorization. To let the user know how the app uses their Focus status, include a usage description string. Provide this usage description string as a value for the `Privacy - Focus Status Usage Description` key in the app's `Info.plist` file.

Donate Intent Interactions

Donate an intent interaction for each communication that takes place in the app. The app should donate both outgoing and incoming communication interactions. Donate incoming interactions to update communication notifications and enable their unique user experience and breakthrough behavior. Donate outgoing interactions when the current user sends messages or initiates calls. This allows the system to make better suggestions and provide communication notification functionality.

The sample initializes interactions in `CommunicationMapper.ineraction(communicationInformation:)` and donates them in the `suggest(communicationInformation:completion:)` method. The provided communication information determines the interaction's direction.

```
    /// Outgoing messages and calls suggest people involved for Focus breakthrough.
    static func suggest(communicationInformation: CommunicationInformation,
                        completion: @escaping (Result<INInteraction, Error>) -> Voic
        do {
            // Create an INInteraction.
            let interaction = try CommunicationMapper.interaction(communicationInfor
            // Donate INInteraction to the system.
            interaction.donate { [completion] error in
                DispatchQueue.global(qos: .userInitiated).async {
                    if let error = error {
                        completion(.failure(error))
                    } else {
                        completion(.success(interaction))
```

```
                }
            }
        }
    } catch let error {
        // Catch CommunicationMapper errors.
        completion(.failure(error))
    }
}
```

Update Notification Content

To display a communication notification, your app should update the notification's content. This allows the notification to break through scheduled notification summaries. It's also possible for communication notifications to break through an enabled Focus when the Allowed People list contains the sender. Use the same intent object that initialized the `incoming` interaction. Wait for the interaction donation to complete before updating notification content.

The sample updates notification content in the Notification Service Extension by calling `CommunicationInteractor.update(notificationContent:communicationInformation:completion:)`.

```
/// Update incoming notifications with a message or call information to allow th
/// - Display an avatar, if present.
/// - Check if sender is allowed to break through.
/// - Update notification title (sender's name) and subtitle (group information)
@available(iOS 15.0, watchOS 8.0, macOS 12.0, *)
static func update(notificationContent: UNNotificationContent,
                   communicationInformation: CommunicationInformation,
                   completion: @escaping (Result<UNNotificationContent, Error>)
    suggest(communicationInformation: communicationInformation) { [notificationC
        switch result {
            case .failure(let error):
                completion(.failure(error))
            case .success(let interaction):
                guard let notificationContentProvider = interaction.intent as? U
                    completion(.failure(CommunicationInteractorError.unexpectedI
                    return
                }
                do {
                    let updatedContent = try notificationContent.updating(from:
                    completion(.success(updatedContent))
                } catch let error {
```

```
                    completion(.failure(error))
                }
            }
        }
    }
```

Request Focus Status

When a user has enabled a Focus, it may be useful to display that status to other users of an app. This informs other users when someone silences their notifications and may not see communications right away.

Accessing the user's Focus status requires explicit user authorization. To request authorization, use `INFocusStatusCenter.default.requestAuthorization(completionHandler:)` and parse the result. Someone can choose to authorize the app to access their Focus status, deny the app access, or choose neither.

```
    /// Requests FocusStatusCenter authorization.
    /// Parameter completion: Result contains AuthorizationStatus or error.
    @available(iOS 15.0, watchOS 8.0, macOS 12.0, *)
    static func requestFocusStatusAuthorization(completion: @escaping (Result<Autho
        INFocusStatusCenter.default.requestAuthorization { status in
            switch status {
                case .denied:
                    completion(.success(.denied))
                case .authorized:
                    completion(.success(.authorized))
                case .notDetermined:
                    completion(.success(.notDetermined))
                case .restricted:
                    completion(.success(.restricted))
                @unknown default:
                    completion(.success(.notSupported))
            }
        }
    }
```

Once authorized, an app can request the user's current Focus status. The perspective of the app is important; the user doesn't appear focused to an app if an enabled Focus allows notifications from that app.

The sample accesses the current Focus status in the `requestFocusStatus(completion:)` method. Unauthorized apps don't receive a Focus status value. Ensure the app handles the `nil`

Focus status case.

```
    /// Requests current focus status.
    /// Requires UserNotifications and FocusStatus to be authorized and Communicatio
    /// Parameter completion: Result contains FocusStatus isFocused Bool, which will
    /// isn't in its Allowed Apps list.
    @available(iOS 15.0, watchOS 8.0, macOS 12.0, *)
    static func requestFocusStatus(completion: @escaping (Result<Bool, Error>) -> Vo
        guard let isFocused = INFocusStatusCenter.default.focusStatus.isFocused else
            completion(.failure(CommunicationInteractorError.focusStatusNotAvailable
            return
        }
        completion(.success(isFocused))
    }
```

Observe Focus status changes from an Intents app extension. This extension launches in the background to handle interactions between your app and SiriKit. To learn more about Intents app extensions, see Creating an Intents App Extension. Ensure your Intents app extension target includes support for INShareFocusStatusIntent. The Intents app extension target's General tab in the project file contains the list of supported intents. Include the class names of all supported intents in the Supported Intents section.

The Intents app extension in the sample handles incoming INShareFocusStatusIntent objects. When doing so, the sample uses the intent itself to access the `isFocused` bool directly instead of using the default `INFocusStatusCenter`.

```
/**
     For this Intent to be handled, the following requirements must be met:
     FocusStatusCenter authorized for parent app (target).
     UserNotifications authorized for parent app (target).
     Communication Notifications capability (entitlement) added to the parent app (t
     */
    func handle(intent: INShareFocusStatusIntent, completion: @escaping (INShareFocu
        let response = INShareFocusStatusIntentResponse(code: .success, userActivity
        if let isFocused = intent.focusStatus?.isFocused {
            // Send isFocused value to servers or AppGroup.
            print("Is user focused: \(isFocused)")
        }
        completion(response)
    }
```

See Also

Sample code

{} Implementing Alert Push Notifications

Add visible alert notifications to your app by using the UserNotifications framework.

{} Implementing Background Push Notifications

Add background notifications to your app by using the UserNotifications framework.