Sample Code

IceCreamBuilder: Building an iMessage Extension

Allow users to collaborate on the design of ice cream sundae stickers.

Download

iOS 12.0+ | iPadOS 12.0+ | Xcode 15.0+

Overview

In this sample project, two or more users work together to build ice cream sundae stickers inside an iMessage Extension. Each completed sticker consists of a base, scoops of ice cream, and a topping. The process of building the sundae is collaborative and each step can be completed by any user who has the app installed. The participants can then send the stickers as images or attach them to any sent message.

The Xcode project consists of two targets:

- An iOS application that in this sample has no functionality, but would usually contain your main application.
- An iMessage Extension target that contains all logic for building and displaying the stickers.

The iMessage Extension target uses a `MSMessagesAppViewController` subclass that presents child view controllers to either show the history of previously created ice cream sundaes or the collaboration interface to build a new sticker.

When the iMessage Extension target is first initialized, the `willBecomeActive(with:)` method of the `MessagesViewController` configures and displays the appropriate child view controller.

Use Presentation Styles

iMessage Extensions can be required to display in several different presentation styles. This sample code demonstrates the `compact` and `expanded` styles.

When the Ice Cream Builder app icon is tapped from the Messages app drawer, the extension initially opens in `compact` mode where the initial view controller replaces the system keyboard. In this case, the `MessagesViewController` adds the `IceCreamsViewController` as a child view controller to display the history of previously created ice cream sundaes, along with an "Add" button to start creating a new one.

The UI for creating a new ice cream sundae sticker benefits from a larger view controller so when the user taps the "Add" button the app requests the `expanded` presentation style by calling `requestPresentationStyle(_:)`. The system then calls `willTransition(to:)` on the `MessagesViewController` where the sample code instantiates a view controller to either continue building the sundae, or to display the completed sundae.

```
let controller: UIViewController
if presentationStyle == .compact {
    // Show a list of previously created ice creams.
    controller = instantiateIceCreamsController()
} else {
     // Parse an `IceCream` from the conversation's `selectedMessage` or create a ne
    let iceCream = IceCream(message: conversation.selectedMessage) ?? IceCream()

    // Show either the in process construction process or the completed ice cream.
    if iceCream.isComplete {
        controller = instantiateCompletedIceCreamController(with: iceCream)
    } else {
        controller = instantiateBuildIceCreamController(with: iceCream)
    }
}
```

Attach Stickers to Messages

The `IceCreamsViewController` uses a collection view to display all previously created ice cream sundae stickers. Each cell uses a `MSStickerView` object that displays a `MSSticker` and also implements the standard functionality of dragging and dropping stickers to attach them to messages in the transcript, or sending the sticker as an image.

Pass Data with Messages

All data passed between users in a conversation needs to be encoded in a URL; you should use `URLComponents` to build and parse these URLs.

```
var queryItems: [URLQueryItem] {
    var items = [URLQueryItem]()

    if let part = base {
        items.append(part.queryItem)
    }
    if let part = scoops {
        items.append(part.queryItem)
    }
    if let part = topping {
        items.append(part.queryItem)
    }

    return items
}
```

The current state of the ice cream sundae also needs to be represented visually in the Messages transcript using an instance of `MSMessageTemplateLayout`. In this sample, the image property contains a rendering of the current state of the ice cream sundae.

Once the message is ready to be sent, create an `MSMessage` encapsulating the message data and layout.

```
fileprivate func composeMessage(with iceCream: IceCream, caption: String, session: M
    var components = URLComponents()
    components.queryItems = iceCream.queryItems

    let layout = MSMessageTemplateLayout()
    layout.image = iceCream.renderSticker(opaque: true)
    layout.caption = caption

    let message = MSMessage(session: session ?? MSSession())
    message.url = components.url!
    message.layout = layout

    return message
}
```

Once a message has been composed, it can be sent by accessing the `activeConversation` property of the `MSMessagesAppViewController`.

```
// Add the message to the conversation.
conversation.insert(message) { error in
    if let error = error {
        print(error)
    }
}
```

The `insert(_:completionHandler:)` method used in this code will place the composed message into the conversation field, but will *not* send it immediately, requiring the user to still tap the send button. If you prefer to send the message immediately then use the `send(_:completionHandler:)` method instead. This method is available on iOS 11 and above.

See Also

Custom iMessage app interface

{} Creating a Sticker App with a Custom Layout

Expand on the Messages sticker app template to create an app with a customized user interface.

`class MSMessagesAppViewController`

The principal view controller for iMessage apps.

`protocol MSMessagesAppTranscriptPresentation`

A protocol that provides support for displaying live messages in the transcript of the Messages app.

`enum MSMessagesAppPresentationStyle`

Presentation styles that describe your iMessage app's appearance.