

# Alchemer Mobile iOS Integration Reference

## System Requirements

Alchemer Mobile (ApptentiveKit) is written entirely in Swift, but features that are not directly available in Objective-C (due to the use of value types or generics) have Objective-C-friendly wrappers.

The current version of Alchemer Mobile (ApptentiveKit) is primarily intended for use with UIKit-based apps on iOS and iPadOS. Apps using SwiftUI may be able to make use of Alchemer Mobile (ApptentiveKit), but currently can't provide support for this use case.

- Minimum Deployment Target: iOS 13.0
- Minimum Xcode Version: 13.0
- No external dependencies required

## SDK Size

The SDK is estimated to add approximately 2.3MB to the size of your app.

## Supported Languages

We have translated all hard-coded strings in our SDK into the following languages. The content of all Interactions comes from our server, and you may translate the text for each Interaction by visiting the [Translations Page](#).

Please note that you must also specify the target language in your Xcode project by selecting the project, clicking Info, and then, under Localizations, click the Add button for the desired language.

Locale Qualifier	Language Name
None	English
ar	Arabic
el	Greek
da	Danish
de	German
es	Spanish
fr	French

Locale Qualifier	Language Name
fr-CA	French Canadian
it	Italian
ja	Japanese
ko	Korean
nl	Dutch
pl	Polish
pt-BR	Brazilian Portuguese
ru	Russian
sv	Swedish
tr	Turkish
zh-Hant	Chinese (Traditional)
zh-Hans	Chinese (Simplified)

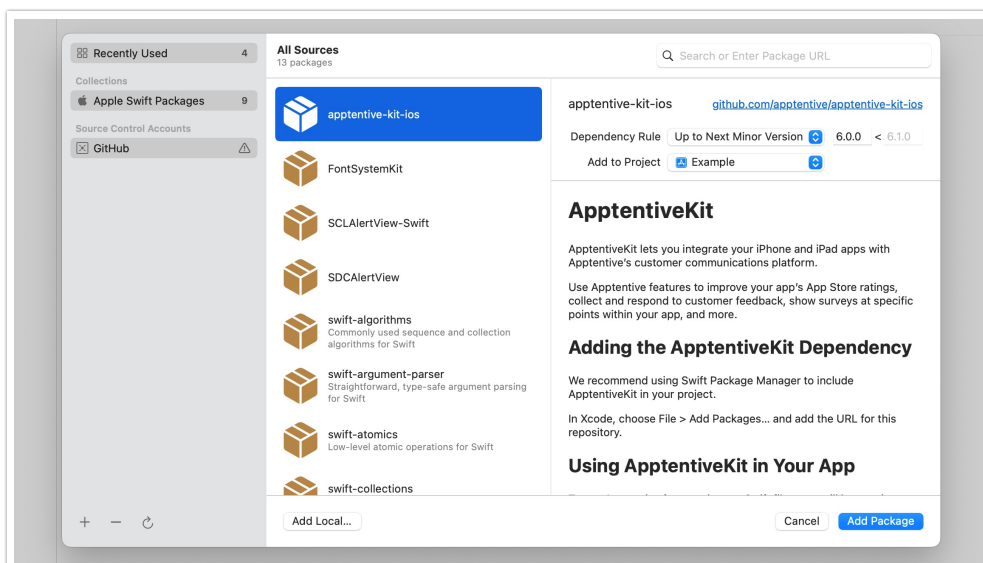
## Adding Alchemer Mobile (ApptentiveKit)

There are several ways to add the Alchemer Mobile (ApptentiveKit) dependency to your app. We recommend using Swift Package Manager.

### Swift Package Manager

In Xcode, choose Add Packages... from the File menu, and enter

`https://github.com/apptentive/apptentive-kit-ios` in the search field.



Select the `apptentive-kit-ios` package from the list. We recommend using the “Up to Next Minor Version” dependency rule to make it easy to update to the latest fully-compatible release of the SDK. When you are finished, click Add Package.

## CocoaPods

If you are using CocoaPods to manage your project’s dependencies, add the following line for your app target in your Podfile:

```
pod 'ApptentiveKit', '~>6.5'
```

We recommend the above version specifier to make it easy to update to the latest fully-compatible release of the SDK.

If you were previously using the `apptentive-ios` pod, be sure to remove it.

## Carthage

If you are using Carthage to manage your project’s dependency,

1. Add `github "apptentive/apptentive-kit-ios" ~> 6.5` to your Cartfile. We recommend this version specifier to make it easy to update to the latest fully-compatible release.
2. Run `carthage update --use-xcframeworks` .
3. On your application targets’ *General* settings tab, in the *Frameworks, Libraries, and Embedded Content* section, drag and drop the `Apptentive.xcframework` folder from the `Carthage/Build` folder on disk.

## XCFramework

You can download the latest release as a pre-built framework from our [Github releases page](#) and drag it into your project. This requires that you manually download any new releases of Alchemer Mobile (ApptentiveKit) to keep your project up to date.

## Sub-Project

You can also clone the [ApptentiveKit repository](#) and drag the `ApptentiveKit.xcodeproj` file into your project in Xcode. This requires that you manually pull any new code from the Alchemer Mobile (ApptentiveKit) repository to keep your project up to date.

# Updating Alchemer Mobile (ApptentiveKit)

Please refer to our [Release Notes](#) for information on the latest version. We recommend using our latest version whenever possible.

Depending on how you originally configured your dependency, you may have to edit the dependency before your dependency manager will allow the update.

## Swift Package Manager

In Xcode, select your project in the Project Navigator, select the project in the editor sidebar, and select the Package Dependencies tab.

If your update settings will allow an update, you can right-click the Alchemer Mobile (ApptentiveKit) package and choose **Update Package**.

If your update settings will *not* allow an update, double-click the Alchemer Mobile (ApptentiveKit) package and set the minimum version to the version you'd like to update to, and consider changing the dropdown to **Up to Next Minor**, which will allow you to more easily receive updates for bug fixes and non-breaking changes.

## CocoaPods

If your Podfile is configured to allow an update, you can run `pod update 'ApptentiveKit'` in the same directory as your Podfile.

If your Podfile is *not* configured to allow an update, edit the line for the Alchemer Mobile (ApptentiveKit) pod with the version you would like to update to. We recommend a setting like `'~>6.5'` which will allow you to more easily receive updates for bug fixes and non-breaking changes. You can then run `pod update 'ApptentiveKit'` as described above.

## Carthage

If your Cartfile is configured to allow an update, run `carthage update ApptentiveKit --use-xcframeworks` from the same directory as your Cartfile.

If your Cartfile is *not* configured to allow an update, edit the line for the Alchemer Mobile (ApptentiveKit) framework with the version you would like to update to. We recommend a setting like `'~>6.5'` which will allow you to more easily receive updates for bug fixes and non-breaking changes. You can then run `carthage update ApptentiveKit --use-xcframeworks` as described above.

## XCFramework

If you integrate via our pre-built XCFramework, you will have to manually download the latest framework from our [Github releases page](#) to replace the one included in your project.

## Sub-Project

If you integrate Alchemer Mobile (ApptentiveKit) as a subproject, you will have to manually pull or download the latest release from our GitHub repository.

# Import Alchemer Mobile (ApptentiveKit)

## Swift

Add `import ApptentiveKit` to each Swift file where you plan to reference Alchemer Mobile methods and properties (typically where you would also import the UIKit or Foundation frameworks).

## Objective-C

Add `@import ApptentiveKit;` to each implementation file where you plan to reference Alchemer Mobile (ApptentiveKit) methods and properties (typically where you would import its corresponding header file).

## Initialize the SDK

Early in your app's lifecycle, it should call Alchemer Mobile's `register(with:completion:)` method. For example:

```
import UIKit
import ApptentiveKit

class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        Apptentive.shared.register(with: .init(key: "<#Your Apptentive App Key#>", signature: "<#Your Apptentive App Signature#>"))

        // Set the log level to debug Apptentive
        ApptentiveLogger.default.logLevel = .debug

        // Other app initialization...

        return true
    }
}
```

**NOTE:** The easiest way to get started with Alchemer Mobile (ApptentiveKit) is to use the shared instance of the Apptentive class ( `Apptentive.shared` ) to call its methods and access its properties. If you prefer, you can also explicitly create an instance of the Apptentive class ( `let myApptentiveInstance = Apptentive()` ) and manage the process of passing it to the places in your app that it will be used. In this case your app must never access the `shared` static property of the Apptentive class. Doing so will result in an assertion failure in debug builds and unpredictable behavior in release builds.

## Engage Events

Events record user interaction. You can use them to determine if and when an Interaction will be shown to your customer. You will use these Events later to target Interactions, and to determine whether an Interaction can be shown. You trigger an Event with the `engage(event:from:)` method. This will record the Event, and then check to see if any Interactions targeted to that Event are allowed to be displayed, based on the logic you set up in the Alchemer Mobile Dashboard.

One good place to engage an event is when a view controller appears, for example in the view controller's `viewDidAppear(_:)` method:

```

override func viewDidAppear(animated: Bool) {
    super.viewDidAppear(animated)

    // Engage the "viewed_list" event.
    Apptentive.shared.engage(event: "viewed_list", from: self)
}

```

Another good place to engage events is from the action methods in your view controllers:

```

@IBAction func likeArticle(_ sender: AnyObject?) {
    // ...

    // Engage the "liked_article" event.
    Apptentive.shared.engage(event: "liked_article", from: self)
}

```

If you want to engage an event when a modal view is dismissed, you will want to call the engage method in the completion block, and pass the presenting view controller as the from parameter:

```

@IBAction func save(_ sender: AnyObject?) {
    // save the item...

    self.dismiss(animated: true) {
        // Engage the "saved_item" event.
        Apptentive.shared.engage(event: "saved_item", from: self.presentingViewController)
    }
}

```

This ensures that the view controller you pass in will still be visible if the event you engage results in an interaction being displayed.

Finally, you can engage an event when your app encounters an error:

```

do {
    try context.save()
} catch let error as NSError {
    print("Error saving context: \(error)")

    // Engage the "core_data_save_failed" event.
    Apptentive.shared.engage(event: "core_data_save_failed", from: self)
}

```

This may allow you to let users know if there is a workaround or app update that fixes the problem.

## Using a Variable as the Event Name

The `engage(event:from:)` method accepts an Event object as its first argument (the Event object conforms to the `ExpressibleByStringLiteral` protocol, so in most cases you can treat events as strings). If you are passing a non-literal string, you will have to wrap it in an Event object:

```
engage(event: .init(name: myEventName), from: self)
```

We recommend that your app engage at least 10 distinct Events. This gives you the flexibility to choose the best place for Interactions to display after your app is live, without having to update your app.

Our web dashboard works best for up to several dozen unique event names. It does not work well if you auto-generate thousands of unique event names. If you plan to target users based on viewing a piece of content out of a collection of hundreds or thousands (say, SKUs in an online retail app), do not create event names for each piece of content. Instead, you can use Custom Person Data for item viewed.

For example, you could set a key of `viewed_item` with a value `123456`. You could then target users for whom that key matches, or is not null.

## Monitoring Events

The SDK will post a notification ( `Notification.Name.apptentiveEventEngaged` ) to the default `NotificationCenter` when an event is engaged, whether the source of that event is your app or the SDK itself.

Your app can listen to this notification and then examine the values in the `userInfo` dictionary for the following keys:

- `eventType` : the extended name of the event, for example `com.apptentive#Survey#submit`
- `interactionType` : the type of the interaction that engaged the event, or `app` if not applicable
- `interactionID` : the internal identifier of the interaction that engaged the event, if applicable
- `eventSource` : the source of the event, either `com.apptentive` (for events that are engaged by the SDK itself) or `local.app` (for events that are engaged by your app)

## List of Internal Events

- `com.apptentive#<InteractionType>#launch` (when an interaction is launched)
- `com.apptentive#<InteractionType>#cancel` (when an interaction is dismissed)
- `com.apptentive#app#launch` (when the app enters the foreground, or the SDK is first initialized following the app being terminated)
- `com.apptentive#app#exit` (when the app enters the background)
- `com.apptentive#Survey#submit`
- `com.apptentive#Survey#continue_partial` (when a user continues with a survey after an action sheet is displayed that explains that they will discard any answers that they entered)
- `com.apptentive#Survey#cancel_partial` (when a user exits a survey after the aforementioned action sheet is displayed)
- `com.apptentive#MessageCenter#read` (when a previously-unread message in Message Center is displayed to the user)
- `com.apptentive#AppleRatingDialog#request` (when `requestReview` is called on `SKStoreReviewController` )
- `com.apptentive#AppleRatingDialog#shown` (when the above method call results in a review request being presented)
- `com.apptentive#AppleRatingDialog#not_shown` (when the above method call does not result in a

review request being presented)

- `com.apptentive#EnjoymentDialog#yes` (when the user taps Yes in the Love Dialog)
- `com.apptentive#EnjoymentDialog#no` (when the user taps No in the Love Dialog)
- `com.apptentive#NavigateToLink#navigate` (when the SDK has opened a URL from a Note button)
- `com.apptentive#TextModal#interaction` (when the SDK launches an interaction from a Note button)
- `com.apptentive#TextModal#dismiss` (when the user taps the dismiss button in a Note)

## Message Center

With the Alchemer Mobile Message Center your customers can send feedback, and you can reply, all without making them leave the app. Handling support inside the app will increase the number of support messages received and ensure a better customer experience.

Message Center lets customers see all the messages they have sent you, read all of your replies, and even send screenshots that may help debug issues.

**Note:** Message Center uses a `QLPreviewController` instance to display attachments, which includes a default share sheet that allows saving images to the device's photo library.

This feature requires a key to be added to the app's `Info.plist` file under the "Privacy – Photo Library Usage Description" key. We suggest setting the value to something like "This will enable the Save Image feature for attachments."

If this key is not present, iOS 15 devices will omit the Save Image option from the share sheet. Versions prior to iOS 15 will crash (in development builds) or fail silently (in release builds) if a photo library usage description is not set and the user chooses the Save Image option from the share sheet.

## Showing Message Center

Find a place in your app for a button that will launch Message Center. This will allow customers to contact you with feedback, or questions if they are having trouble using your app, as well as allow them to see your responses.

```
import UIKit
import ApptentiveKit

class SettingsViewController: UIViewController {

    // ...

    @IBAction func openMessageCenter(sender: UIButton) {
        Apptentive.shared.presentMessageCenter(from: self)
    }
}
```



## Checking Unread Message Count

You can also check to see how many messages are waiting to be read in the customer's Message Center using Apptentive's `unreadMessageCount` property. This property is compatible with Key-Value Observing (KVO), so your app can monitor it and be notified when it changes:

```
var observation = Apptentive.shared.observe(\.unreadMessageCount, options: [.new]) { _, change in
    print("Unread message count changed to: \{(change.newValue!)}")
}
```

## Attachments

Attachments are messages that you can send from the SDK programmatically, which will be visible to you in the Conversation View, but will not be visible to your customers in Message Center. They are great for sending contextual information or logs from rare crash events.

### Hidden File Attachments

```
let fileData = try Data(contentsOf: fileURL)

sendAttachment(fileData, mediaType: "application/zip")
```

### Hidden Image Messages

```
let image = UIImage(named: "my image")

sendAttachment(image)
```

### Hidden Text Messages

```
sendAttachment("Error creating file: \{error}")
```

## Presenting Message Center with Custom Data

You can attach custom data when presenting message center. Then, if a consumer sends a message, your custom data will be associated with the message and visible in the Conversations view in the dashboard. If the consumer closes Message Center without sending a message, the custom data will not be sent to the dashboard (if you want to record data regardless of whether a message is sent, consider using person or device custom data).

```
let customData = ["widgetID": 123]
Apptentive.shared.presentMessageCenter(from: self, with: customData)
```

As with person and device custom data, custom data must be a `Dictionary` with `String` keys and values that are numeric, Boolean, or strings. Nested data (such as arrays and sub-dictionaries) are not supported.

## Customer Information

### Set Customer Contact Information

If you already know the customer's email address or name, you can pass them to us to display in the conversation view on your Alchemer mobile dashboard.

```
Apptentive.shared.personEmailAddress = <#Email Address#>
Apptentive.shared.personName = <#Person Name#>
```

Message Center provides dialogs that allow your customers to set their name and email as well. Calling the above methods will overwrite what your customer enters. If you don't want to overwrite what they enter, you can check their values first, using

```
Apptentive.shared.personEmailAddress and Apptentive.shared.personName .
```

## Custom Data

You can send Custom Data associated with a person's profile that is using the app, or the device. In particular, this is useful for sending a Customer ID and other information that helps you understand and support your users better. Custom Data can also be used for configuring when Interactions will run. You can add custom data of type `String`, `Int`, and `Bool`.

In general, Custom Data can be sent as Person Custom Data or Device Custom Data. However, if sending a Customer ID, you must send it as Person Custom Data. For more on the benefits of setting a Customer ID, see [here](#).

After the Custom Data field has been set, it will appear on the targeting screen for any Interaction within a few minutes. You may need to refresh your browser to see recent changes.

```
Apptentive.shared.personCustomData["CustomerID"] = "1234321"
Apptentive.shared.personCustomData["city"] = "Seattle"
Apptentive.shared.personCustomData["points"] = 500
Apptentive.shared.personCustomData["is_premium"] = true

Apptentive.shared.deviceCustomData["primary_account"] = "test@apptentive.com"
Apptentive.shared.deviceCustomData["user_count"] = 5
Apptentive.shared.deviceCustomData["full_version"] = false
```

Because the `CustomData` type is a `struct`, you will need to use these alternative methods in Objective-C:

```
[Apptentive.shared addCustomPersonDataString: @"1234321", withKey: @"CustomerID"];
[Apptentive.shared addCustomPersonDataString: @"Seattle", withKey: @"city"];
[Apptentive.shared addCustomPersonDataNumber: @500, withKey: @"points"];
[Apptentive.shared addCustomPersonDataBool: true, withKey: @"is_premium"];

[Apptentive.shared addCustomDeviceDataString: @"test@apptentive.com", withKey: @"primary_account"];
[Apptentive.shared addCustomDeviceDataNumber: @5, withKey: @"user_count"];
[Apptentive.shared addCustomDeviceDataBool: false, withKey: @"full_version"];
```

## Customer Authentication

If your app involves sensitive data and is likely to be used by multiple customers on the same device, you can use Customer Authentication to protect a customer's information after they log out of your app.

Customer Authentication requires that you have authentication built into your app, and will also require you to modify your server's authentication code to pass authentication information back to your app, and then to the Alchemer Mobile (ApptentiveKit) SDK. For more information on this feature, see our [Customer Authentication Configuration](#).

When not using Customer Authentication, the Alchemer Mobile (ApptentiveKit) SDK will still function normally using the built-in safeguards of iOS, but information passed to the SDK will be accessible to someone in possession of an unlocked device.

### How we log a customer in

Your server will authenticate a customer when they log in. At that time, you will need to generate a JSON Web Token (JWT). The JWT should contain the following claims:

- A subject ( `sub` ) claim that uniquely identifies the customer
- An issuer ( `iss` ) claim corresponding to your company
- An issued-at ( `iat` ) claim corresponding to the date/time of issue
- An expiry ( `exp` ) claim corresponding to the expiration date. Note that expiry dates are capped at three days after the issued-at date.

The resulting JWT should be signed with the JWT Signing Secret in your app's [API & Development](#) page using the HS-512 algorithm.

Alchemer Mobile will securely manage the JWT. It is important to never reuse a JWT and NEVER INCLUDE THE JWT SIGNING SECRET IN YOUR APP or anywhere it could be accessed by unauthorized parties.

### Logging a Customer In

The JWT will be a string. When your server generates a JWT, you will need to send it back to your app, and then log in to Alchemer Mobile with it:

```
try Apptentive.shared.logIn(with: jwt) { result in
  switch result {
  case .success:
    // handle success case

  case .failure(let error):
    // handle failure case
  }
}
```

### Logging a Customer Out

You should make sure to log a customer out any time you invalidate the customers session in your app. That means that when a customer explicitly logs out, you should also log them out of Alchemer Mobile. When they are logged out after a certain amount of time, you should likewise also log them out of Alchemer Mobile.

```
Apptentive.shared.logout()
```

Note: If your customer has logged out of your app, but you don't log them out of Alchemer Mobile, their personal information may be visible to other app users.

## Refreshing the JWT

Because of the limited time during which the JWT is valid, it should be periodically refreshed using the `updateToken(_:completion:)` method:

```
try self.apptentive.updateToken(jwt) { result in
    switch result {
    case .success:
        // handle success case

    case .failure(let error):
        // handle failure case
    }
}
```

## Handling Authentication Failures

If the JWT expires or otherwise becomes invalid, events, messages, survey responses and the like will pause sending. By adopting the `ApptentiveDelegate` protocol, the Alchemer Mobile SDK can notify your app and give your it an opportunity to refresh the JWT using the aforementioned method.

```
func authenticationDidFail(with error: Error) {
    print("Apptentive authentication failed with error: \(error)")

    MyAPI.getApptentiveToken(for: MyUser.current) { result in
        switch result {
        case .success(let jwt):
            Apptentive.shared.updateToken(jwt) { updateResult in
                switch updateResult {
                case .success:
                    // handle success case
                case .failure:
                    // handle failure case
                }
            }

        case .failure(let error):
            // handle failure case
        }
    }
}
}Logged Out Experience
```

When no customer is logged in, Alchemer Mobile public API methods will have no effect or in some cases return an error result.

If you are using Message Center, and the button that launches it is visible in a part of your app that your customers can access without logging in to your app, you should follow the Message Center instructions above to only show or enable the button when Message Center can be shown.

## Logging

To set the level of importance for events logged to the system log, set the `logLevel` static property on `ApptentiveLogger`. The available levels are:

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `fault`

## Showing or Hiding Sensitive Information

There are three privacy levels for information logged by the SDK:

- `auto`: This will print the information unreacted when the Alchemer Mobile instance's `shouldHideSensitiveLogs` property is `false`. The default value of this property matches the behavior of `private` log messages.
- `private`: This will hide the information unless a debugger is currently attached to the process, as when debugging through Xcode.
- `public`: This will always show the information.

Strings and objects logged by the SDK are marked `.auto`, and numbers and booleans are marked `.public`.

## Theming and Customization

There are several different options for customizing the look and feel of Alchemer Mobile (ApptentiveKit's) interactions.

### Themes

The SDK will ordinarily apply a styling theme using Alchemer Mobile default colors alongside system fonts and symbols. This theme is designed to look consistent across our iOS and Android SDKs, but may differ from your app's look and feel and the rest of the iOS system.

Carrier 8:51 AM

## What should we build? ✕

Please help us figure this out!

Which two qualities for an app are the most important to you?

select up to 2

- Better user interface
- Cloud support
- Login with Facebook / Google / Twitter


Is there anything you'd like to add?

**Submit**

Survey using .apptentive theme

Carrier 8:07 AM

## Message Center ✕



**Hello!**



We'd love to get feedback from you on our app. The more details you can provide, the better.

FEBRUARY 25, 2022

Test Message

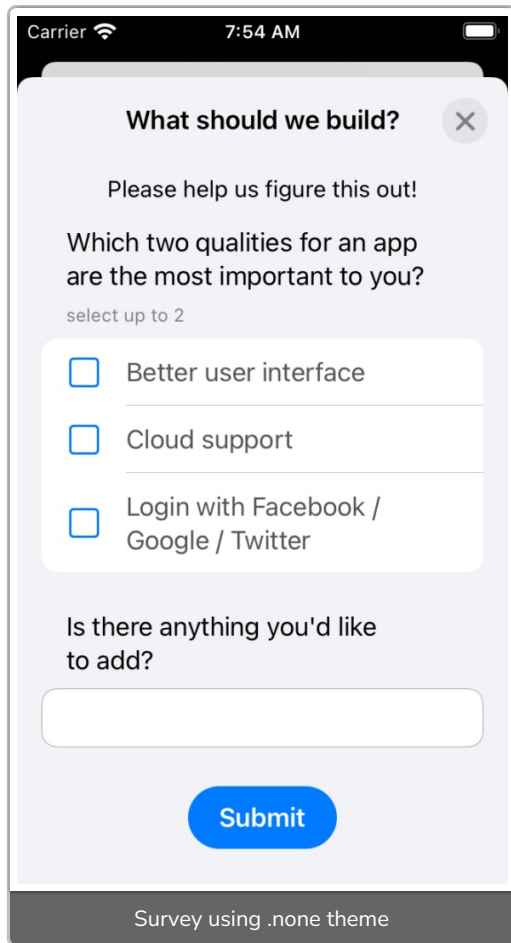
Sent Today, 8:06 AM

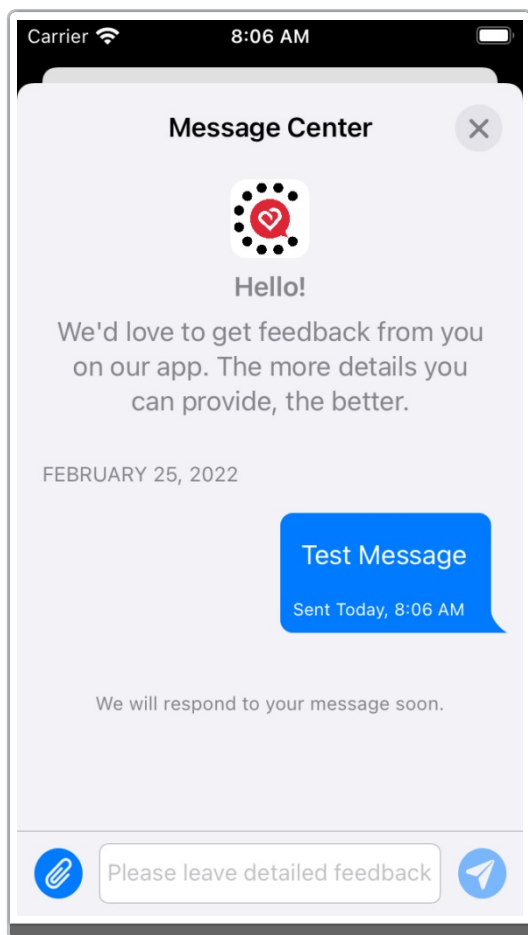
We will respond to your message soon.

Message Center using .apptentive theme

By setting the Alchemer Mobile's class's `theme` property to `.none` (note: this must be done before calling the `register(with:completion:)` method), the interactions take on a more native iOS look and feel, but it will differ substantially from the Android SDK.





Message Center using .none theme

Setting the theme to `.none` may also provide an easier starting point for customizing the user interface to your liking.

## UIAppearance

With the use of the `.none` theme, the Alchemer Mobile (ApptentiveKit) interaction UI will pick up certain `UIAppearance` overrides that your app has set (for example, navigation bar tint).

If an appearance setting in your app makes Alchemer Mobile (ApptentiveKit) interactions unattractive or unreadable, you can use

`appearance(whenContainedInInstancesOf: [ApptentiveNavigationController.self])` to make a corrective appearance change to Alchemer Mobile (ApptentiveKit) interactions (specifically those, like Message Center and Surveys, that use view controllers other than `UIAlertController`).

## UIKit Extensions

For styling changes that aren't amenable to `UIAppearance`, Alchemer Mobile (ApptentiveKit) defines a number of extensions to common `UIKit` classes to allow setting colors, fonts, images, and more. For more information, see our forthcoming Customization Guide.



## Overriding InteractionPresenter

For particularly extensive customizations, we recommend subclassing the `InteractionPresenter` class and setting your subclass as the value for Alchemer Mobile's `interactionPresenter` property. You can override the methods that present each interaction type, instantiating your own view controllers and presenting them as you see fit. A view model for each interaction is provided to configure your view controllers for displaying the interaction and reacting to user input. More information on this technique will be added soon.

Related Articles

---