Alchemer Mobile SDK Flutter Plugin Guide

Supported Platforms

This Flutter Plugin wraps our existing native Android and iOS SDKs. The complete descriptions for the features contained in our SDKs are located here:

- Android API level 21+
- iOS version 11+

Android System Requirements

- `minSDK` version: >= 21 (Android 5.0)
- `compileSDK` version: 31 to 33
- Gradle version: >= 7.0.0
    - `classpath 'com.android.tools.build:gradle:7.0.0'`

Prerequisites (Android only)

- Your MainActivity (Android Activity) should inherit from `FlutterFragmentActivity`
- You will need to update your app to use Material Components
    - This should be a simple process and is highly recommended.
    - There are Bridge themes available if you cannot inherit them from the MaterialComponents theme.

> It is important to inherit your MainActivity (Launcher Activity in your manifest) from FlutterFragmentActivity and using Material theme starting from Alchemer Mobile (Apptentive) Flutter Version 6.1.0.

Adding Alchemer Mobile (formerly Apptentive) Flutter Plugin to Your App

Run this command in your Flutter project to add Apptentive_Flutter as a dependency:

```
$ flutter pub add apptentive_flutter
```

It will add this line to your project's *pubspec.yaml*:

```
dependencies:
  apptentive_flutter: ^6.1.0
```

Registering Alchemer Mobile

Import Alchemer Mobile to your project:

```
import 'package:apptentive_flutter/apptentive_flutter.dart';
```

You should register Alchemer Mobile when your app starts. Create a function to register Alchemer Mobile (and call it in your app's initState() function.

```
class _MyAppState extends State<MyApp> {
  @override
  void initState() {
    super.initState();
    // Initialize Apptentive on app startup
    // This is a function you will create in the next section
    initializeApptentive();
  }
}
```

Upon initialization, detect which platform your customer is using to properly set which Alchemer Mobile Dashboard key and signature the Alchemer Mobile SDK will be registered with. You will need to make at least two Alchemer Mobile Dashboards for your Flutter app, one for Android and one for iOS. You can find your key and signature by going to Settings > API & Development for the targeted app.

```
import 'dart:io' show Platform;

...

Future<void> initializeApptentive() async {
    final String apptentiveKey;
    final String apptentiveSignature;
    if (Platform.isAndroid) {
      apptentiveKey = "<YOUR_ANDROID_KEY>";
      apptentiveSignature = "<YOUR_ANDROID_SIGNATURE>";
    } else if (Platform.isIOS) {
      apptentiveKey = "<YOUR_IOS_KEY>";
      apptentiveSignature = "<YOUR_IOS_SIGNATURE>";
    } else {
      ...
    }
    ...
}
```

Once you set the correct key and signature, use them to register the Alchemer Mobile SDK by creating an Alchemer Mobile Configuration. Inside the Alchemer Mobile Configuration, also set the log level for logs printed by the SDK while your app is running. It is recommended to keep it at *LogLevel.debug* while developing, and *LogLevel.info* for production.

Finally, register Alchemer Mobile (Formerly Apptentive) using the configuration you created using the *register()* function. It a bool saying if it was successfully registered or not.

```
Future<void> initializeApptentive() async {
  ...
  final ApptentiveConfiguration configuration = ApptentiveConfiguration(
      apptentiveKey: apptentiveKey,
      apptentiveSignature: apptentiveSignature,
      // Set your log level here
      logLevel: LogLevel.debug
  );
  bool successful = await ApptentiveFlutter.register(configuration);
}
```

Alchemer Mobile (formerly Apptentive) Configuration optional parameters (Android Only)

The configuration options are enabled with defaults that are production-ready. Check out here for more details.

Register Callback Listeners

Set your callback functions. Supported callback functions are:

- *surveyFinishedCallback* – When a survey finishes or excited, returns if the survey was completed or not.
- *authenticationFailedCallback* – When authentication fails, returns the reason and the error message. (Coming soon on 6.x)
- *messageCenterUnreadCountNotificationCallback* – When a customer receives a message in Message Center, returns the new number of unread messages. This will be called within ~15 seconds if Message Center is opened, and within ~5 minutes if Message Center is closed.
- *messageSentNotification (iOS only)* – When a customer sends a message in Message Center, returns the sender as a String.

```
ApptentiveFlutter.surveyFinishedCallback = (bool completed) {
  print("Survey Finished?: ${completed}");
};

ApptentiveFlutter.authenticationFailedCallback = (String reason, String errorMessage) {
  print("Authentication failed because due to following reason: ${reason} Error message: ${errorMessage}");
};

ApptentiveFlutter.messageCenterUnreadCountChangedNotification = (int count) {
  print("Message Center unread message count is now: ${count}");
};

ApptentiveFlutter.messageSentNotification = (String sentByUser) {
  print("Message sent by user: " + sentByUser);
};
```

IMPORTANT: Register callback listeners **after** you've registered the Alchemer Mobile SDK. This allows the SDK to call the callback functions you set.

```
ApptentiveFlutter.registerListeners();
```

Add Events

Events record user interaction. Use them to determine when an Interaction will be shown to your customers. At a minimum, you should include 20 – 50 Events in your app to start taking advantage of Alchemer Mobile. For tips and ideas, check out this guide.

Add Events to your app using the *engage(eventName)* function. When this function is called, Alchemer Mobile 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 your Alchemer Mobile Dashboard.

```
ApptentiveFlutter.engage(eventName: "Button_Pressed");
```

The *engage()* function also returns a *Future<bool>* determining if the Event was successfully engaged or not. **All Alchemer Mobile (Apptentive) functions return a success bool.**

```
ApptentiveFlutter.engage(eventName: "Button_Pressed").then((success) {
  if (!success) {
    // Not engaged, do something
  } else {
    // Engaged, do something
  }
}
```

Add Message Center

Our Message Center will allow your customers to contact you with feedback or questions about your app. For more details, see this guide. Message Center has two ways of being launched:

- From other Interactions. Message Center can be shown to customers that say “No” on the Love Dialog, presented as a button on Notes. From there, customers can send feedback and start a conversation.
- Being opened by a button or action via code in your app, such as when your customer taps on a “Leave Feedback” button. Use the *showMessageCenter()* function to display the Message Center. If using Message Center in any capacity, we recommend using this method so customers can contact you on their time.

```
OutlinedButton(
  onPressed: () {
    ApptentiveFlutter.showMessageCenter();
  },
  child: Text('Leave Feedback'),
),
```

Check Unread Message Count

To check how many unread messages a customer has in Message Center, use the following function:

```
ApptentiveFlutter.getUnreadMessageCount().then((count) {
  print("Unread Message Count: $count");
});
```

Add Custom Data

Custom Person and Device Data

To add Custom Person Data, use the *addCustomPersonData(key,value)* function. For Custom Device Data, use *addCustomDeviceData(key,value).* The key is the name of the Custom Data, and the value is the data itself. We support boolean, string, and numeric values. If you add a different value of Custom Data to a key you've already added, the old data will be overwritten.

We strongly recommend sending a customer ID as Custom Person Data as we offer deeper reporting capabilities and exports based on this information.

```
ApptentiveFlutter.addCustomPersonData(key: "name", value: "data");
ApptentiveFlutter.addCustomDeviceData(key: "name", value: "data");
```

Generally, Custom Data does not need to be deleted. But if necessary for your app, use the *removeCustomPersonData(key)* function. This will completely remove the custom data for your customer. Similarly, use *removeCustomDeviceData(key)* to remove Custom Device Data.

```
ApptentiveFlutter.removeCustomPersonData(key: "name");
ApptentiveFlutter.removeCustomDeviceData(key: "name");
```

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 using the *setPersonName()* and *setPersonEmail()* functions.

```
ApptentiveFlutter.setPersonName(name: name);
ApptentiveFlutter.setPersonEmail(email: email);
```

Push Notifications

Alchemer Mobile can send push notifications to ensure your customers see your replies to their feedback in Message Center.

Supported Push Providers

- FCM
- GCM
- Amazon SNS
- Airship

Integrating with Alchemer Mobile

Use the following method to set your push integration with Alchemer Mobile

```
ApptentiveFlutter.setPushNotificationIntegration(provider: <your_provider_here>, token: <your_token_here>);
```

The following are the provider enums you can add to the method call:

- PushProvider.apptentive (For FCM and GCM)
- PushProvider.amazon
- PushProvider.urban_airship

For the tokens, use these:

- FCM, GCM, Amazon SNS token strings
- Airship channelID string

Android / iOS Differences

On Android, your push provider will receive a message payload when the app is in the background and a response is sent to the Message Center conversation in the dashboard. You will need to create your own notification using the data in the message payload.

On iOS, when a customer sends a message in Message Center for the first time, they will be asked if the app can send them notifications (if they have not been asked before). If you are using an Alchemer Mobile Push integration, then a push notification will show to the customer when a response is sent to their conversation in Message Center and the app is closed or in the background.

Also, for iOS, make sure you properly set up your application for push. You can follow the steps in this guide to learn more.

> Remember to set up your push integration in your Alchemer Mobile Dashboard under Settings > Integrations.

Other

Styling Alchemer Mobile (Android only)

When modifying your Android project in your Flutter app, follow the native instructions to customize how Alchemer Mobile looks for you.

As of 6.1.0 styling is available only for Android

Multi user (Coming soon for 6.x)

If you have multiple consumers using your app, you may want to use Multi user to protect each customer's information from one another. See our native guides for details: Android | iOS

Hidden Message Center messages

Starting from Alchemer Mobile (Apptentive) Flutter 6.1

If there is additional info you'd like to send to the server that will help with working with a consumer in Message Center, you can send hidden messages will not be shown on the device, but will be shown in support agent's message thread with the consumer.

```
ApptentiveFlutter.sendAttachmentText(message: "<your message>");
```

Unsupported Features

If you are interested in any of the below features, please let us know:

Hidden Attachments

Sending hidden attachments is not currently supported by our Flutter Plugin.

Related Articles