Xamarin.Android Plugin

This document will show you how to integrate the Alchemer Mobile SDK into your Xamarin.Android app, configure it, and test to make sure it's working properly. Each section lists the minimum necessary configuration, as well as optional steps. Click here for migration documentation if you have previously used our Xamarin Android plugin.

System Requirements

Minimum SDK version: 21 (Android 5.0)

Dependencies

Our SDK has a dependency on the following support libraries.

- GoogleGson
- Xamarin.AndroidX.Legacy.Support.V4
- Xamarin Kotlin StdLib
- Xamarin.Google.Android.Play.Core
- Xamarin.AndroidX.Fragment
- Xamarin.AndroidX.Fragment.Ktx
- Xamarin.AndroidX.AppCompat
- Xamarin.AndroidX.Lifecycle.Common
- Xamarin.AndroidX.Lifecycle.LiveData
- Xamarin.Google.Android.Material
- Xamarin.GooglePlayServices.Base

> If you use a newer version of any support library, you will need to include a newer version of all four of the above libraries in your app. This avoids potential issues caused by a mismatch in support library versions.

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.

Locale Qualifier	Language Name
None	English
`ar`	Arabic
`el`	Greek
`da`	Danish

Locale Qualifier	Language Name
`de`	German
`es`	Spanish
`fr`	French
`fr-rCA`	French Canadian
`it`	Italian
`ja`	Japanese
`ko`	Korean
`nl`	Dutch
`pl`	Polish
`pt`	Brazilian Portuguese
`ru`	Russian
`sv`	Swedish
`tr`	Turkish
`zh`	Chinese (Traditional)
`zh-rCN`	Chinese (Simplified)

Adding Alchemer Mobile (formerly Apptentive)

NuGet

Using the NuGet package manager is the easiest way to integrate. Follow the official Microsoft Visual Studio NuGet guide to add the following package to your project:

- Apptentive.Android

Register Alchemer Mobile (Apptentive)

Register Alchemer Mobile (Apptentive) in your `Application` class.

```
using System;
using Android.App;
using Android.Runtime;
using ApptentiveSDK;
using ApptentiveLogLevel = Apptentive.Com.Android.Util.LogLevel;

namespace ApptentiveSample
{
   [Application]
   public class MyApplication : Application
   {
      public MyApplication(IntPtr handle, JniHandleOwnership ownerShip)
         : base(handle, ownerShip)
      {
      }

      public override void OnCreate()
      {
         base.OnCreate();
         var configuration = new ApptentiveConfiguration("Your Apptentive Key", "Your Apptentive Signature");
         configuration.LogLevel = ApptentiveLogLevel.Verbose;
         ApptentiveSDK.Apptentive.Register(this,configuration);
      }
   }
}
```

> Make sure you use the Alchemer Mobile (Apptentive) App Key and Signature for the Android app you created in the Alchemer Mobile console. Sharing these keys between two apps, or using keys from the wrong platform is not supported, and will lead to incorrect behavior. You can find them here.

Styling Alchemer Mobile

Alchemer Mobile will inherit your app's styles by default. If you are using a Light/Dark AppCompat theme, Alchemer Mobile will look like your app by default. But if you are using another theme, or if you want to force Alchemer Mobile to adopt different styles than your app, please follow instructions in Android Interface Customization.

Registering Activity

In order to display a Prompt or love dialog you must extend the Activity you wish to display it from to adopt **IApptentiveActivityInfo,** and register the activity in the `OnResume` method.

```
public class MainActivity : AppCompatActivity, IApptentiveActivityInfo
{
protected override void OnResume()
 {
  base.OnResume();
  ApptentiveSDK.Apptentive.RegisterApptentiveActivityInfoCallback(this);
 }
public Activity ApptentiveActivityInfo
{
   get
   {
    return this;
   }
}
}
```

You can also unregister the activity like so:

```
ApptentiveSDK.Apptentive.UnregisterApptentiveActivityInfoCallback();
```

Message Center

> See: How to Use Message Center

Showing 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 send you, read all of your replies, and even send screenshots that may help debug issues.

Add Message Center to talk to your customers.

Find a place in your app where you can add a button that opens Message Center. Your settings page is a good place.

```
protected override void OnCreate(Bundle savedInstanceState)
{
   ...
   var messageCenterButton = FindViewById<Button>(Resource.Id.messageCenterButton);

    if (ApptentiveSDK.Apptentive.CanShowMessageCenter())
       {
        ApptentiveSDK.Apptentive.ShowMessageCenter();
       }
}
```

Unread Message Count

Use the provided property to get the unread message count for Message Center.

```
protected override void onCreate(Bundle savedInstanceState)
{
    base.OnCreate(savedInstanceState);
    SetContentView(Resource.Layout.Main);
    int unreadCount = ApptentiveSDK.Apptentive.UnreadMessageCount;
}
```

Attachments

Hidden 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

Here are the three methods to send hidden attachments to the dashboard.

```
 ApptentiveSDK.Apptentive.SendAttachmentFile(fileStream, mimeType);
ApptentiveSDK.Apptentive.SendAttachmentFile(byteArray, mimeType);
ApptentiveSDK.Apptentive.SendAttachmentFile(uriPath);
```

You can also send hidden text.

```
 ApptentiveSDK.Apptentive.SendAttachmentText("text");
```

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()` 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.

```
 ApptentiveSDK.Apptentive.Engage("my_event");
```

You can also utilize the callback.

```
 ApptentiveSDK.Apptentive.Engage("my_event",null, (engaged) => Console.WriteLine("Interaction engaged:
" + engaged));
```

Here you can add custom data to your engage call.

```
 IDictionary<string, Java.Lang.Object> customDataDictionary = new   Dictionary<string, Java.Lang.Object>();
customDataDictionary.Add("PersonCustomDataKey", new Java.Lang.String("PersonCustomDataValue"))
ApptentiveSDK.Apptentive.engage("event", customDataDictionary);
```

Interactions

All of the following Interactions can be configured in the Alchemer Mobile Dashboard to show up when any of your Events are engaged.

Love Dialogs

Love Dialogs can help learn about your customer, asking customers that love your app to rate it in the applicable app store, and customer who don't love it yet to give you feedback, or answer a Survey.

> See: How to Use Ratings Prompts

Surveys

> Please set `android:enableOnBackInvokedCallback="false"` if needed. This allows the confirmation dialog to be shown when the user exits the survey when the back button is pressed.

Surveys are a powerful tool for learning about your customers' needs.

> See: How to Use Surveys

Prompts

Prompts (formerly Notes) allow you to show an alert to customers, and optionally direct them to a Survey, Message Center, a Deep Link, or simply dismiss the Prompts.

> See: How to Use Prompts

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

Firebase Cloud Messaging

If you are using Firebase Cloud Messaging (FCM) directly, without another push provider layered on top, please follow these instructions.

1. Follow the FCM instructions to Set Up a Firebase Cloud Messaging Client App .

2. Add the Xamarin.Firebase.Messaging package.

3. In your `FirebaseInstanceIdService`, pass Apptentive your **token**.

```
using System;
  using Android.App;
  using Firebase.Messaging;
  using Android.Util;
  using ApptentiveSDK.Android;

  namespace ApptentiveSample
  {
     [Service]
     [IntentFilter(new[] { "com.google.firebase.INSTANCE_ID_EVENT" })]
     public class MyFirebaseIIDService : FirebaseInstanceIdService
     {
        const string TAG = "MyFirebaseIIDService";
        public override void OnNewToken(string token)
        {
       ApptentiveSDK.Apptentive.SetPushNotificationIntegration(this, ApptentiveSDK.Apptentive.PushProvide
  rApptentive, token);
        }
     }
  }
```

In your `FirebaseMesagingService`, get the title, body, and `PendingIntent` from the incoming push, and create a `Notification` to display to your customer. If the returned `PendingIntent` is `null`, then the push did not come from Alchemer Mobile, and you should handle it yourself.

```
 using System;
using Android.App;
using Android.Content;
using Android.Media;
using Android.Support.V4.App;
using Android.Util;
using ApptentiveSDK.Android;
using Firebase.Messaging;

namespace ApptentiveSample
{
    [Service]
    [IntentFilter(new[] { "com.google.firebase.MESSAGING_EVENT" })]
    public class MyFirebaseMessagingService : FirebaseMessagingService
    {
        const string TAG = "MyFirebaseMsgService";
        /**
         * Called when message is received.
         */
    public override void OnMessageReceived(RemoteMessage message)
    {
    var data = message.Data;
    String title = null;
    String body = null;
    var isPush = ApptentiveSDK.Apptentive.IsApptentivePushNotification(data);
    var channel = new NotificationChannel("channel_id", "channel_name", NotificationImportance.Default)
    {
     Description = "channel_description"
    };
    var notificationManager = GetSystemService(NotificationService) as NotificationManager;
    notificationManager.CreateNotificationChannel(channel);
    int notificationId = 1;
    if (ApptentiveSDK.Apptentive.IsApptentivePushNotification(data))
     {
    title = ApptentiveSDK.Apptentive.GetTitleFromApptentivePush(data);
    body = ApptentiveSDK.Apptentive.GetBodyFromApptentivePush(data);
    ApptentiveSDK.Apptentive.BuildPendingIntentFromPushNotification(this,(pendingIntent) =>
     {
    title = ApptentiveSDK.Apptentive.GetTitleFromApptentivePush(data);
    body = ApptentiveSDK.Apptentive.GetBodyFromApptentivePush(data);
    ApptentiveSDK.Apptentive.BuildPendingIntentFromPushNotification(this,(pendingIntent) =>
     {
    if (pendingIntent == null)
     {
     Console.WriteLine(TAG, "Push notification was not for the active conversation. Doing nothing.");
     return;
    }
     ShowNotification(pendingIntent, title, body, notificationId, notificationManager);
   }, data);
   }
```

Can Show Interaction

Here you can check if an event can show an interaction.

```
if (ApptentiveSDK.Apptentive.QueryCanShowInteraction("event_name"))
{
...
}
```

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.

```
ApptentiveSDK.Apptentive.PersonEmail = email;
```

```
ApptentiveSDK.Apptentive.PersonName = 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.

```
var email = ApptentiveSDK.Apptentive.PersonEmail;
```

```
var name = ApptentiveSDK.Apptentive.PersonName;
```

Custom Data

You can send Custom Data associated with either the device, or the person using the app. This is useful for sending user IDs and other information that helps you support your users better. Custom Data can also be used for configuring when Interactions will run. You can add custom data of type `string`, `number`, and `bool`.

Examples

```
Apptentive.AddCustomPersonData("user_id",(Java.Lang.Number)intValue);
Apptentive.AddCustomPersonData("country", "United States");
Apptentive.AddCustomPersonData("pro_membership", (Java.Lang.Boolean)personBoolValue);

Apptentive.AddCustomDeviceData("wifi_only", (Java.Lang.Boolean)deviceBoolValue);
```

Proguard Rules

Be sure to add the following ProGuard rules to your ProGuard file:

```
 # WORKING PROGUARD RULES:
-keep class apptentive.com.android.feedback.survey.model.* { *; }
-keep class com.google.android.play.core.review.** { *; }
-keep class com.apptentive.android.sdk.** { *; }
-keep class apptentive.com.android.feedback.model.** { *; }
-keep class apptentive.com.android.feedback.engagement.interactions.InteractionData { *; }
-keep class apptentive.com.android.encryption.** { *; }

# Gson library
-keepattributes Signature
-keepattributes *Annotation*
-dontwarn sun.misc.**
-keep class com.google.gson.** { *; }
-keep class org.apache.commons.io.** { *; }
-keep class org.apache.commons.logging.** { *; }
```

Related Articles