Legacy Android Integration Reference

This document will show you how to integrate the Alchemer Mobile (Formerly Apptentive) Android SDK into your app, configure it, and test to make sure it's working properly. Each section lists the minimum necessary configuration, as well as optional steps.

System Requirements

Minimum SDK version: 14 (Android 4.0)

Compile SDK version: <5.8.3 -> 28 (Android 9) , >=5.8.3 -> 31 (Android 12)

> Please refer to the Android SDK Release Notes for troubleshooting updating to specific versions.

Dependencies

Our SDK has a dependency on the following support libraries, version **28.0.0**

- Android v4 Support Library
- Android v7 Appcompat Library
- Android v7 Cardview Library
- Android Design Support Library

> 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

> To ensure the SDK functions properly, when upgrading please make sure to read the Migration Guide for each version above the version you are currently using.

Add Dependency

In your `build.gradle`, add a dependency to Alchemer Mobile (formerly Apptentive), replacing [[SDK_version]] with most recent SDK

```
repositories {
  jcenter()
}

dependencies {
  implementation 'com.apptentive:apptentive-android:[[SDK_version]]'
}
```

When prompted by Android Studio, click **Sync Now**

Note: Alchemer Mobile is open source, available at apptentive/apptentive-android

Register Alchemer Mobile

Register Alchemer Mobile in your `Application` class.

```
public class YourApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();
    ApptentiveConfiguration configuration = new ApptentiveConfiguration("YOUR_APPTENTIVE_APP_KEY", "YOUR_APPTE
NTIVE_APP_SIGNATURE");
    // Set the log level to debug Apptentive
    configuration.logLevel = ApptentiveLog.Level.VERBOSE
    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.

Integrating Without an Application Class

If you didn't already have an `Application` class defined in your app, you will need to create one and add it in your Manifest. Simply create a subclass of `android.app.Application`, make sure it calls `Apptentive.register()` like above, and add it to your `AndroidManifest.xml` in the `<application>` element with `android:name="YourApplication"` like this:

```
<application android:name=".YourApplication"
        android:label="Your App Name"
        android:icon="@drawable/icon"
        android:theme="@style/YourTheme">
```

Migrating from Support Library to AndroidX

Alchemer Mobile SDK migrated to AndroidX in `5.5.0` release. If your application still uses legacy Support Libraries – make sure to use `5.4.x` artifacts instead of the latest `5.5.x`.

For more information check the official guide:
https://developer.android.com/jetpack/androidx/migrate

Delaying Alchemer Mobile SDK Registration

In some cases, you might want to delay Alchemer Mobile SDK registration due to end-user agreements or any async network data fetching. In this case, Alchemer Mobile SDK registration would be divided into two parts:

First, register Alchemer Mobile callbacks in your `Application` class:

```
public class YourApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();
    Apptentive.registerCallbacks(this);
  }
```

Second, register Alchemer Mobile SDK later in the application flow:

```
Application application = ...;
ApptentiveConfiguration configuration = new ApptentiveConfiguration("YOUR_APPTENTIVE_APP_KEY", "YOUR_APPTENT
IVE_APP_SIGNATURE");
Apptentive.register(application, configuration);
```

Device Storage Encryption

If your application maintain sensitive user data (account numbers, health information, etc) and you **pass** it to Alchemer Mobile SDK (as a custom person/device data) – you may want to enable encrypted device storage.

While registering the SDK:

```
ApptentiveConfiguration configuration = new ApptentiveConfiguration("YOUR_APPTENTIVE_APP_KEY", "YOUR_APPTENT
IVE_APP_SIGNATURE");
configuration.setShouldEncryptStorage(true);
Apptentive.register(this, configuration);
```

Most of developers would not need this option since Alchemer Mobile SDK does not store sensitive or security vulnerable information (anything an attacker can use to compromise user accounts).

Customizing Device Storage Encryption

In rare cases, you may need to control how Alchemer Mobile stores sensitive data on the device. To accomplish this you can pass a custom `Encryption` implementation while registering Alchemer Mobile SDK:

```
ApptentiveConfiguration configuration = new ApptentiveConfiguration("YOUR_APPTENTIVE_APP_KEY", "YOUR_APPTENTIVE_APP_SIGNATURE");
Encryption encryption = new Encryption() {
 @Override
 public byte[] encrypt(byte[] data) throws EncryptionException {
  try {
   byte[] encrypted = ...;
   return encrypted;
  } catch (Exception e) {
   throw new EncryptionException(e);
  }
 }
 @Override
 public byte[] decrypt(byte[] data) throws EncryptionException {
  try {
   byte[] decrypted = ...;
   return decrypted;
  } catch (Exception e) {
   throw new EncryptionException(e);
  }
 }
};
configuration.setEncryption(encryption);
Apptentive.register(this, configuration);
```

Make sure you don't change the encryption/decryption algorithms after the app release since Alchemer Mobile won't track it for you.

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.

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.

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

```
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.settings_layout);

 final Button button = (Button) findViewById(R.id.message_center_button);
 Apptentive.canShowMessageCenter(new Apptentive.BooleanCallback() {
  @Override
  public void onFinish(boolean canShowMessageCenter) {
   // Don't show the button until Message Center is available
   if (canShowMessageCenter) {
    button.setVisibility(View.VISIBLE);
    button.setOnClickListener(new View.OnClickListener() {
     @Override public void onClick(View v) {
     Apptentive.showMessageCenter(YourActivity.this);
    }
   });
  } else {
   button.setVisibility(View.GONE);
  }
 }
}
```

Unread Message Count Notification

You can receive a callback when a new unread message comes in. You can use this callback to notify your customer, and display a badge letting them know how many unread messages are waiting for them. Because this listener could be called at any time, you should store the value returned from this method, and then perform any user interaction you desire at the appropriate time.

```
public class MyActivity extends AppCompatActivity {

 private UnreadMessagesListener listener;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.data);

  listener = new UnreadMessagesListener() {
   @Override
   public void onUnreadMessageCountChanged(int unreadMessages) {
    //Your code here
   }
  };
  Apptentive.addUnreadMessagesListener(listener);
 }
}
```

> **Note:** Do not pass an anonymous listener to Alchemer Mobile.

The listener will not be run on the UI thread, and may be called from a different Activity than the one that is set on the listener. The correct way of setting Alchemer Mobile listeners is to create a listener with the same lifecycle as it's intended usage, then passing it to Alchemer Mobile. For

example, if the listener is going to be used in a specific activity, make the listener an Activity data member. This way, when the activity is gone, the listener will automatically be garbage collected and Alchemer Mobile will know about this through `WeakReference` (and we'll stop calling the listener).

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

- Apptentive.sendAttachmentFile(byte[] data, String mimeType)
- Apptentive.sendAttachmentFile(InputStream is, String mimeType)
- Apptentive.sendAttachmentFile(String Uri)

Hidden Text Messages

- Apptentive.sendAttachmentText(String text)

```
 // Send a file.
InputStream is = new FileInputStream("filePath");
Apptentive.sendAttachmentFile(is);

// Send a text message.
Apptentive.sendAttachmentText("Message to display in the conversation view.");
```

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.

One good place to add an Event is when an Activity gains focus.

```
@Override
public void onWindowFocusChanged(boolean hasFocus) {
  super.onWindowFocusChanged(hasFocus);
  if (hasFocus) {
    // Engage an Event called "main_activity_focused".
    Apptentive.engage(this, "main_activity_focused");
  }
}
```

Another is when a button is tapped or clicked.

```
sendButton.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    Apptentive.engage(this, "send_button_clicked");
  }
});
```

When the user performs an action that indicates they are having a good experience:

```
private void userLikedArticle(boolean liked) {
  if (liked) {
    Apptentive.engage(this, "user_liked_article");
  }
}
```

If your app has a settings switch, you can add an Event when the switch is flipped.

```
Switch enableNotificationsSwitch = (Switch) findViewById(R.id.enable_notifications_switch);
enabableNotificationsSwitch.setChecked(Preferences.isShowSpectrum());
enableNotificationsSwitch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
  @Override
  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    Preferences.setEnableNotifications(isChecked);
    toggleNotifications(isChecked);
    Apptentive.engage(SoundColorActivity.this, "toggle_enable_notifications");
  }
});
```

You can also add an Event when your app encounters an error.

```
try {
  processUserInput();
} catch (Exception e) {
  Log.e(TAG, e);
  Apptentive.engage(this, "exception_processing_user_input");
}
```

If you need to know whether Alchemer Mobile launched an Interaction, you can pass in a callback that will tell you whether the interaction was displayed.

```
Apptentive.engage(this, "event_name", new Apptentive.BooleanCallback() {
  @Override
  public void onFinish(boolean interactionDisplayed) {
    if (!interactionDisplayed) {
      MyActivity.this.doSomethingElse();
    }
  }
});
```

You can add an Event almost anywhere in your app, just remember that if you want to show an Interaction at that Event, it needs to be a place where launching an Activity will not cause a problem in your app.

> We recommend that you create at least 20 – 50 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.

Note: Make sure you don't start another `Activity` right after calling `Apptentive.engage()`. If you do, and the `engage()` call launches an Interaction, then the `Activity` you launch will cover it up. Instead, call `engage()` in the `onResume()` of that new `Activity`, or check the return value of `engage()`. If it returns `true`, it is about to launch an `Activity`.

Event Names

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.

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 Dialog & Rating Dialog

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

Prompting customers to leave ratings and reviews with an in-app Rating Dialog is a great way to engage customers and request feedback.

Please note that, if your app implements specific Play Core or Play Services, the Google In-App Review requires at least Play Core version 1.8.0+ and Play Services Base 17.4.0+.

See: How to Use the Love Dialog and Rating Dialog

Setting Rating Provider

If you host your app in an app store other than Google Play, you will need to make sure customers who want to rate your app will be able to do so. To choose which app store the **Rating Dialog Interaction** will take you to, we've built several **Rating Providers**. A **Rating Provider** is an

implementation of the IRatingProvider interface, and its job is to provide a simple interface to open the app store. To use another supported rating provider, you can make a call to Apptentive.setRatingProvider(IRatingProvider ratingProvider). If you would like to use an app store that we don't yet support, you can implement the IRatingProvider interface, and pass your implementation to `setRatingProvider()`.

Supported Rating Providers

- Google Play
- Amazon Appstore

Using the Amazon Appstore Rating Provider

```
Apptentive.setRatingProvider(new AmazonAppstoreRatingProvider());
```

Surveys

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

> See: How to Use Surveys

Survey Finished Listener

You can get a callback when a Survey is finished.

```
Apptentive.setOnSurveyFinishedListener(new OnSurveyFinishedListener() {
  @Override
  public void onSurveyFinished(boolean completed) {
    // Your code
  }
});
```

Notes

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 Note.

> See: How to Use Notes

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

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. Go to Integrations, choose Alchemer Mobile (Apptentive) Push, and enter your FCM Server Key. Follow these steps to find your survey key.
    a. Open the Firebase Console: https://console.firebase.google.com
    b. Click on the appropriate project
    c. Click the little gear in the upper left corner, and select “Project Settings”
    d. Click the “Cloud Messaging” tab
    e. Copy the Server Key
    f. Enter this key in the Apptentive Push integration
3. In your `FirebaseMessagingService`, pass Alchemer Mobile (Apptentive) your **token**.

```
private static final String TAG = "Firebase";
    private static final String CHANNEL_ID = "com.apptentive.NOTIFICATION_CHANNEL_MESSAGE_CENTER";

    @Override
    public void onNewToken(String token) {
      super.onNewToken(token);

      Log.i(TAG, "Firebase instance token: " + token);
      Apptentive.setPushNotificationIntegration(Apptentive.PUSH_PROVIDER_APPTENTIVE, token);
    }
```

4. Still in your `FirebaseMessagingService`, 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 (Apptentive), and you should handle it yourself. You also need to create a notification channel for Android-O.

```
@Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
      super.onMessageReceived(remoteMessage);

      final Map<String, String> data = remoteMessage.getData();

      if (Apptentive.isApptentivePushNotification(data)) {
        Apptentive.buildPendingIntentFromPushNotification(new Apptentive.PendingIntentCallback() {
          @Override
          public void onPendingIntent(PendingIntent pendingIntent) {
            if (pendingIntent != null) {
          String title = Apptentive.getTitleFromApptentivePush(data);
          String body = Apptentive.getBodyFromApptentivePush(data);

          // IMPORTANT: you need to create a notification channel for Android-O
          createNotificationChannel();

          Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
          NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(MyFirebaseMessagingService.this,
    CHANNEL_ID)
              .setSmallIcon(R.drawable.apptentive_ic_stat_notify_a)
              .setContentTitle(title)
              .setContentText(body)
              .setAutoCancel(true)
              .setSound(defaultSoundUri)
              .setContentIntent(pendingIntent);
          NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVI
    CE);
          notificationManager.notify(0, notificationBuilder.build());
        } else {
            // Push notification was not for the active conversation. Do nothing.
        }
          }
        }, data);
      } else {
        // This push did not come from Apptentive. It should be handled by your app.
      }
    }

    private void createNotificationChannel() {
      // Create the NotificationChannel, but only on API 26+ because
      // the NotificationChannel class is new and not in the support library
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(CHANNEL_ID, "Apptentive", IMPORTANCE_DEFAULT);
        channel.setDescription("Apptentive Message Center");
        NotificationManager notificationManager = getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
      }
    }
```

Google Cloud Messaging

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

> Note: GCM has been superseded by FCM. While GCM will continue to work, you should use FCM for new push integrations.

1. Follow the GCM instructions to Set up a GCM Client App .

2. Go to Integrations, choose Apptentive Push, and enter your GCM Server Key.

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

```
@Override
    protected void onHandleIntent(Intent intent) {
      InstanceID instanceID = InstanceID.getInstance(this);
      try {
        String token = instanceID.getToken(getString(R.string.gcm_defaultSenderId), GoogleCloudMessaging.INSTANCE_ID
    _SCOPE, null);
        Apptentive.setPushNotificationIntegration(Apptentive.PUSH_PROVIDER_APPTENTIVE, token);
      } catch (IOException e) {
      }
    }
```

4. In your `GcmListenerService`, 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 (Formerly Apptentive), and you should handle it yourself.

```
@Override
    public void onMessageReceived(String from, final Bundle data) {
      super.onMessageReceived(from, data);

      if (Apptentive.isApptentivePushNotification(data)) {
        Apptentive.buildPendingIntentFromPushNotification(new Apptentive.PendingIntentCallback() {
          @Override
          public void onPendingIntent(PendingIntent pendingIntent) {
            if (pendingIntent != null) {
              String title = Apptentive.getTitleFromApptentivePush(data);
              String body = Apptentive.getBodyFromApptentivePush(data);

              Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
              NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(YourGcmListenerReceiver.this, "
    channel_id")
                  .setSmallIcon(R.drawable.apptentive_ic_a)
                  .setContentTitle(title)
                  .setContentText(body)
                  .setAutoCancel(true)
                  .setSound(defaultSoundUri)
                  .setContentIntent(pendingIntent);
                  NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_S
    ERVICE);
                  notificationManager.notify(0, notificationBuilder.build());
            } else {
              // Push notification was not for the active conversation. Do nothing.
            }
          }
        }, data);
      } else {
        // This push was not for Apptentive. It should be handled by your app.
      }
    }
```

Amazon SNS

If you are already using Amazon Web Services SNS to send pushes to your app, follow these instructions. If you are not already using SNS, please follow the instructions for FCM instead.

1. In your `IntentService`, pass Alchemer Mobile (formerly Apptentive) your **token**.

```
@Override
    protected void onHandleIntent(Intent intent) {
      InstanceID instanceID = InstanceID.getInstance(this);
      try {
        String token = instanceID.getToken(getString(R.string.gcm_defaultSenderId), GoogleCloudMessaging.INSTANCE_ID
    _SCOPE, null);
        Apptentive.setPushNotificationIntegration(Apptentive.PUSH_PROVIDER_APPTENTIVE, token);
      } catch (IOException e) {
      }
    }
```

2. In your `GcmListenerService`, 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 (formerly Apptentive), and you should handle it yourself.

3.

```
@Override
    public void onMessageReceived(String from, final Bundle data) {
      super.onMessageReceived(from, data);

      if (Apptentive.isApptentivePushNotification(data)) {
        Apptentive.buildPendingIntentFromPushNotification(new Apptentive.PendingIntentCallback() {
          @Override
          public void onPendingIntent(PendingIntent pendingIntent) {
            if (pendingIntent != null) {
              String title = Apptentive.getTitleFromApptentivePush(data);
              String body = Apptentive.getBodyFromApptentivePush(data);

              Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
              NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(YourGcmListenerReceiver.this, "
    channel_id")
                  .setSmallIcon(R.drawable.apptentive_ic_a)
                  .setContentTitle(title)
                  .setContentText(body)
                  .setAutoCancel(true)
                  .setSound(defaultSoundUri)
                  .setContentIntent(pendingIntent);
                  NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_S
    ERVICE);
                  notificationManager.notify(0, notificationBuilder.build());
            } else {
              // Push notification was not for the active conversation. Do nothing.
            }
          }
        }, data);
      } else {
        // This push was not for Apptentive. It should be handled by your app.
      }
    }
```

Airship

If you are already using Airship to send pushes to your app, follow these instructions. If you are not currently using push notifications in your app, we recommend integrating with FCM instead.

1. In your `AirshipReceiver`, pass us the Channel ID when it is created or updated.

```
@Override
    protected void onChannelCreated(@NonNull Context context, @NonNull String channelId) {
      super.onChannelCreated(context, channelId);
      Apptentive.setPushNotificationIntegration(Apptentive.PUSH_PROVIDER_URBAN_AIRSHIP, channelId);
    }

    @Override
    protected void onChannelUpdated(@NonNull Context context, @NonNull String channelId) {
      super.onChannelUpdated(context, channelId);
      Apptentive.setPushNotificationIntegration(Apptentive.PUSH_PROVIDER_URBAN_AIRSHIP, channelId);
    }
```

2. When your `AirshipReceiver` receives a push, if it came from Alchemer Mobile (formerly Apptentive), extract the title, body, and a `PendingIntent`, and use them to construct a `Notification` object. If the `PendingIntent` is `null`, the push did not come from Alchemer Mobile (formerly Apptentive), and you will need to handle it yourself.

```
@Override
    protected void onPushReceived(@NonNull Context context, @NonNull PushMessage message, boolean notificationPos
    ted) {
      Bundle pushBundle = message.getPushBundle();

      if (Apptentive.isApptentivePushNotification(bundle)) {
        PendingIntent pendingIntent = Apptentive.buildPendingIntentFromPushNotification(pushBundle);
        if (pendingIntent != null) {
          String title = Apptentive.getTitleFromApptentivePush(pushBundle);
          String body = Apptentive.getBodyFromApptentivePush(pushBundle);
          Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
          NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(context)
            .setSmallIcon(R.drawable.notification)
            .setContentTitle(title)
            .setContentText(body)
            .setAutoCancel(true)
            .setSound(defaultSoundUri)
            .setContentIntent(pendingIntent);
          NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION
    _SERVICE);
          notificationManager.notify(0, notificationBuilder.build());
        } else {
          // This push came from Apptentive, but it's not for the active conversation.
        }
      } else {
        // This push didn't come from Apptentive.
        super.onPushReceived(context, message, notificationPosted);
      }
    }
```

Customer Authentication

At Alchemer Mobile, you are our customer, and we refer to your customers as consumers. If you have multiple consumers using your app, you may want to use Customer Authentication to protect each customer's information from one another. 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 Alchemer Mobile. For more information on this feature, see our Customer Authentication Configuration Guide.

If you do not want to use Customer Authentication, or don't have an authentication mechanism in your app, then Alchemer Mobile will still function, but all information will be stored in the same conversation.

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) with a specific format, and signed with the JWT Signing Secret in your app's API & Development page.

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. You will also need to pass in a callback that will allow you to handle login failures. Your callback must implement the Apptentive.LoginCallback interface.

> Alchemer Mobile will securely manage the JWT. It is important not to reuse a JWT, or to store it in the app.

```
Apptentive.login(customerJwt, yourLoginCallback);
```

Apptentive.LoginCallback

```
public interface LoginCallback {
  void onLoginFinish();
  void onLoginFail(String errorMessage);
}
```

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.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.

Handling Authentication Failures

The JWT you create will have an expiration date, and will be signed with a secret. When the JWT expires, the server will reject any requests made with it. In this case, you should ask your customer to log in again. Other failure reasons are provided as well, but are only likely to occur during integration if there is a mistake in how you generate a JWT.

> It is a good practice to choose an expiration that is longer than your normal session duration

so that Alchemer Mobile does not cause your customer to need to re-authenticate.

```
Apptentive.setAuthenticationFailedListener(yourAuthenticationFailedListener);
```

AuthenticationFailedListener

Implement your AuthenticationFailedListener, and keep a static reference to it, to avoid it being garbage collected. We do not store a strong reference to this listener.

```
public interface AuthenticationFailedListener {
  void onAuthenticationFailed(AuthenticationFailedReason reason);
}
```

Logged Out Experience

When no customer is logged in, Alchemer Mobile's public API methods will no-op. 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 hide the button unless Message Center can be shown.

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.setPersonEmail(String email);
```

```
Apptentive.setPersonName(String 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.

```
Apptentive.getPersonEmail();
```

```
Apptentive.getPersonName();
```

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`, `Number`, and `Boolean`.

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 triggered, 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.

Examples

```
Apptentive.addCustomPersonData("customer_id", 1234567890);
Apptentive.addCustomPersonData("country", "United States");
Apptentive.addCustomPersonData("pro_membership", true);

Apptentive.addCustomDeviceData("wifi_only", true);
```

Custom Rating Providers

There are some scenarios where you may want to utilize a custom rating provider. For example:

- Direct your consumers to a custom rating provider other than the Google Play Store (e.g., Amazon Appstore).
- Enable your consumers to leave a Play Store rating from behind a captive portal by displaying the Play Store as a WebView within your app instead of within the Play Store app, bypassing the need to whitelist a potentially unmanageable list of URLs. That is, some apps may provide some online functionality without consumers being fully connected to the internet (for example, some airlines allow consumers to access entertainment within their app without requiring the consumer to purchase full internet access). In this scenario it may be unmanageable to whitelist all potential URLs of the Play Store *app* due to different URLs based on version, etc., which may result in the rating Play Store app failing to load if not properly whitelisted.

> **Note:** This solution requires an update to your app.

Step 1) Create a custom rating provider

```
import android.content.Context;
import android.content.Intent;
import android.net.Uri;

import com.apptentive.android.sdk.module.rating.IRatingProvider;

import java.util.Map;

class CustomRatingProvider implements IRatingProvider {
   @Override
   public void startRating(Context context, Map<String, String> args) {
      Uri uri = Uri.parse("market://details?id=" + args.get("package"));
      Intent intent = new Intent(Intent.ACTION_VIEW, uri);
      context.startActivity(intent);
   }

   @Override
   public String activityNotFoundMessage(Context context) {
      return "Activity not found";
   }
}
```

Step 2) Register the custom rating provider with the Alchemer Mobile SDK

```
Apptentive.setRatingProvider(new CustomRatingProvider());
```

Other

Customizing the Look and Feel

Please see our Customization Guide for more information.

Permissions

Alchemer Mobile SDK requires some permissions to be granted for its operation. These are:

- `android.permission.ACCESS_NETWORK_STATE` : Required to verify if network connectivity is available.
- `android.permission.INTERNET` : Required to actually transmit data to Alchemer Mobile servers.
- `android.permission.WRITE_EXTERNAL_STORAGE` : Required for caching attachment files from Message Center. As of SDK version 5.1.4, we are limiting this permission to devices with API level 18 or lower.

Removing External Storage Permission

If your app does not require access to external storage – you might want to remove the `WRITE_EXTERNAL_STORAGE` -permission since it might not be needed by your app. To perform this, use a remove instruction for manifest merging:

- Open your `AndroidManifest.xml` file.
- Add the tools-namespace to the root element if not already present:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools" ...>
```

- Add the `remove` instruction for the `WRITE_EXTERNAL_STORAGE` -permission after the other permissions:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                 tools:node="remove"/>
```

Modifying External Storage Permission

If your app requires access to both `WRITE_EXTERNAL_STORAGE` and `READ_EXTERNAL_STORAGE` permissions – you might need to replace existing permission from Alchemer Mobile SDK:

- Open your `AndroidManifest.xml` file.
- Add the tools-namespace to the root element if not already present:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools" ...>
```

- Add the `replace` instruction for the `WRITE_EXTERNAL_STORAGE` -permission after the other permissions:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                 tools:node="replace"/>
```

Related Articles