

# LiveEngage Enterprise In-App Messenger SDK: Android Deployment Guide

Document Version: 1.3.0.1  
August 2016

[Introduction](#)

[Platform Support](#)

[Deployment](#)

[Security](#)

[Deploying the App Messaging SDK](#)

[Download the SDK package](#)

[Basic Integration](#)

[Import the downloaded lp\\_messaging\\_sdk module into your project](#)

[Add to the build.gradle of your app the following lines:](#)

[Add the following permission to your app's AndroidManifest.xml file:](#)

[Code integration:](#)

[Testing Your Integration](#)

[Advanced Configuration Options](#)

[LivePerson API Methods](#)

[initialize](#)

[showConversation](#)

[showConversation with authentication support](#)

[hideConversation](#)

[getConversationFragment](#)

[getConversationFragment with authentication support](#)

[reconnect](#)

[setUserProfile](#)

[registerLPPusher](#)

[unregisterLPPusher](#)

[handlePush](#)

[getSDKVersion](#)

[setCallback](#)

[removeCallBack](#)

[checkActiveConversation](#)

[checkAgentID](#)

[markConversationAsUrgent](#)

[markConversationAsNormal](#)

[checkConversationIsMarkedAsUrgent](#)

[resolveConversation](#)

[shutDown](#)

[logout](#)

[Interface and class definitions](#)

[ICallback](#)

[InitLivePersonCallBack](#)

[AgentData](#)

[LogoutLivePersonCallback](#)

## [LivePerson Callbacks Interface](#)

[Error indication](#)

[Token Expired](#)

[Conversation started](#)

[Conversation resolved](#)

[Connection state has changed](#)

[Agent details changed](#)

[CSAT Screen dismissed](#)

[Conversation marked as urgent](#)

[Conversation marked as normal](#)

## [Configuring the SDK](#)

[Brand](#)

[Styling](#)

[Agent Message Bubble](#)

[Visitor Message Bubble](#)

[System messages](#)

[Feedback screen](#)

[Message Edit Text](#)

[Miscellaneous](#)

[Modifying strings](#)

[Set up your app key to enable push notifications](#)

[Dependencies](#)

[Open source list](#)

# Introduction

This document describes the process for integrating the LivePerson App Messaging SDK into mobile native apps based on Android OS. It provides a high-level overview, as well as a step-by-step guide on how to consume the SDK, build the app with it, and customize it for the needs of the app.

# Platform Support

- **Supported OS:** Android 4.0+ (Gingerbread, ICS, KitKat ,Lollipop, Marshmallow)
- **Certified devices:** Samsung Galaxy S1, Samsung Galaxy S2, Samsung Galaxy S3, Samsung Galaxy S4, Samsung Galaxy S5, Samsung Galaxy S6, Samsung Galaxy S7, Nexus 4, Nexus 5, Nexus 5x, Nexus 6, LG G2, LG G3, LG G4, LG G5, Huawei P8, Huawei P9, HTC M9

## Deployment

- **Embeddable library for AAR:** Binary distribution of an Android Library Project
- **Installers:** Gradle

## Security

Security is a top priority and key for enabling trusted, meaningful engagements.

LivePerson's comprehensive security model and practices were developed based on years of experience in SaaS operations, close relationships with Enterprise customers' security teams, frequent assessments with independent auditors, and active involvement in the security community.

LivePerson has a comprehensive security compliance program to help ensure adherence to internationally recognized standards and exceed market expectations. Among the standards LivePerson complies with are: SSAE16 SOC2, ISO27001, PCI-DSS via Secure Widget, Japan's FISC, SafeHarbor, SOX, and more.

Our applications are developed under a strict and controlled Secure Development Life-Cycle: Developers undergo secure development training, and security architects are involved in all major projects and influence the design process. Static and Dynamic Code Analysis is an inherent part of the development process and, upon maturity, the application is tested for vulnerabilities by an independent penetration testing vendor. On average, LivePerson undergoes 30 penetration tests each year.

## Deploying the App Messaging SDK

To deploy the App Messaging SDK, you are required to complete the following steps:

1. Download the SDK package
2. Setup the SDK package in Android Studio

3. Test the SDK

## Download the SDK package

1. Download the latest Messaging SDK from the following link: [SDK Repository](#)
2. Extract the ZIP file to a folder on your computer.  
There should be 4 items in the downloaded package:
  - a. LP\_Messaging\_SDK/lp\_messaging\_sdk - Module that should be added to your project. This module contains the following:
    - i. LivePerson.class - Main entry point for the messaging SDK
    - ii. .aars files
    - iii. Resources
  - b. SampleApp-Source - demonstrate how to use the messaging SDK.
  - c. SampleApp-APK - sample app installation file.

## Basic Integration

1. **Import the downloaded lp\_messaging\_sdk module into your project**
  - a. In the Android Studio menu bar select: File → New → Import module...
  - b. Navigate to the folder where you extracted the SDK project. Navigate to the lp\_messaging\_sdk module, and click finish.
2. **Add to the build.gradle of your app the following lines:**
  - a. *compileSdkVersion* and *buildToolsVersion* should be at least on version 23.
  - b. Add the following code under the android section:

```
repositories {  
    flatDir {  
        dirs project(':lp_messaging_sdk').file('aars')  
    }  
}
```

- c. Under the dependencies section add the following line:

```
compile project(':lp_messaging_sdk')
```

### Example Build.gradle file

```

apply plugin: 'com.android.application'
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.0"

    repositories {
        flatDir {
            dirs project(':lp_messaging_sdk').file('aars')
        }
    }

    defaultConfig {
        applicationId "xxx"
        minSdkVersion xx
        targetSdkVersion xx
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile project(':lp_messaging_sdk')
}

```

### 3. Add the following permission to your app's *AndroidManifest.xml* file:

```

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

```

#### 4. Add the following imports to your class imports section:

```
import com.liveperson.api.LivePersonCallback;
import com.liveperson.infra.InitLivePersonProperties;
import com.liveperson.infra.callbacks.InitLivePersonCallBack;
import com.liveperson.messaging.TaskType;
import com.liveperson.messaging.model.AgentData;
import com.liveperson.messaging.sdk.api.LivePerson;
```

#### 5. Code integration:

- a. Changes to your main activity - initialize the messaging SDK

```
String brandID = "YourLivepersonAccountIdString";
String appID = <your app package name>
LivePerson.initialize(MainActivity.this, new InitLivePersonProperties(brandID, appID, new
InitLivePersonCallBack() {
    @Override
    public void onInitSucceed() {

    }

    @Override
    public void onInitFailed(Exception e) {

    }
}));
```

**brandID** - is your liveperson account id, if you don't have one please contact your LivePerson representative.

**appID** - your app id, used for registering LP pusher service.

**onInitSuccess** - Callback that indicates the init process has finished successfully.

**onInitFailed** - Callback that indicates the init process has failed.

Example implementation:

```
LivePerson.initialize(MainActivity.this, new InitLivePersonProperties(brandID, appID, new
InitLivePersonCallBack() {
    @Override
    public void onInitSucceed() {
        initFragment();
        LivePerson.setUserProfile(appID, firstName, lastName, phone);
    }
}));
```

```
@Override
public void onInitFailed(Exception e) {
    Toast.makeText(MainActivity.this, "Init Failed",
        Toast.LENGTH_SHORT).show();
}
});
```

- b. The SDK supports 2 operation modes: Activity and Fragment.
  - i. Activity mode implements the toolbar that shows the agent name the consumer is talking with, the *Is Typing* indicator showing when the agent is typing and the menu button.  
In addition to this, when using the activity mode, the SDK deals with initializing the SDK.
  - ii. In fragment mode the SDK returns the conversation fragment to the caller that needs to be placed inside a container. Also, the caller is responsible of initializing the SDK and if needed implement a toolbar or other indicators according to the provided SDK [callbacks](#).

When you want to open the conversation window you need to call to one of the two following APIs.

**Note:** Make sure the init process finished successfully - these should be called from the [onInitSucceed\(\)](#) callback:

- iii. Open conversation window in separate activity - This will start a new conversation activity:

```
LivePerson.showConversation(getActivity());
```

Using this method the SDK implements the controls on the action bar.

- iv. Open conversation window in a fragment - This returns a conversation fragment to be placed in a container in your activity:

```
LivePerson.getConversationFragment();
```

When using fragment mode you should use the provided SDK callbacks in your app in order to implement functionalities like menu items, action bar indications, Agent name and typing indicator.

- c. Push notification support



- i. Get your app's AppKey from [Google GCM](#) and set it in the LiveEngage backend as explained [here](#) to identify your app by LiveEngage.
- ii. On every app launch get the GCM Token from your device and register it on the LiveEngage push service using the [registerLPPusher\(\)](#) API call so it knows which device is supposed to get every push message.
- iii. Upon receiving a push message to your app, [handle](#) it so it is shown to the customer.

```
public class MyGcmListenerService extends GcmListenerService {  
  
    /**  
     * Called when message is received.  
     *  
     * @param from SenderID of the sender.  
     * @param data Data bundle containing message data as key/value pairs.  
     *           For Set of keys use data.keySet().  
     */  
    @Override  
    public void onMessageReceived(String from, Bundle data) {  
  
        // Sends the message into the SDK  
        LivePerson.handlePush(this, data, lpAccount, true);  
    }  
}
```

## Testing Your Integration

After you have integrated the SDK with your app it's time to start your first messaging conversation. To do so, follow these steps:

1. Launch the [agent's console](#) and login.
2. Run your app
3. Click your button that call to the start conversation (from section [4.b](#)).
4. From your app's conversation view send a message.
5. Make sure the message is received on the agent console.

# Advanced Configuration Options

## LivePerson API Methods

Detailed below are the LivePerson API methods that shall be called by the developer, and demonstrated on the sample app.

API Name	Purpose
<a href="#"><u>Initialize</u></a> (Deprecated)	To initialize the resources required by the SDK
<a href="#"><u>Initialize</u></a> with SDK properties object	To initialize the resources required by the SDK with properties object.
<a href="#"><u>showConversation</u></a>	To display the messaging activity
<a href="#"><u>showConversation</u></a> with authentication support	To display the messaging activity with the addition of authentication support
<a href="#"><u>hideConversation</u></a>	To hide the conversation activity
<a href="#"><u>getConversationFragment</u></a>	To get the conversation fragment
<a href="#"><u>getConversationFragment</u></a> with authentication support	
<a href="#"><u>reconnect</u></a>	To reconnect with new authentication key
<a href="#"><u>setUserProfile</u></a>	To take custom parameters about the consumer as an input, set them for the messaging agent, and attach them to the transcript
<a href="#"><u>registerLPPusher</u></a>	To register to LivePerson push services
<a href="#"><u>unregisterLPPusher</u></a>	To unregister from LivePerson push services
<a href="#"><u>handlePush</u></a>	To receive all incoming push messages in a single function
<a href="#"><u>getSDKVersion</u></a>	To return the SDK version
<a href="#"><u>setCallback</u></a>	To gets events from SDK - need to implement <b>LivePersonCallback</b>
<a href="#"><u>removeCallBack</u></a>	To stop getting events from the SDK
<a href="#"><u>checkActiveConversation</u></a>	To check whether there is an active conversation
<a href="#"><u>checkAgentID</u></a>	To return agent data such as, first name, last name, email, avatarURL, through callback

<a href="#"><u>markConversationAsUrgent</u></a>	To mark the current conversation as urgent
<a href="#"><u>markConversationAsNormal</u></a>	To mark the current conversation as normal
<a href="#"><u>checkConversationIsMarkedAsUrgent</u></a>	To check whether the current conversation is marked as urgent
<a href="#"><u>resolveConversation</u></a>	To resolve the current conversation
<a href="#"><u>shutDown</u></a>	To shut down the SDK
<a href="#"><u>clearHistory</u></a>	Clear all conversations from device
<a href="#"><u>logOut</u></a>	Logout from the SDK - when all user data should be removed

### Initialize (Deprecated)

<i>public static void initialize (Context context, String brandId, InitLivePersonCallBack initCallBack)</i>	
context	A context from the host app
brandId	An account Id
initCallBack	An <a href="#"><u>InitLivePersonCallBack</u></a> implementation

To allow user interaction, the Messaging Mobile SDK must be initiated. This API initializes the resources required by the SDK; all subsequent API calls, except to the handlePush, assume that the SDK has been initialized.

When the conversation screen is displayed, the server connection for messaging will be established. If a user session is already active and an additional SDK init call is made, it will be ignored and will not start an additional session.

Note: this method was deprecated please use the new method below.

### Initialize with SDK properties object

<i>public static void initialize (Context context, InitLivePersonProperties initProperties)</i>	
context	A context from the host app
initProperties	An object with all the properties needed to initialize the SDK

To allow user interaction, the Messaging Mobile SDK must be initiated. This API initializes the resources required by the SDK; all subsequent API calls, except to the `handlePush`, assume that the SDK has been initialized.

When the conversation screen is displayed, the server connection for messaging will be established. If a user session is already active and an additional SDK init call is made, it will be ignored and will not start an additional session.

This method gets an `InitLivePersonProperties` which includes the properties needed for the init phase of the SDK.

### **showConversation**

<i>public static boolean showConversation(Activity activity)</i>	
activity	The calling activity

The *showConversation* API displays the messaging screen as a new Activity with the conversation fragment. The consumer can then start or continue a conversation. The conversation screen is controlled entirely by the SDK.

This method returns a boolean value to indicate success or failure in opening the messaging screen. If the operation is successful this method returns *true*, else it returns *false*.

Initiating the conversation screen opens the websocket to the LivePerson Messaging Server.

### **showConversation with authentication support**

<i>public static boolean showConversation(Activity activity, String authenticationKey)</i>	
activity	The calling activity
authenticationKey	The authentication key

Same as above with the addition of authentication support. You should use this alternative if you know your system implementation involves an authentication step, usually this means that LivePerson backend will verify the authentication token sent by the SDK with your system servers. If the key cannot be verified or your backend - this call will fail.

### **hideConversation**

<i>public static void hideConversation(Activity activity)</i>
---

activity	The calling activity
----------	----------------------

The *hideConversation* API hides the conversation Activity. The conversation screen is shown again by calling Start Conversation.

Note: Hiding the conversation closes the websocket.

Note: When using the SDK's Activity the back button performs the same function.

## getConversationFragment

```
public static Fragment getConversationFragment();
```

The *getConversationFragment* method creates and return the conversation fragment.

Note: this API does not show the actual screen, but only creates the fragment. Your implementation needs to handle when and how to show it.

## getConversationFragment with authentication support

```
public static Fragment getConversationFragment(String authKey)
```

authKey	The authentication key
---------	------------------------

Same as above with the attention of authentications support. You should use this alternative if you know your system implementation involves an authentication step, usually this means the LivePerson backend will verify the authentication token sent by the SDK with your system servers. If the key cannot be verified or your backend isn't setup with LivePerson backend - this call will fail.

## reconnect

```
public static void reconnect(String authKey)
```

authKey	The authentication key
---------	------------------------

Reconnect with a new authentication key.

When connecting with an authentication key, the connection may be closed once the token is expired. When this happens, the [onTokenExpired](#) callback method is called. In this case, the application needs to obtain a fresh key and reconnect by calling the *reconnect* method.

## setUserProfile

<i>public static void setUserProfile(String appId, String firstName, String lastName, String phone)</i>	
appId	The host app Id
firstName	User's first name
lastName	User's last name
phone	User's phone

The *setUserProfile* API takes custom parameters about the consumer as an input and sets it to be displayed on the messaging agent console consumer transcript. This can be set at any time either before, after, or during a messaging session.

## registerLPPusher

<i>public static void registerLPPusher(String brandId, String appId, String gcmToken)</i>	
brandId	The account Id (e.g. 652838922)
appId	The host app Id (e.g. com.liveperson.myApp)
gcmToken	The <a href="#">GCM Token</a> . Usually used to pass the Google provided token. However, this parameter can contain any string value.

Note: If you use the *gcmToken* as a custom value, you need to handle the mapping between this custom value and the actual gcm token in your server.

## unregisterLPPusher

<i>public static void unregisterLPPusher(String brandId, String appId)</i>	
brandId	The account Id
appId	The host app Id

Unregister from registered push notification service.

## handlePush

<i>public static void handlePush(Context context, Bundle data, String brandId, boolean showNotification)</i>	
context	A context from the host app
data	A Bundle that contains the message. The bundle should hold a string with key named "message"
brandId	The account Id
showNotification	Used to instruct the SDK to either show or not show a notification to the user. If you wish your app will handle the display of the notification you can set this as false.

All incoming push messages are received by the host app. The host app can choose to fully handle any push message and display a notification message or partially handle it and let the SDK to display the notification.

Handling the push message allows the host app to:

- Receive non-messaging related push messages.
- Handle custom in-app alerts upon an incoming message.

Note: Whether the host app fully handles any push messages or partially, any messaging push message should be sent to the SDK using the *handlePush* method.

## getSDKVersion

<i>public static String getSDKVersion()</i>
---

Returns the SDK version.

## setCallback

```
public static void setCallback(final LivePersonCallback listener)
```

listener	A <a href="#">LivePersonCallback</a> implementation
----------	---

Set SDK callback listener. The host app gets updates from the SDK using this callback listener. See [LivePerson Callbacks Interface](#) for more information.

## removeCallBack

```
public static void removeCallBack()
```

Removes the registered *LivePersonCallback* callback.

## checkActiveConversation

```
public static void checkActiveConversation(final ICallback<Boolean, Exception> callback)
```

callback	An <a href="#">ICallback</a> implementation
----------	---

This method checks whether there is an active (unresolved) conversation. The result will be returned to the provided callback.

## checkAgentID

```
public static void checkAgentID(final ICallback<AgentData, Exception> callback)
```

callback	An <a href="#">ICallback</a> implementation
----------	---

If there is an active conversation, this API returns agent data through the provided callback. If there is no active conversation, the API returns null.

[AgentData definition](#)



## markConversationAsUrgent

```
public static void markConversationAsUrgent()
```

Marks the current conversation as urgent.

## markConversationAsNormal

```
public static void markConversationAsNormal()
```

Marks the current conversation as normal.

## checkConversationIsMarkedAsUrgent

```
public static void checkConversationIsMarkedAsUrgent(final ICallback<Boolean, Exception> callback)
```

callback	An <a href="#">ICallback</a> implementation
----------	---

Checks whether the current conversation is marked as urgent. The result is returned through the provided callback.

## resolveConversation

```
public static void resolveConversation()
```

Resolves the current conversation.

## shutDown

```
public static void shutDown()
```

Shutting down the SDK and removing the footprint of the user session from local memory. After shutdown the SDK is unavailable until re-initiated. Message history is saved locally on the

device and synced with the server upon reconnection.

The server continues to send push notifications when the SDK is shut down. to unregister from push services call [unregisterLPPusher](#) API.

Note: This does not end the current messaging conversation.

Important: This method must not be called when the conversation screen is displayed.

## ClearHistory

```
public static boolean clearHistory()
```

Clear all conversations from the device. This clears all conversations and messages from the device only and does not removes them from the server, thus if the account has history enabled and is used on a new device all conversations will be loaded from server.

The return value indicates whether the action completed successfully or not:

*True* - all conversations cleared successfully.

*False* - conversations were not cleared since there is an open conversation.

Note: the *clearHistory* API call will work only if there is currently no active conversation.

## logout

```
public static void logOut(Context context, String brandId, String appId, LogoutLivePersonCallback  
logoutCallback){
```

context	A context from the host app
brandId	An account Id
appId	The host app Id
logoutCallback	An <a href="#">LogoutLivePersonCallback</a> implementation

Logout from the SDK - when all user data should be removed.

It's calling [unregisterLPPusher](#), [shutDown](#) and, in addition, deleting all user data (messages and user details) from the device.

In order to unregister from push it must be called when there is network available.

After logout the SDK is unavailable until re-initiated.

**This method does not require the SDK to be initialized.**

Note: This does not end the current messaging conversation.

Important: This method must not be called when the conversation screen is displayed.

## Interface and class definitions

### ICallback

```
public interface ICallback<T, E extends Throwable> {  
    void onSuccess(T value);  
    void onError(E exception);  
}
```

### InitLivePersonCallBack

```
public interface InitLivePersonCallBack {  
    void onInitSucceed();  
    void onInitFailed(Exception e);  
}
```

### AgentData

```
public class AgentData {  
  
    public String mFirstName;  
    public String mLastName;  
    public String mAvatarURL;  
    public String mEmployeeId;  
    public String mNickName;  
}
```

### LogoutLivePersonCallback

```
public interface LogoutLivePersonCallback{
```

```
void onLogoutSucceed();  
void onLogoutFailed();  
}
```

## InitLivePersonProperties

```
Public class InitLivePersonProperties{  
    Private string brandId;  
    Private string appld;  
    Private InitLivePersonCallBack initCallBack;  
}
```

## LivePerson Callbacks Interface

The SDK provides a callback mechanism to keep the host app updated on events related to the conversation. This section details each callback.

LivePersonCallback definition:

```
public interface LivePersonCallback{  
  
    void onError(TaskType type, String message);  
    void onTokenExpired();  
    void onConversationStarted();  
    void onConversationResolved();  
    void onConnectionChanged(boolean isConnected);  
    void onAgentTyping(boolean isTyping);  
    void onAgentDetailsChanged(AgentData agentData);  
    void onCsatDismissed();  
    void onConversationMarkedAsUrgent();  
    void onConversationMarkedAsNormal();  
    void onOfflineHoursChanges(boolean isOfflineHoursOn);  
}
```

```
enum TaskType {  
    CSDS,  
    IDP,  
    VERSION,  
    OPEN_SOCKET  
}
```

### Error indication

The *onError(TaskType type, String message)* method is called to indicate that an internal SDK error has occurred.

Parameters:

type - the type of the error

Message - a detailed message on the error

TaskType (defined above) indicate the category of the error as follows:

Type	Description
CSDS	Internal server error
IDP	An error occurred during the authentication process. This usually due to a wrong or expired authentication key
VERSION	Your host app is using an old SDK version and cannot be initialized.
OPEN_SOCKET	Error opening a socket to the server

### Token Expired

The *onTokenExpired()* method is called if the token used in the session has expired and no longer valid. The host app needs to [reconnect](#) with a new authentication key.

### Conversation started

The *onConversationStarted()* method is called whenever a new conversation is started by either the consumer or the agent.

### **Conversation resolved**

The *onConversationResolved()* method is called when the current conversation is marked as resolved by either the consumer or the agent.

### **Connection state has changed**

The *onConnectionChanged(boolean isConnected)* method is called when the connection to the conversation server has established or disconnected.

Parameters:

*isConnected* - indicates the connection state. *true* - connection establish, *false* - disconnected.

### **Agent details changed**

The *onAgentDetailsChanged([AgentData](#) agentData)* method is called when the assigned agent of the current conversation has changed or his details updated.

This callback is also called with null value when there is no agent that is associated with the conversation, for instance when the consumer is returned to queue. You need to check for null value before using the agentData object.

Parameters:

*agentData* - contains first name, last name, avatar url and employee Id.

### **Agent typing**

The *onAgentTyping(boolean isTyping)* method is called when the assigned agent is typing a message. When there is a 2 seconds of idle time, this method is getting called again to notify with *isTyping* false to indicate the agent stop typing.

### **CSAT Screen dismissed**

The *onCsatDismissed()* method is called when the feedback screen is dismissed (user pressed "Submit" button / user pressed on back button etc.)

### **Conversation marked as urgent**

The *onConversationMarkedAsUrgent()* is called when the current conversation is marked as urgent.

### **Conversation marked as normal**

The *onConversationMarkedAsNormal()* is called when the current conversation is marked as normal.

**Offline Hours Changes**

The `onOfflineHoursChanges(boolean isOfflineHoursOn)` is called when there is a change in the agent availability. When the agent is in off hours mode this method is getting called with `isOfflineHoursOn` true. When the agent return to online state `isOfflineHoursOn` it will be called with `isOfflineHoursOn` false.

# Configuring the SDK

The SDK allows you to configure the look and feel of the conversation screen with your branding.xml file.

In order to do so, you need to create, under the **values** folder, a new resource file called branding.xml. This file MUST contain all the exact resource-names as listed below:

## Brand

Resource Name	Description
<code>&lt;string name="brand_name"&gt;</code>	The brand name will be shown as a title on the toolbar when there is no active conversation.
<code>&lt;string name="language"&gt;</code>	The language is defined by a two-letter <a href="#">ISO 639-1</a> language code, for example, "en" for English. If no value is provided, the SDK will use the language according to the device's locale.
<code>&lt;string name="country"&gt;</code>	Country code. If no value is provided, the SDK will use the country according to the device's locale. For more information about language and country, click <a href="#">here</a> .
<code>&lt;integer name="message_receive_icons"&gt;</code>	For each message, there are three indicators available: Message sent, Message received, Message read. You can customize the indicators according to your needs, by using a number between 1 and 3: 0 - text (sent, delivered etc.) instead of icons 1 - Sent only 2 - Sent+received 3 - Sent+received+read
<code>&lt;string-array name="message_receive_text"&gt;</code>	If you set 0 in the resource message_receive_icons, you can specify what texts appears for each state. You must have 4 items, in the following order: 1 <sup>st</sup> item - message sent 2 <sup>nd</sup> item - message delivered 3 <sup>rd</sup> item - message read 4 <sup>th</sup> item - message not delivered
<code>&lt;string name="custom_button_icon_name"&gt;</code>	Custom button icon filename without extension. This will be displayed on the toolbar. onClick listener is available by implementing the callback



	listener using: LivePerson.setCallback (ILivePersonCallback:onCustomGuiTapped).
<pre>&lt;string name="custom_button_icon_description" &gt;</pre>	Content description for custom button. It briefly describes the view and is primarily used for accessibility support. Set this property to enable better accessibility support for your application
<pre>&lt;string name="clear_history_show_confirm_dialog" og"&gt;</pre>	Define if to show confirm dialog before clearing history or not. True by default.

### Brand Message Bubble - the first brand message

Resource Name	Description
<pre>&lt;dimen name="brand_bubble_stroke_width"&gt;</pre>	Int number for the outline width.
<pre>&lt;color name="brand_bubble_stroke_color"&gt;</pre>	Color code for the outline color.
<pre>&lt;color name="brand_bubble_message_text_color"&gt;</pre>	Color code for the text of the brand bubble.
<pre>&lt;color name="brand_bubble_message_link_text_color"&gt;</pre>	Color code for links in the text of the brand bubble.
<pre>&lt;color name="brand_bubble_timestamp_text_color"&gt;</pre>	Color code for the timestamp of the brand bubble.
<pre>&lt;color name="brand_bubble_background_color"&gt;</pre>	Color code for the background of the brand bubble.
<pre>&lt;color name="brand_logo_background_color"&gt;</pre>	Color code for the background of the default brand logo next to the bubble.

### Agent Message Bubble

Resource Name	Description
<pre>&lt;dimen name="agent_bubble_stroke_width"&gt;</pre>	Int number for the outline width.
<pre>&lt;color name="agent_bubble_stroke_color"&gt;</pre>	Color code for the outline color.
<pre>&lt;color name="agent_bubble_message_text_color"&gt;</pre>	Color code for the text of the agent bubble.
<pre>&lt;color name="agent_bubble_message_link_text_color"&gt;</pre>	Color code for links in the text of the agent bubble.
<pre>&lt;color name="agent_bubble_timestamp_text_color"&gt;</pre>	Color code for the timestamp of the

	agent bubble.
<code>&lt;color name="agent_bubble_background_color"&gt;</code>	Color code for the background of the agent bubble.
<code>&lt;color name="agent_avatar_background_color"&gt;</code>	Color code for the background of the agent default avatar next to the bubble.
<code>&lt;color name="agent_avatar_icon_color"&gt;</code>	Color code for the agent default icon in the avatar next to the bubble.

## Visitor Message Bubble

Resource Name	Description
<code>&lt;color name="consumer_bubble_message_text_color"&gt;</code>	Color code for the text of the consumer bubble.
<code>&lt;color name="consumer_bubble_message_link_text_color"&gt;</code>	Color code for links in the text of the consumer bubble.
<code>&lt;color name="consumer_bubble_timestamp_text_color"&gt;</code>	Color code for the timestamp of the consumer bubble.
<code>&lt;color name="consumer_bubble_background_color"&gt;</code>	Color code for the background of the consumer bubble.
<code>&lt;color name="consumer_bubble_state_text_color"&gt;</code>	Color code for state text next to the consumer bubble.
<code>&lt;dimen name="consumer_bubble_stroke_width"&gt;</code>	integer in dp for the bubble stroke width of the consumer bubble.
<code>&lt;color name="consumer_bubble_stroke_color"&gt;</code>	Color code for the stroke of the consumer bubble.

## System messages

Resource Name	Description
<code>&lt;color name="system_bubble_text_color"&gt;</code>	Color code for the text of the system messages.

## Survey screen

Resource Name	Description
<code>&lt;color name="feedback_fragment_background_color"&gt;</code>	Feedback dialog background color
<code>&lt;color name="feedback_fragment_title_question"&gt;</code>	Feedback dialog title color

<code>&lt;color name="feedback_fragment_star"&gt;</code>	Feedback dialog star color
<code>&lt;color name="feedback_fragment_rate_text"&gt;</code>	Feedback dialog rating title color
<code>&lt;color name="feedback_fragment_title_yesno"&gt;</code>	Feedback dialog yes/no color
<code>&lt;color name="feedback_fragment_yesno_btn_selected_background"&gt;</code>	Feedback dialog yes/no selected background color
<code>&lt;color name="feedback_fragment_yesno_btn_default_background"&gt;</code>	Feedback dialog yes/no default background
<code>&lt;color name="feedback_fragment_yesno_btn_text_selected"&gt;</code>	Feedback dialog yes/no text color when selected
<code>&lt;color name="feedback_fragment_yesno_btn_text_default"&gt;</code>	Feedback dialog yes/no text color when in default
<code>&lt;color name="feedback_fragment_yesno_btn_stroke_default"&gt;</code>	Feedback dialog yes/no stroke color when in default
<code>&lt;color name="feedback_fragment_yesno_btn_stroke_selected"&gt;</code>	Feedback dialog yes/no stroke color when selected
<code>&lt;dimen name="feedback_fragment_yesno_btn_stroke_width_default"&gt;</code>	Feedback dialog yes/no stroke width size when in default
<code>&lt;dimen name="feedback_fragment_yesno_btn_stroke_width_selected"&gt;</code>	Feedback dialog yes/no stroke width size when in selected
<code>&lt;color name="feedback_fragment_submit_message"&gt;</code>	Feedback dialog submit message text color
<code>&lt;color name="feedback_fragment_submit_btn_enabled"&gt;</code>	Feedback dialog submit button color when enabled
<code>&lt;color name="feedback_fragment_submit_btn_text_enabled"&gt;</code>	Feedback dialog submit button text color when enabled
<code>&lt;color name="feedback_fragment_submit_btn_disabled"&gt;</code>	Feedback dialog submit button color when disabled
<code>&lt;color name="feedback_fragment_submit_btn_text_disabled"&gt;</code>	Feedback dialog submit button text color when disabled

<code>led"&gt;</code>	
<code>&lt;color name="feedback_fragment_submit_btn_stroke_enabled"&gt;</code>	Feedback dialog submit button stroke color when enabled
<code>&lt;color name="feedback_fragment_submit_btn_stroke_disabled"&gt;</code>	Feedback dialog submit button stroke color when disabled
<code>&lt;dimen name="feedback_fragment_submit_btn_stroke_width_enabled"&gt;</code>	Feedback dialog submit button stroke width size when enabled
<code>&lt;dimen name="feedback_fragment_submit_btn_stroke_width_disabled"&gt;</code>	Feedback dialog submit button stroke width size when disabled
<code>&lt;bool name="show_yes_no_question"&gt;</code>	Defines whether to show or hide the yes/no question in the feedback dialog (true=show, false=hide)
<code>&lt;bool name="show_feedback"&gt;</code>	Defines whether to show the feedback dialog
<code>&lt;string name="default_agent_name"&gt;</code>	The default agent name to display in the toolbar and in the feedback dialog when the assigned agent does not have a nickname defined.

## Message Edit Text

Resource Name	Description
<code>&lt;color name="edit_text_underline_color"&gt;</code>	Color code for the Enter Message control underline color
<code>&lt;color name="lp_enter_msg_text"&gt;</code>	Define the input message text color
<code>&lt;color name="lp_enter_msg_hint"&gt;</code>	Define the input message hint color
<code>&lt;color name="lp_send_button_text_enable"&gt;</code>	Define the color of the send button when it's enable.
<code>&lt;color name="lp_send_button_text_disable"&gt;</code>	Define the color of the send button when it's disabled.

## General Style

Resource Name	Description
<code>&lt;color name="conversation_background"&gt;</code>	Define the color code for the entire view background In activity mode - it also the color of

	android:windowBackground
--	--------------------------

## Conversation Activity Style - (activity mode only!)

Resource Name	Description
<code>&lt;color name="lp_colorPrimary"&gt;</code>	Define the primary color of the activity (android:colorPrimary)
<code>&lt;color name="lp_colorPrimaryDark"&gt;</code>	Define the primary dark color of the activity (android:colorPrimaryDark)

## Miscellaneous

Resource Name	Description
<code>&lt;bool name="disableTTRPopup"&gt;</code>	Defines whether to disable the TTR snackbar popup (true=disable>false by default
<code>&lt;bool name="contextual_menu_on_toolbar"&gt;</code>	Enable multiple message copy menu over the app toolbar. If <i>true</i> , when long pressing a message on chat it will select the message and enable a context menu over the toolbar enabling the user to copy multiple messages. If <i>false</i> , long pressing a message will display a copy popup menu.
<code>&lt;color name="bubble_selected_background_color"&gt;</code>	Define the background color of item when it's selected to be copied. (If multiple message copy is enable)
<code>&lt;string name="notification_large_icon_name"&gt;</code>	The name of an resource to use as the large icon of the push notification (used only when using <a href="#">handlePush</a> with <i>showNotification=true</i> )
<code>&lt;integer name="encryptionVersion"&gt;</code>	Defines the encryption version to use. Currently available version 1 only. 1 - encrypt data 0 - disable encryption
<code>&lt;string name="csds_url"&gt;</code>	For vanity URL purposes. For regular uses please use: <b><i>adminlogin.liveperson.net</i></b>
<code>&lt;integer name="idp_num_history_conversation"&gt;</code>	When user is authenticated, this indicates the number of recent conversations to reload from server (including their messages) when running for the first time.

<code>&lt;bool name="show_timestamp_in_ttr_notification"&gt;</code>	When <i>true</i> the TTR snackbar will display the time until the agent responds. If set to <i>false</i> a general message is displayed.
<code>&lt;bool name="show_urgent_button_in_ttr_notification"&gt;</code>	When <i>true</i> the TTR snackbar will include a “mark as urgent” button.
<code>&lt;integer name="ttr_duration"&gt;</code>	Set the duration that the TTR snackbar will be visible.
<code>&lt;bool name="send_agent_profile_updates_when_conversation_closed"&gt;</code>	When <i>true</i> the callback <code>LivePersonCallback#onAgentDetailsChanged</code> will be called with the agent details updates even if the last conversation is closed - (In that case - it'll provide the assigned agent of the last conversation). If <i>false</i> this callback will be called only when the current conversation is active. True by default
<code>&lt;string name="custom_button_icon_name"&gt;</code>	Deprecated
<code>&lt;bool name="ttr_message_off_hours_enabled"&gt;</code>	Defines whether to show the off hours snackbar popup (true=enable)
<code>&lt;integer name="ttrShowFrequencyInSeconds"&gt;</code>	Define the frequency of the TTR (time to response) messages.
<code>&lt;bool name="enable_client_only_masking"&gt;</code>	Defines whether to enable or disable client side only masking. False by default.
<code>&lt;bool name="enable_real_time_masking"&gt;</code>	Defines whether to enable or disable real time masking. False by default.
<code>&lt;string name="client_only_masking_regex"&gt;</code>	Defines the java regex for client side only masking. By default does not contain any value
<code>&lt;string name="real_time_masking_regex"&gt;</code>	Defines the java regex for real time masking. By default does not contain any value

**Note:** There is an option to change the whole style of the message EditText. In the app's styles.xml file, override the `lp_enter_message_style` with the required style. For example:

```

<style name="lp_enter_message_style" parent="Theme.AppCompat.Light.NoActionBar">
<item name="colorControlActivated">#F8BBD0</item>
...
</style>

```

## ProGuard Configuration

The SDK handles his own and all his dependencies ProGuard rules, there is no need to add any ProGuard specific rules that related to the SDK.

The SDK ProGuard will run automatically when the ProGuard option is enabled in the gradle file of your application.

In case there is no ProGuard activated, the SDK ProGuard will be disabled also.

## Modifying strings

You may change every string appearing on the SDK interface by overriding the respective string key.

String name	Used in	Default value
lp_enter_message	Enter message text box when empty	Enter a message...
lp_send	The "Send" button text	Send
lp_no_network_toast_message	A toast message when there is not network	There is no connectivity. Please connect to the internet and try again
lp_no_action_not_available_toast_message	A toast message when the required action is not available (e.g. Mark as urgent when there is no active conversation)	Action not available - no open conversation

lp_today	Today header in conversation	TODAY
lp_yesterday	Yesterday header in conversation	YESTERDAY
lp_tomorrow	Used in the off-hours message	TOMORROW
lp_conversation_ended_msg	Message when the conversation was resolved	Conversation resolved by %1\$s \n %2\$s
lp_conversation_resume_msg	Message when the conversation was resumed by the agent	Conversation was resumed by %1\$s \n %2\$s
lp_first_message	System message before the first conversation	How can I help you today?
lp_loading_message	Text above the loading icon when loading previous messages	Loading...
lp_default_user_name	Name in system message in conversation when the consumer resolved the conversation.	You
lp_default_agent_name	Name in system message in conversation when the agent resolved the conversation.	Agent
lp_is_typing	Text in conversation activity when agent is typing	typing...
lp_mark_as_urgent_menu_text	“Mark as urgent” string in menu and snack bar	Mark as Urgent
lp_mark_as_resolved_menu_text	“Mark as resolved” string in menu	Mark as resolved
lp_clear_history_menu_text	“Clear history” string in menu	Clear history
lp_clear_history_confirm_string	“Clear history” confirmation dialog text	All of your existing conversation history will be lost. Are you sure?



lp_clear	“Clear” button text on “Clear history” dialog	Clear
lp_end_conversation_first	Dialog text that is shown in case trying to clear history when a conversation is open	Please end the conversation first
lp_dismiss_as_urgent_two_lines	“Dismiss urgent” string in menu and snack bar	Dismiss Urgent
lp_mark_as_urgent_dialog_header	Mark as urgent confirmation dialog header	Are you sure you want to mark conversation as urgent?
lp_dismiss_urgent_dialog_header	Dismiss urgent confirmation dialog header	Are you sure you want to dismiss the urgent state?
lp_dismiss_as_urgent_menu_text	Dismiss urgent menu text	Dismiss Urgent
lp_mark_as_resolved_dialog_message	Resolve conversation confirmation dialog text	Are you sure you want to resolve this conversation?
lp_mark_as_urgent_dialog_message	Mark as urgent confirmation dialog text	This means that your conversation will get top priority.
lp_dismiss_urgent_dialog_message	Dismiss urgent confirmation dialog text	This means that your conversation will get normal priority.
lp_ttr_message_with_timestamp	Text in TTR snackbar when timestamp is shown	The agent will respond within the next
lp_ttr_message_minutes	(plurals string that contains: “one” and “others”) The <i>one</i> or <i>others</i> strings is concatenated to the <i>lp_ttr_message_with_timestamp</i> string above according to whether it’s single <b>minute</b> multiple <b>minutes</b>	
	one	%1\$s minute
	others	%1\$s minutes
lp_ttr_message_hours	(plurals string that contains: “one” and “others”)	

	The <i>one</i> or <i>others</i> strings is concatenated to the <i>lp_ttr_message_with_timestamp</i> string above according to whether it's single <b>hour</b> multiple <b>hours</b>	
	one	%1\$s hour
	others	%1\$s hours
lp_ttr_message_days	(plurals string that contains: "one" and "others") The <i>one</i> or <i>others</i> strings is concatenated to the <i>lp_ttr_message_with_timestamp</i> string above according to whether it's single <b>day</b> multiple <b>days</b>	
	one	%1\$s day
	others	%1\$s days
lp_ttr_message_no_timestamp	Text in TTR snackbar when timestamp is not shown	An agent will respond shortly
lp_toolbar_menu_description	contentDescription of the conversation activity toolbar menu	menu
lp_feedback_1	String displayed when one star is selected in the feedback dialog	Very Dissatisfied
lp_feedback_2	String displayed when two star are selected in the feedback dialog	Dissatisfied
lp_feedback_3	String displayed when three star are selected in the feedback dialog	Neither
lp_feedback_4	String displayed when four star are selected in the feedback dialog	Satisfied
lp_feedback_5	String displayed when five star are selected in the	Very Satisfied

	feedback dialog	
lp_feedback_thank_you	Text displayed after the feedback dialog is submitted	Survey submitted successfully.\nThank you!
lp_feedback_submit	The feedback submit button text	Submit
lp_feedback_rate_title	Default feedback dialog text	Overall, how satisfied are you with the experience?
lp_feedback_yesno_question	Yes/No question text in feedback dialog	Did we solve your issue today?
lp_feedback_submit_message	Submit message text at the bottom of feedback dialog	Your feedback helps us serve you better.\n It will not be shared with any customer service representative.
lp_feedback_yesno_negative_title	Negative button text in the feedback dialog	NO
lp_feedback_yesno_positive_title	Positive button text in the feedback dialog	YES
lp_feedback_question	Feedback dialog text with agent name	How would you rate \nthe connection with %1\$s?"
lp_end	End conversation "End" button text	End
lp_skip	Feedback dialog toolbar skip button text	Skip
lp_done	Feedback dialog toolbar done button text (after submitting)	Done
lp_ok	Confirmation dialog OK button	OK
lp_cancel	Confirmation dialog Cancel button	Cancel
lp_menu_copy	Copy menu button text when selecting messages in conversation	Copy
lp_end_conversation	End conversation title	End Conversation

lp_resend_failed_conversation_closed	Toast message displayed when trying to resend a failed message when conversation is already closed	This conversation has already ended.
lp_resend_failed_masked_message	Toast message displayed when trying to resend a failed masked message	Message failed to send. Please re-enter message and send again.
lp_new_messages	Notification message displayed when there are multiple push messages	new messages
lp_message_time_now	Message timestamp for the latest messages ("Now")	Now
lp_message_time_now_with_state	Message timestamp for the latest messages that has a sending state ("now")	now
lp_message_time_min_ago	Message timestamp for older messages ("5 min ago")	Min ago
lp_ttr_message_off_hours_time_zone_id	Represent java timezone id that is used in the off hours message. For a full list of the available ids- use the "Aliases" from <a href="#">here</a> .	US/Pacific
lp_ttr_message_off_hours_message_today_tomorrow	Message to show when the online hours are later today or tomorrow.  Includes 2 params: %1\$s - for today/tomorrow %2\$s - for the time	Thanks for your message. We will be back online %1\$s (today/tomorrow) at %2\$s
lp_ttr_message_off_hours_message	Message to show when the online hours is more	Thanks for your message. We will be back online at %1\$s

	<p>than 2 days from now.</p> <p>includes 1 param: %1\$s - for the full date (MMM dd, yyyy hh:mm a)</p>	
lp_system_message_real_time_masked	Text of system message, added after detecting a real time masked message (if this feature is enabled)	Your personal data has been masked to protect your security and cannot be read by the agent.
real_time_mask_character	The character used to mask real time message	*
lp_system_message_client_only_masked	Text of system message, added after detecting a client only masked message (if this feature is enabled)	Your personal data has been masked to protect your security but can be read by the agent.
client_only_mask_character	The character used to mask client only string	*

## Modifying resources

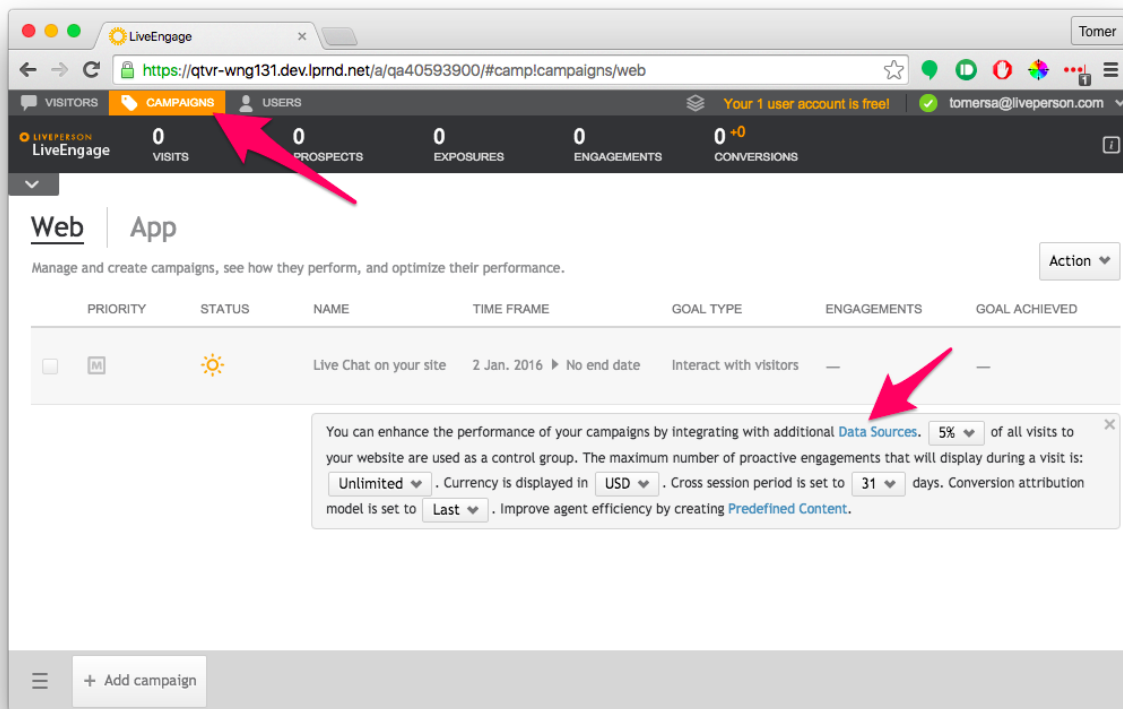
The SDK utilizes several resources as part of its GUI. To customize those resources please add appropriate resources on your project:

Description	Resources name	Size
Default brand avatar on the avatar next to brand bubble (the first brand message) and on agent avatar appearing on the action bar before an agent is assigned.	lp_messaging_ui_brand_logo	

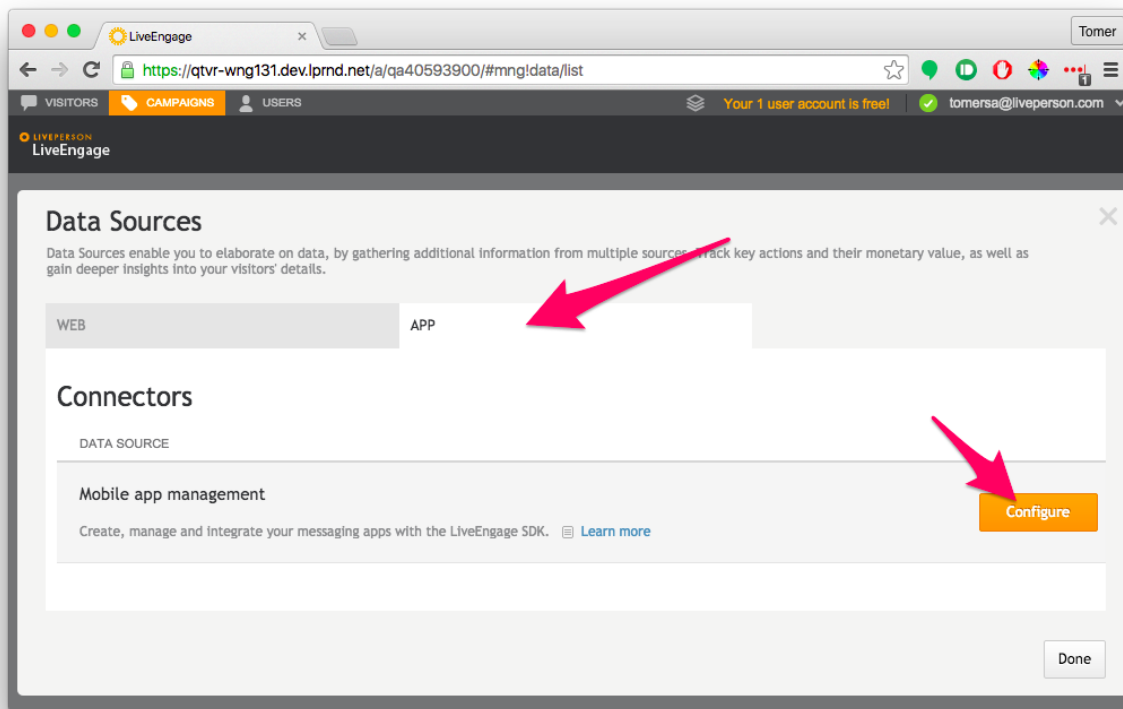
<p>In case you want to define the background color for this avatar - override "brand_logo_background_color" resource id. (this is relevant only for bubble brand's avatar only. Background color of agent avatar on action bar is "agent_avatar_background_color"</p>		
<p>Default agent avatar appearing next to an agent's bubble when no avatar URL is assigned on LiveEngage and on agent avatar appearing on the action bar.</p> <p>In case you want to define the background color for this avatar - override "agent_avatar_background_color" resource id.</p>	lp_messaging_ui_ic_agent_avatar	

## Set up your app key to enable push notifications

Log into your LiveEngage account using an administrator's credentials and navigate to Campaigns->Data Sources:

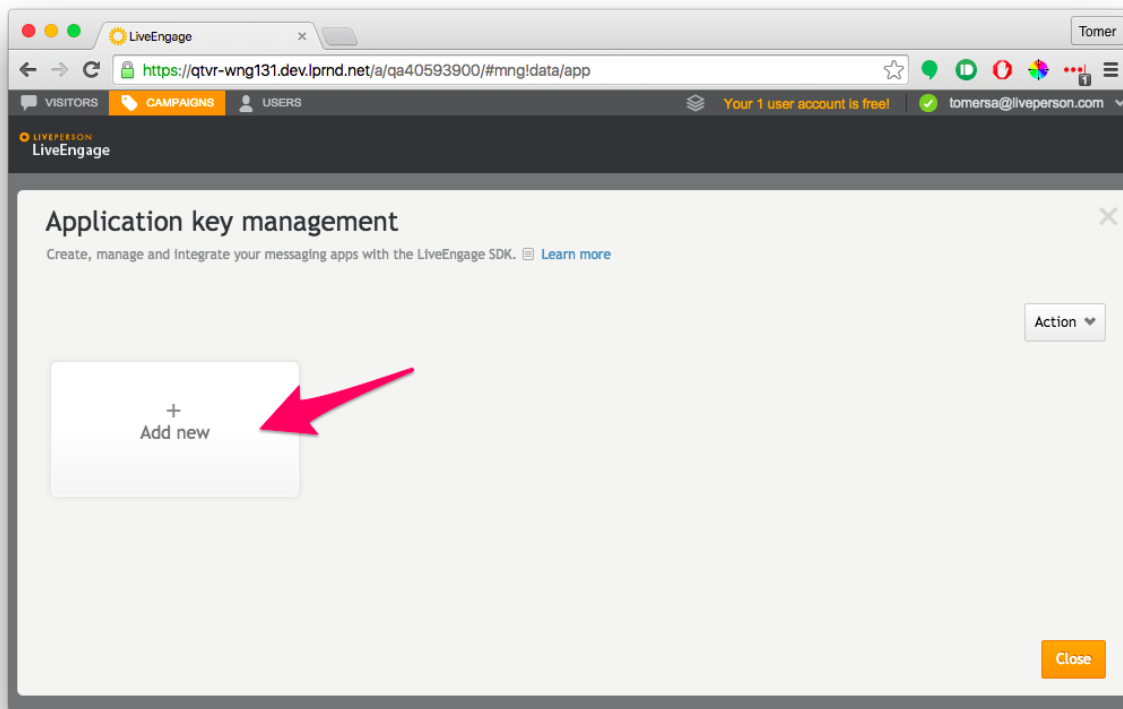


Then, select the “App” section and click “Configure”:

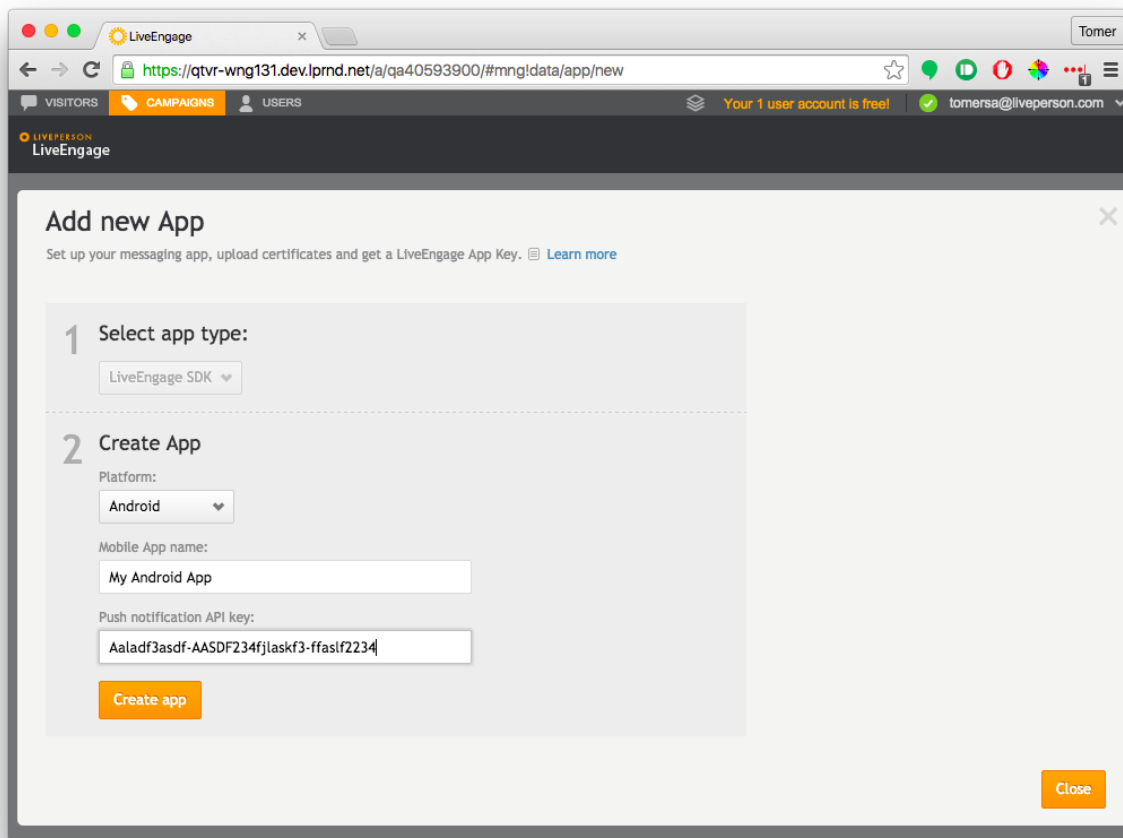


Now, click on the “Add new” button to create associate your app with the LiveEngage account:





Select your platform as Android, enter your app's name and your [push notification API key](#) in the appropriate fields. Then click "Create App":



The screenshot shows a web browser window with the LiveEngage interface. The main content area is titled "Add new App" and includes a sub-header: "Set up your messaging app, upload certificates and get a LiveEngage App Key. [Learn more](#)". The form is divided into two steps:

- 1 Select app type:** A dropdown menu is set to "LiveEngage SDK".
- 2 Create App:** This section contains three input fields:
  - Platform:** A dropdown menu set to "Android".
  - Mobile App name:** A text input field containing "My Android App".
  - Push notification API key:** A text input field containing "Aaladf3asdf-AASDF234fjlaskf3-ffaslf2234".

Below the input fields is an orange "Create app" button. In the bottom right corner of the form, there is an orange "Close" button.

Click "Close" to finish the process.

# Dependencies

**com.squareup.okhttp3:okhttp:3.2.0**

An HTTP+HTTP/2 client for Android and Java applications

**com.neovisionaries:nv-websocket-client:1.26**

High-quality WebSocket client implementation in Java.

**com.facebook.fresco:fresco:0.9.0**

An Android library for managing images and the memory they use.

## Open source list

Name	Site	Licence
Fresco	<a href="http://frescolib.org/">http://frescolib.org/</a>	<a href="https://github.com/facebook/fresco/blob/master/LICENSE">https://github.com/facebook/fresco/blob/master/LICENSE</a>
OKHTTP	<a href="http://square.github.io/okhttp/">http://square.github.io/okhttp/</a>	<a href="https://github.com/square/okhttp/blob/master/LICENSE.txt">https://github.com/square/okhttp/blob/master/LICENSE.txt</a>
nv-websocket-client	<a href="https://github.com/TakahikoKawasaki/nv-websocket-client">https://github.com/TakahikoKawasaki/nv-websocket-client</a>	<a href="https://github.com/TakahikoKawasaki/nv-websocket-client/blob/master/LICENSE">https://github.com/TakahikoKawasaki/nv-websocket-client/blob/master/LICENSE</a>

This document, materials or presentation, whether offered online or presented in hard copy ("LivePerson Informational Tools") is for informational purposes only. LIVEPERSON, INC. PROVIDES THESE LIVEPERSON INFORMATIONAL TOOLS "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

The LivePerson Informational Tools contain LivePerson proprietary and confidential materials. No part of the LivePerson Informational Tools may be modified, altered, reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), without the prior written permission of LivePerson, Inc., except as otherwise permitted by law. Prior to publication, reasonable effort was made to validate this information. The LivePerson Information Tools may include technical inaccuracies or typographical errors. Actual savings or results achieved may be different from those outlined in the LivePerson Informational Tools. The recipient shall not alter or remove any part of this statement.

Trademarks or service marks of LivePerson may not be used in any manner without LivePerson's express written consent. All other company and product names mentioned are used only for identification purposes and may be trademarks or registered trademarks of their respective companies. LivePerson shall not be liable for any direct, indirect, incidental, special, consequential or exemplary damages, including but not limited to, damages for loss of profits, goodwill, use, data or other intangible losses resulting from the use or the inability to use the LivePerson Information Tools, including any information contained herein.

© 2015 LivePerson, Inc. All rights reserved.