LiveEngage Enterprise In-App Messenger SDK: Android

Deployment Guide

Version 1.4.0

September 2016

Quick Start

The LivePerson SDK provides brands with a simple, yet enterprise-grade and secure in-app messaging solution. Through in-app messaging, brands will foster connections with their customers and increase app engagement and retention.

This Quick Start will quickly get you up and running with a project powered by LivePerson. When you're done, you'll be able to send messages between an Android device and LiveEngage. To complete this Quick Start, you will need a LiveEngage account. You can get the number and login information from the LivePerson account team.

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

Step 1: Download and unzip SDK

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:
● LP_Messaging_SDK/lp_messaging_sdk - Module that should be added to your project. This module contains the following:
   ○ LivePerson.java - Main entry point for the Messaging SDK
   ○ Resources (.aars files)
● SampleApp-Source - demonstrate how to use the Messaging SDK.
● SampleApp-APK - sample app installation file.

Step 2: Create a 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 the following lines to the build.gradle of your app :
   a. *compileSdkVersion* and *buildToolsVersion* (should be at least 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 displays the agent name the consumer is talking with. The 'Is Typing' indicator displays 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 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 such as 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 displayed 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);
  }
}
```

Step 3: Test 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 Workspace](#) and log in.
2. Run your app
3. Click the button that calls to start the conversation (from section [4.b](#)).
4. From your app's conversation view, send a message.
5. Make sure the message is received in the Agent Workspace.

Advanced Configurations

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
Initialize (Deprecated)	Initialize the resources required by the SDK
Initialize with SDK properties object	Initialize the resources required by the SDK with properties object
showConversation	Display the messaging activity
showConversation with authentication support	Display the messaging activity with the addition of authentication support
hideConversation	Hide the conversation activity
getConversationFragment	Get the conversation fragment
getConversationFragment with authentication support	Get the conversation fragment with the addition of authentication support
reconnect	Reconnect with new authentication key
setUserProfile	Take custom parameters about the consumer as an input, set them for the messaging agent, and attach them to the transcript
registerLPPusher	Register to LivePerson push services
unregisterLPPusher	Unregister from LivePerson push services
handlePush	Receive all incoming push messages in a single function
getSDKVersion	Return the SDK version
setCallback	Get events from SDK - need to implement **LivePersonCallback**

removeCallBack	Stop getting events from the SDK
checkActiveConversation	Check whether there is an active conversation
checkAgentID	Return agent data such as, first name, last name, email, avatarURL, through callback
markConversationAsUrgent	Mark the current conversation as urgent
markConversationAsNormal	Mark the current conversation as normal
checkConversationIsMarkedAsUrgent	Check whether the current conversation is marked as urgent
resolveConversation	Resolve the current conversation
shutDown	Shut down the SDK
shutDown	Shut down the SDK (deprecated)
clearHistory	Clear all conversations from device
logOut	Logout from the SDK - when all user data should be removed

Initialize (Deprecated)

public static void initialize (Context context, String brandId, InitLivePersonCallBack initCallBack)	
context	A context from the host app
brandId	An account ID
initCallBack	An InitLivePersonCallBack 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

public static void initialize (Context context, InitLivePersonProperties initProperties)	
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

public static boolean showConversation(Activity activity)	
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

public static boolean showConversation(Activity activity, String authenticationKey)	
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 the LivePerson backend will verify the authentication token sent by the SDK with your system servers. If the key cannot be verified on your company's backend servers, this call will fail.

hideConversation

public static void hideConversation(Activity activity)	
activity	The calling activity

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

Notes:
Hiding the conversation closes the websocket.
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 authentication 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 set up with the 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

public static void setUserProfile(String appId, String firstName, String lastName, String phone)	
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 Workspace consumer transcript. This can be set at any time either before, after, or during a messaging session.

registerLPPusher

public static void registerLPPusher(String brandId, String appId, String gcmToken)	
brandId	The account Id (e.g. 652838922)
appId	The host app Id (e.g. com.liveperson.myApp)
gcmToken	The GCM Token. 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

public static void unregisterLPPusher(String brandId, String appId)	
brandId	The account ID
appId	The host app ID

Unregister from registered push notification service.

handlePush

public static void handlePush(Context context, Bundle data, String brandId, boolean showNotification)	
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 allow the SDK to display the notification.

Handling the push message allows the host app to do the following:
· 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

public static String getSDKVersion()

Returns the SDK version.

setCallback

public static void setCallback(final LivePersonCallback listener)	
listener	A LivePersonCallback implementation

Sets the 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 ICallback 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 ICallback 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 ICallback 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(final ShutDownLivePersonCallback shutdownCallback)	
shutdownCallback	A *ShutDownLivePersonCallback* implementation to get indication whether the shutdown succeeded or failed

Shuts down the SDK and removes 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.

ShutDownLivePersonCallback callback description:
- *onShutdownSucceed()* method is called when the shutdown process finished successfully.
- *onShutdownFailed()* method is called when the shutdown process failed (for example, shutdown was called when the conversation screen is displayed in the foreground).

Note: This does not end the current messaging conversation.

shutDown

(Deprecated. Please use the above *shutDown(ShutDownLivePersonCallback)* method)

public static void shutDown()

Shuts down the SDK and removes 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 remove them from the server. If the account has history enabled and is used on a new device, all conversations will be loaded from the server.

The return value indicates whether the action was completed successfully or not:
True - All conversations were 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 *LogoutLivePersonCallback* implementation

Logout from the SDK - when all user data should be removed.
Calls *unregisterLPPusher*, shutDown and, in addition, deletes 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);
}
```

ShutDownLivePersonCallback

```
public interface ShutDownLivePersonCallback {
      void onShutdownSucceed();
      void onShutdownFailed();
}
```

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 appId;
    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 onCsatSubmitted(String conversationId);
    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 error
Message - A detailed message on the error

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

Type	Description
CSDS	Internal server error.
IDP	An error occurred during the authentication process. This is 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 their details are 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 2 seconds of idle time, this method is called again to notify with isTyping false to indicate that the agent stopped typing.

CSAT Screen dismissed

The *onCsatDismissed()* method is called when the feedback screen is dismissed (user clicked Submit button, user clicked Back button, etc.).

CSAT Screen submitted

The *onCsatSubmitted(String conversationId)* method is called when the user clicked the Submit button on the feedback screen.
conversationId - The id of the conversation the survey is related to.
This callback comes in addition to the *onCsatDismissed* callback when clicking Submit .

Conversation marked as urgent

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

Conversation marked as normal

The onConversationMarkedAsNormal() method 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 agent availability. When the agent is in off hours mode this method is called with isOfflineHoursOn true. When the agent return to online state, isOfflineHoursOn is 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 resource-names as listed below:

Brand

Resource Name	Description
`<string name="brand_name">`	The brand name will be shown as a title on the toolbar when there is no active conversation.
`<string name="language">`	The language is defined by a two-letter ISO 639-1 language code, for example, "en" for English. If no value is provided, the SDK will use the language according to the device's locale.
`<string name="country">`	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 here.
`<integer name="message_receive_icons">`	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
`<string-array name="message_receive_text">`	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: 1st item - message sent

	2nd item - message delivered 3rd item - message read 4th item - message not delivered
`<string` `name="custom_button_icon_name">`	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).

`<string`
`name="custom_button_icon_description"`
`>`
	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.

`<string`
`name="clear_history_show_confirm_dial`
`og">`
	Define if to show confirm dialog before clearing history or not.
	True by default.

Brand Message Bubble - the first brand message

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

Agent Message Bubble

Resource Name	Description
`<dimen name="agent_bubble_stroke_width">`	Int number for the outline width.
`<color name="agent_bubble_stroke_color">`	Color code for the outline color.
`<color name="agent_bubble_message_text_color">`	Color code for the text of the agent bubble.

Resource Name	Description
`<color name="agent_bubble_message_link_text_color">`	Color code for links in the text of the agent bubble.
`<color name="agent_bubble_timestamp_text_color">`	Color code for the timestamp of the agent bubble.
`<color name="agent_bubble_background_color">`	Color code for the background of the agent bubble.
`<color name="agent_avatar_background_color">`	Color code for the background of the agent default avatar next to the bubble.
`<color name="agent_avatar_icon_color">`	Color code for the agent default icon in the avatar next to the bubble.

Visitor Message Bubble

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

System messages

Resource Name	Description
`<color name="system_bubble_text_color">`	Color code for the text of the system messages.

Survey screen

Resource Name	Description
`<color name="feedback_fragment_background_color">`	Feedback dialog background color.

`<color name="feedback_fragment_title_question">`	Feedback dialog title color.
`<color name="feedback_fragment_star">`	Feedback dialog star color.
`<color name="feedback_fragment_rate_text">`	Feedback dialog rating title color.
`<color name="feedback_fragment_title_yesno">`	Feedback dialog yes/no color.
`<color name="feedback_fragment_yesno_btn_selected_background">`	Feedback dialog yes/no selected background color.
`<color name="feedback_fragment_yesno_btn_default_background">`	Feedback dialog yes/no default background.
`<color name="feedback_fragment_yesno_btn_text_selected">`	Feedback dialog yes/no text color when selected.
`<color name="feedback_fragment_yesno_btn_text_default">`	Feedback dialog yes/no text color when in default.
`<color name="feedback_fragment_yesno_btn_stroke_default">`	Feedback dialog yes/no stroke color when in default.
`<color name="feedback_fragment_yesno_btn_stroke_selected">`	Feedback dialog yes/no stroke color when selected.
`<dimen name="feedback_fragment_yesno_btn_stroke_width_default">`	Feedback dialog yes/no stroke width size when in default.
`<dimen name="feedback_fragment_yesno_btn_stroke_width_selected">`	Feedback dialog yes/no stroke width size when in selected.
`<color name="feedback_fragment_submit_message">`	Feedback dialog submit message text color.
`<color name="feedback_fragment_submit_btn_enabled">`	Feedback dialog submit button color when enabled.
`<color name="feedback_fragment_submit_btn_text_enabled">`	Feedback dialog submit button text color when enabled.
`<color`	Feedback dialog submit button color

`name="feedback_fragment_submit_btn_disabled">`	when disabled.
`<color name="feedback_fragment_submit_btn_text_disabled">`	Feedback dialog submit button text color when disabled.
`<color name="feedback_fragment_submit_btn_stroke_enabled">`	Feedback dialog submit button stroke color when enabled.
`<color name="feedback_fragment_submit_btn_stroke_disabled">`	Feedback dialog submit button stroke color when disabled.
`<dimen name="feedback_fragment_submit_btn_stroke_width_enabled">`	Feedback dialog submit button stroke width size when enabled.
`<dimen name="feedback_fragment_submit_btn_stroke_width_disabled">`	Feedback dialog submit button stroke width size when disabled.
`<color name="feedback_fragment_agent_details_name">`	Define the color of the agent name on agent details section in feedback dialog. Visible only if `show_agent_details_csat` is true.
`<string name="default_agent_name">`	The default agent name to display in the toolbar and in the feedback dialog when the assigned agent does not have a nickname defined.
`<bool name="show_feedback">`	Defines whether to show the feedback dialog.
`<bool name="show_agent_details_csat">`	Define if the agent's name and avatar are visible on top of feedback dialog. (true=show, false=hide) **NOTE: if both `show_yes_no_question` and `show_agent_details_csat` are set to true, `show_yes_no_question` will be ignored and will not be visible.**
`<bool name="show_yes_no_question">`	Defines whether to show or hide the yes/no question in the feedback dialog (true=show, false=hide) **NOTE: if both `show_yes_no_question` and `show_agent_details_csat` are set to true, `show_yes_no_question` will be ignored and will not be visible.**
`<bool name="show_csat_thank_you">`	Define if "thank you" screen will appear

	after submitting the survey. (true=show, false=hide)

Message Edit Text

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

General Style

Resource Name	Description
`<color name="conversation_background">`	Define the color code for the entire view background. In activity mode - Also the color of android:windowBackground

Conversation Activity Style - (activity mode only!)

Resource Name	Description
`<color name="lp_colorPrimary">`	Define the primary color of the activity (android:colorPrimary).
`<color name="lp_colorPrimaryDark">`	Define the primary dark color of the activity (android:colorPrimaryDark).

Miscellaneous

Resource Name	Description
`<bool name="disableTTRPopup">`	Defines whether to disable the TTR snackbar popup (true=disable)false by default
`<bool name="contextual_menu_on_toolbar">`	Enable multiple message copy menu over the app toolbar. If *true*, 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 *false*, long pressing a message will display a copy popup menu.
`<color name="bubble_selected_background_color">`	Define the background color of item when it's selected to be copied (if multiple message copy is enabled).
`<string name="notification_large_icon_name">`	The name of a resource to use as the large icon of the push notification (used only when using handlePush with *showNotification*=true).
`<integer name="encryptionVersion">`	Defines the encryption version to use. Currently available version 1 only. 1 - encrypt data 0 - disable encryption
`<string name="csds_url">`	For vanity URL purposes. For regular use please use: ***adminlogin.liveperson.net***
`<integer name="idp_num_history_conversation">`	When user is authenticated, this indicates the number of recent conversations to reload from the server (including their messages) when running for the first time.
`<bool name="show_timestamp_in_ttr_notification">`	When *true* the TTR snackbar will display the time until the agent responds. If set to *false,* a general message is displayed.
`<bool name="show_urgent_button_in_ttr_notification">`	When *true* the TTR snackbar will include a "mark as urgent" button.
`<integer name="ttr_duration">`	Set the duration that the TTR snackbar will be visible.
`<bool name="send_agent_profile_updates_when_conversation_closed">`	When *true* the callback LivePersonCallback#onAgentDetailsChanged will be called with the agent details updates even if the last conversation is closed (in this case it will provide the assigned agent of the last conversation). If *false,* this callback will be called only when the current conversation is active. True by default.
`<string name="custom_button_icon_name">`	Deprecated

`<bool name="ttr_message_off_hours_enabled">`	Defines whether to show the off hours snackbar popup (true=enable).
`<integer name="ttrShowFrequencyInSeconds">`	Define the frequency of the TTR (time to response) messages.
`<bool name="enable_client_only_masking">`	Defines whether to enable or disable client side only masking. False by default.
`<bool name="enable_real_time_masking">`	Defines whether to enable or disable real time masking. False by default.
`<string name="client_only_masking_regex">`	Defines the java regex for client side only masking. By default does not contain any value.
`<string name="real_time_masking_regex">`	Defines the Java regex for real time masking. By default does not contain any value
`<string name="lp_date_format">`	Define date format. More info here
`<string name="lp_time_format">`	Define time format . More info here
`<string name="lp_date_time_format">`	Define date-time format. More info here

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.

Example:

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

ProGuard Configuration

The SDK handles its own obfuscation and all its dependencies according to ProGuard rules.
There is no need to add any ProGuard specific rules that relate 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 also be disabled.

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	Write a message
lp_send	The "Send" button text	Send
lp_no_network_toast_message	A toast message when there is no network	No internet connection. Please check your connection 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_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 resolve 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 this conversation as urgent?
lp_dismiss_urgent_dialog_header	Dismiss urgent confirmation dialog header	Are you sure you want to mark this conversation as not urgent?
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 this topic is resolved?
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_me ssage	Dismiss urgent confirmation dialog text	This means that your conversation will get normal priority.
lp_ttr_message_with_timesta mp	Text in TTR snackbar when timestamp is shown	An agent will respond within the next
lp_ttr_message_minutes	(plurals string that contains: "one" and "others") The *one* or *others* strings is concatenated to the *lp_ttr_message_with_timest amp* string above according to whether it's single **minute** multiple **minutes**	
	one	%1$s minute
	others	%1$s minutes
lp_ttr_message_hours	(plurals string that contains: "one" and "others") The *one* or *others* strings is concatenated to the *lp_ttr_message_with_timest amp* string above according to whether it's single **hour** multiple **hours**	
	one	%1$s hour
	others	%1$s hours
lp_ttr_message_days	(plurals string that contains: "one" and "others") The *one* or *others* strings is concatenated to the *lp_ttr_message_with_timest amp* string above according to whether it's single **day** multiple **days**	
	one	%1$s day
	others	%1$s days
lp_ttr_message_no_timestam p	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 stars are selected in the feedback dialog	Dissatisfied
lp_feedback_3	String displayed when three stars are selected in the feedback dialog	Neither
lp_feedback_4	String displayed when four stars are selected in the feedback dialog	Satisfied
lp_feedback_5	String displayed when five stars are selected in the feedback dialog	Very Satisfied
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 representatives.
lp_feedback_yesno_negative_ title	Negative button text in the feedback dialog	NO
lp_feedback_yesno_positive_t itle	Positive button text in the feedback dialog	YES
lp_feedback_question	Feedback dialog text with agent name	How would you rate your connection with our agent?

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	Resolve the 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 been resolved.
lp_resend_failed_masked_me ssage	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	Represents Java timezone ID that is used in the off hours message. For a full list of the available IDs, use the "Aliases" from [here](#).	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 than 2 days from now. includes 1 param: %1$s - for the full date (MMM dd, yyyy hh:mm a)	Thanks for your message. We will be back online at %1$s
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 the 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. Only the agent can read it.
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 to your project:

Description	Resources name	Size
Default brand avatar on the	lp_messaging_ui_brand_logo	

avatar next to brand bubble (the first brand message) and on agent avatar appearing on the action bar before an agent is assigned. In case you want to define the background color for this avatar - override "brand_logo_background_col or" resource id. (This is relevant for bubble brand's avatar only. Background color of agent avatar on action bar is "agent_avatar_background_c olor").		
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. In case you want to define the background color for this avatar, override "agent_avatar_background_c olor" resource id.	lp_messaging_ui_ic_agent_a vatar	

Locale - Timestamps Formatting

Android provides 4 different default types of date and time formats:

SHORT is completely numeric (12.13.52 or 3:30pm)

MEDIUM is longer and contains the first 3 letters of the month (Jan 12, 1952)

LONG is longer (January 12, 1952 or 3:30:32pm)

FULL specifies the complete time and date (Tuesday, April 12, 1952 AD or 3:30:42pm) PST.

For each feature we added a special resource ID in case customizing the date/time formatting is needed. By default, all these formatting resources are empty in order to take the default device locale.

We define 3 configurable formatting resources:

- For date only (separator):
  <string name="lp_date_format"></string>
- For time only (bubble's timestamp & off hours time in case of today/tomorrow):
  <string name="lp_time_format"></string>
- For date & time together (resolve message & off hours time in case of other date):
  <string name="lp_date_time_format"></string>

Off Hours

Date & Time

Today and tomorrow off hours message use default SHORT time without date according to the locale (default or custom) and to device setting.

If device is set to 12 hours format :
    "Thanks for your message. We will be back online today/tomorrow at *3:30pm*"
If device is set to 24 hours format :
    "Thanks for your message. We will be back online today/tomorrow at *15:30*"

In case you want special **hour** format, you can use
<string name="lp_time_format"></string>
With any **time** format. For ex. - "hh:mm a", "HH:mm" etc..

Date off hours message (not today/tomorrow) use default LONG date and SHORT time according to the locale (default or custom) and to device setting.

If device is set to 12 hours format :
> "Thanks for your message. We will be back online *January 12, 2017* at *3:30pm*"

If device is set to 24 hours format :
> "Thanks for your message. We will be back online *January 12, 2017* at *15:30*"

In case you want special **date/hour** format, you can use

<string name="lp_date_time_format"></string>

With any **date & time** format. For ex. - "MMM d, yyyy hh:mm a", "EEEE dd/mm/yy HH:mm" etc..

Timezone

Off hours can appear in different time zone with this resource ID:

<string name="lp_ttr_message_off_hours_time_zone_id"></string>

Can find list of timezones id here
For ex. - "US/Pacific", "Europe/Berlin".

Bubble timestamp

Bubbles contains only time in SHORT time format, according to the locale (default or custom) and to device setting.
If device is set to 12 hours format : "*3:30pm*"
If device is set to 24 hours format : "*15:30*"

If you wish to configure this time format, override this resource ID:

<string name="lp_time_format"></string>

With any **time** format. For ex. - "hh:mm a", "HH:mm" etc..
This will apply to all bubble's timestamp.

Separator timestamp

Separator contains only date in SHORT date format, according to the locale (default or custom) and to device setting.
"9/25/16" for US locale / "2016/9/25" for JP locale

If you wish to configure this time format - override this resource id :

<string name="lp_date_format"></string>

With any **date** format. For ex. - "MMM d, yyyy", "EEEE dd/mm/yy" etc.

Resolve message

Resolve message use default SHORT date and SHORT time according to the locale (default or custom) and to device setting.
If device is set to 12 hours format (US locale):
　　　"Conversation resolved by [agent name] \n *9/25/16 , 3:30pm*"
If device is set to 24 hours format (US locale):
　　　"Conversation resolved by [agent name] \n *9/25/16 , 15:30*"

In case you want special **date/hour** format, you can use
<string name="lp_date_time_format"></string>
With any **date & time** format. For ex. - "MMM d, yyyy hh:mm a", "EEEE dd/mm/yy HH:mm" etc..

CSAT Behavior

Overview

This document describes the CSAT behaviour and configurations in the Messaging SDK.
You can find all the related configurations in the resources ID table, under [Survey Screen](#).

Show CSAT flow

Show if:
1. CSAT configured to appear according to `<bool name="show_feedback">`
2. Conversation has an assigned agent.
3. Conversation's CSAT wasn't previously submitted.

Dismiss CSAT

The CSAT view is dismissed in one of four cases:
1. User pressed the submit button (answers are sent to the survey).
2. User choose to skip the CSAT (skipped button is pressed).
3. The CSAT is automatically dismissed if it was filled in any other device.
4. If agent resumed the conversation while csat is visible - it will automatically dismissed.

CSAT UI content

CSAT screen includes several content containers:

agentView (avatar and agent name)
1. Could be hidden or not according to `<bool name="show_agent_details_csat">`
2. Contains agent avatar:
   a. If conversation has assigned agent and its image was downloaded previously using profileUrl, this image will be presented in the view.
   b. If no image available, default avatar is presented. It's background and tint color is according to agent bubble with `lp_messaging_ui_ic_agent_avatar` and `agent_avatar_background_color`. More info in ['Modifying Resources'](#)
3. Contains agent name:
   a. By default it's an empty label.
   b. If conversation has assigned agent, the agent's nickName will be used.

ratingQuestionView (stars)
1. Always visible - can't configure its visibility.
2. Stars color is defined by `<color name="feedback_fragment_star">`

3. Rating question includes 'Agent' by default in the text. If conversation has assigned agent and the agent's nickName is not empty, this nickName will be used instead.

resolutionConfirmationView (yes/no)
1. Could be hidden or not according to `<bool name="show_yes_no_question">`
2. If agentView is shown (`"show_agent_details_csat"`), this view will be always hidden (even if `"show_yes_no_question"` is set to true)
3. The question text color is defined with `<color name="feedback_fragment_title_yesno">` all the configuration related to Yes/No buttons explained in 'Survey Screen' resources table and starting with the prefix : `"feedback_fragment_yesno_btn_"`.

Thank You view

1. Could be hidden or not according to `<bool name="show_csat_thank_you">`.
2. If it's not hidden, the Thank You screen will appear for 3 seconds after clicking Submit, and then all the CSAT view will disappear.

Set up your app key to enable push notifications

Log into your LiveEngage account using an administrator's credentials and navigate to **Campaigns.**
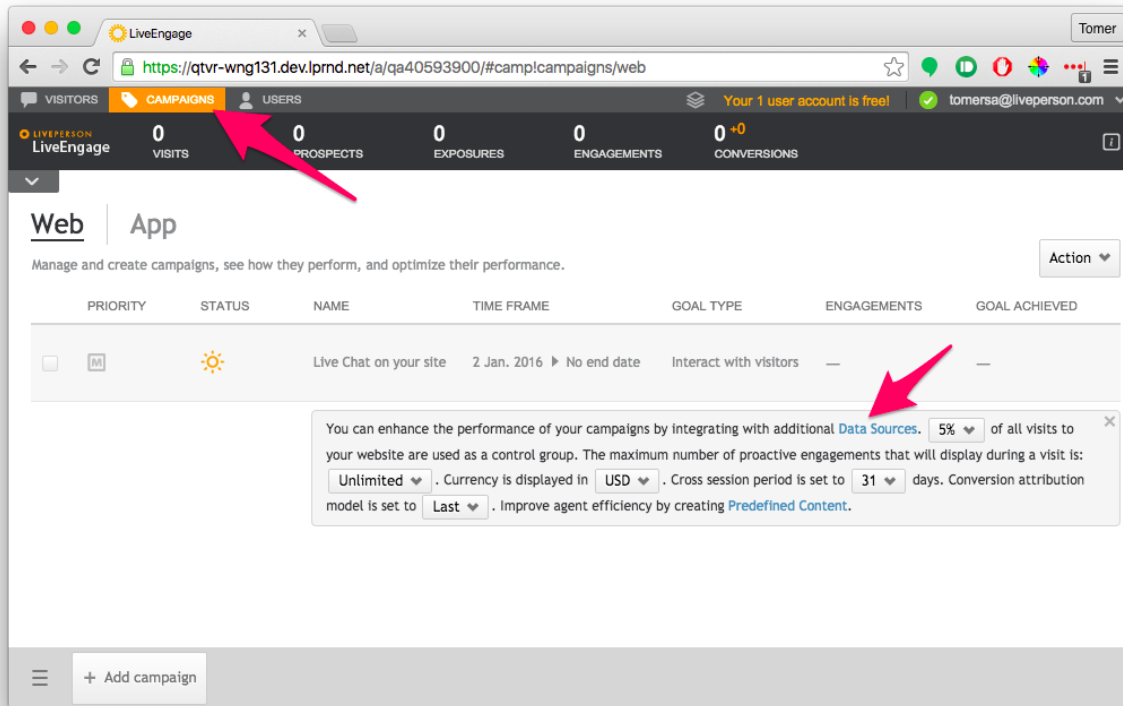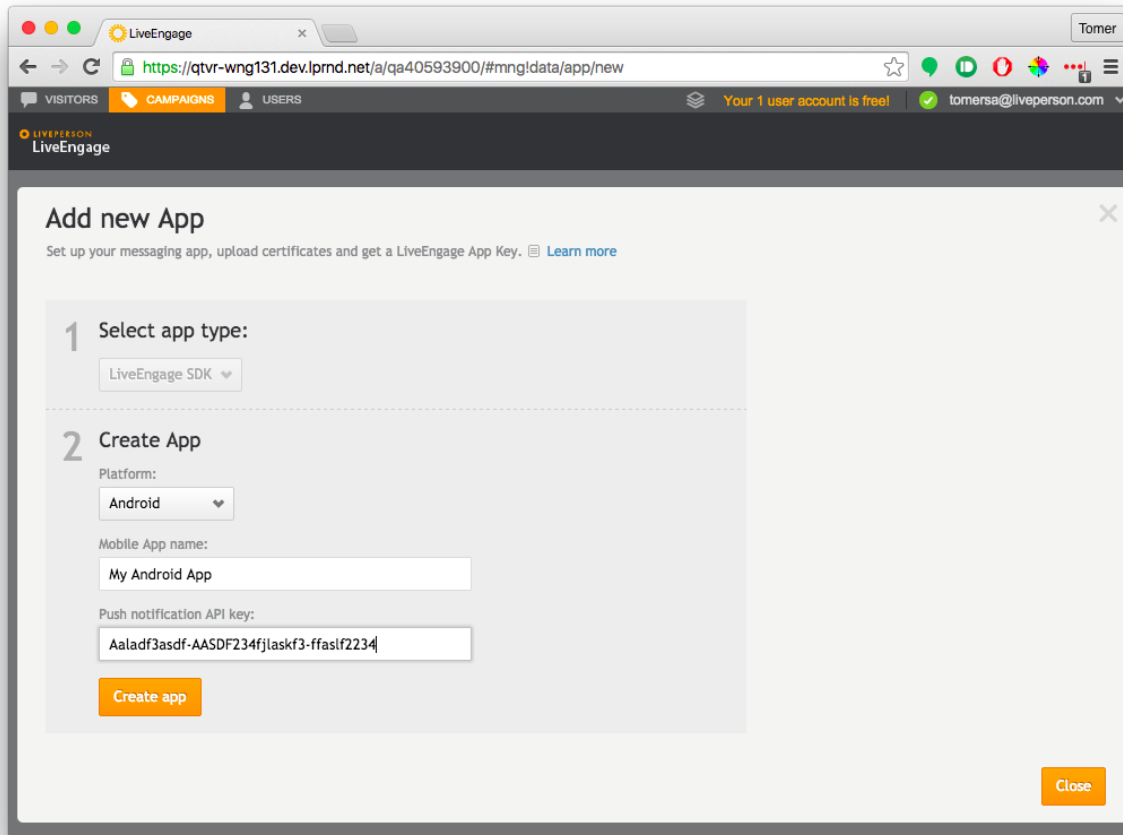
1. Click **Data Sources**, and then select **App**.
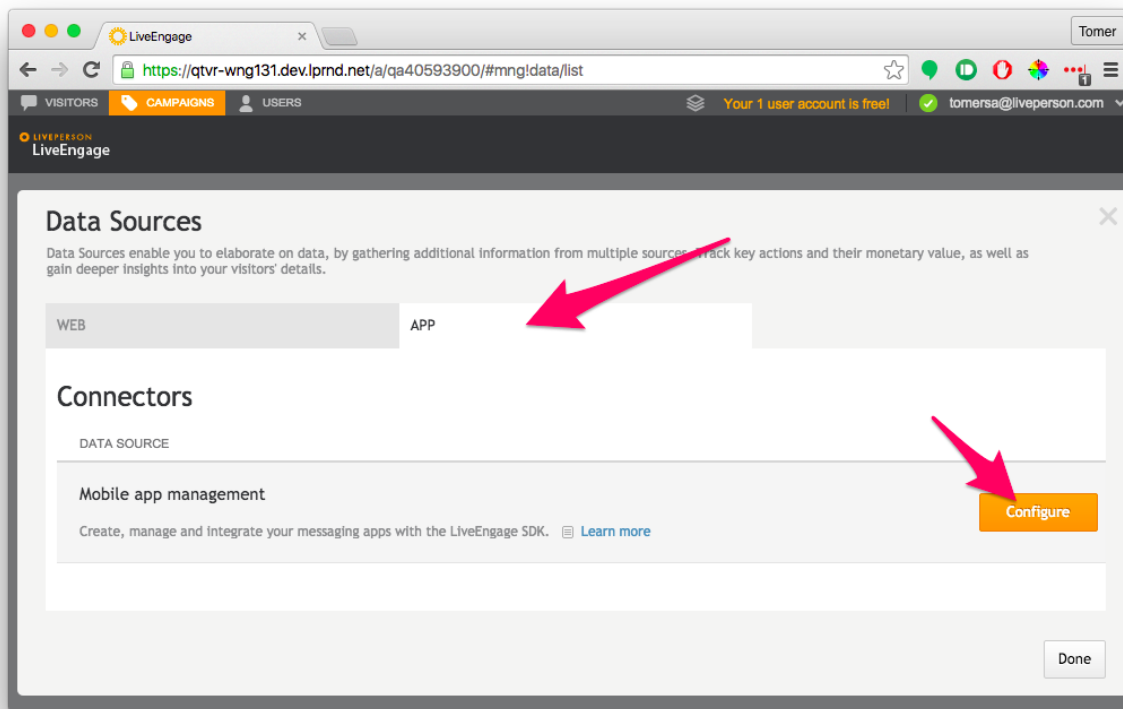
2. Click **Configure**.
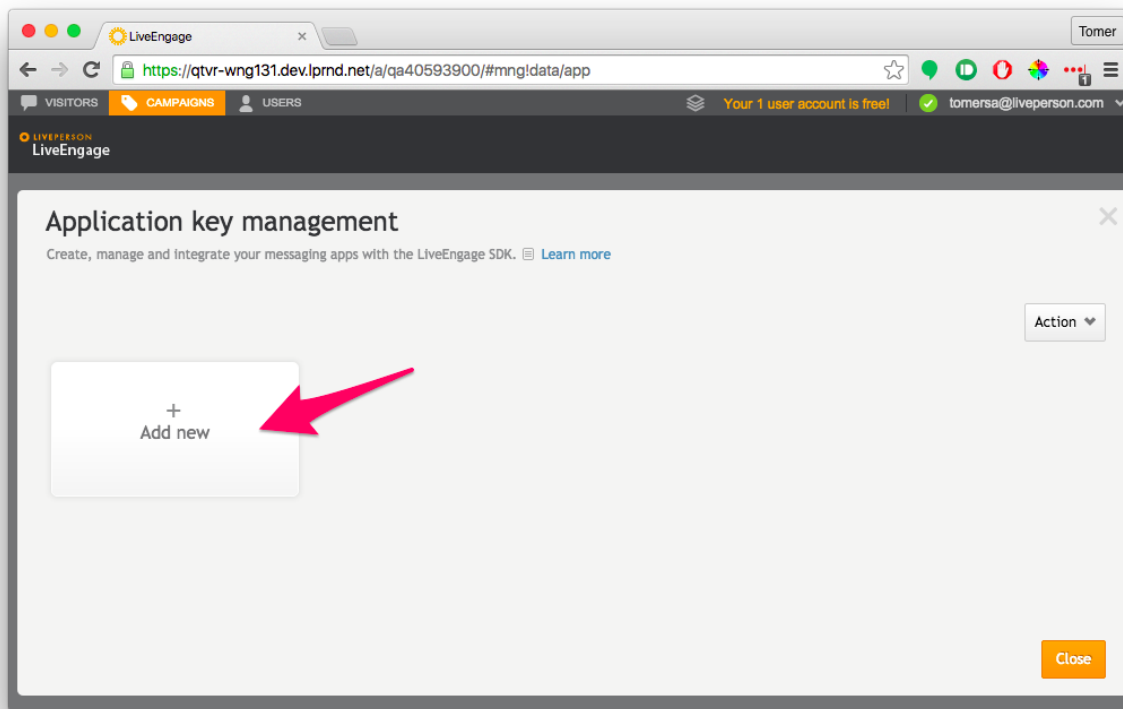
3. Click **Add new** to associate your app with the LiveEngage account.

4. Select your platform as Android, enter your app's name and your push notification API key, and then click **Create app**.

4. Click **Close** to finish the process.

Dependencies

com.squareup.okhttp3:okhttp:3.4.1
An HTTP+HTTP/2 client for Android and Java applications

com.neovisionaries:nv-websocket-client:1.30
High-quality WebSocket client implementation in Java.

com.facebook.fresco:fresco:0.13.0
An Android library for managing images and the memory they use.

Open Source List

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

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.