LiveEngage Enterprise In-App Messenger SDK: Android Deployment Guide

Document Version: 1.0
May 2016

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.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 accountId = "YourLivepersonAccountIdString";
LivePerson.initialize(MainActivity.this, accountId, 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.

** **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, brandId, 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
initialize	To initialize the resources required by the SDK
showConversation	To display the messaging activity
showConversation **with authentication support**	To display the messaging activity with the addition of authentication support
hideConversation	To hide the conversation activity
getConversationFragment	To get the conversation fragment
getConversationFragment **with authentication support**	
reconnect	To reconnect with new authentication key
setUserProfile	To take custom parameters about the consumer as an input, set them for the messaging agent, and attach them to the transcript
registerLPPusher	To register to LivePerson push services
unregisterLPPusher	To unregister from LivePerson push services
handlePush	To receive all incoming push messages in a single function
getSDKVersion	To return the SDK version
setCallback	To gets events from SDK - need to implement **LivePersonCallback**
removeCallBack	To stop getting events from the SDK
checkActiveConversation	To check whether there is an active conversation
checkAgentID	To return agent data such as, first name, last name, email, avatarURL, through callback
markConversationAsUrgent	To mark the current conversation as urgent
markConversationAsNormal	To mark the current conversation as normal

checkConversationIsMarkedAsUrgent	To check whether the current conversation is marked as urgent
resolveConversation	To resolve the current conversation
shutDown	To shut down the SDK
logOut	Logout from the SDK - when all user data should be removed

initialize

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.

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 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 setup with LivePerson backend - 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.

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

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 console 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 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

public static String getSDKVersion()

Returns the SDK version.

setCallback

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

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.

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.
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();
}
```

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(String brandId);
    void onConversationStarted();
    void onConversationResolved();
    void onConnectionChanged(boolean isConnected);
    void onAgentDetailsChanged(AgentData agentData);
    void onCsatDismissed();
    void onConversationMarkedAsUrgent();
    void onConversationMarkedAsNormal();
}

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(String brandId)* 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.

Parameters:
brandId - indicates the account Id for which the token has expired.

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.

Parameters:

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

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.

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
`<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

Styling

Resource Name	Description
`<color name="conversation_background">`	Color code for the entire view background.

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.
`<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.

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.
`<color 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 name="feedback_fragment_submit_btn_disabled">`	Feedback dialog submit button color 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
`<bool name="show_yes_no_question">`	Defines whether to show or hide the yes/no question in the feedback dialog (true=show, false=hide)
`<bool name="show_feedback">`	Defines whether to show the feedback dialog
`<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.

Message Edit Text

Resource Name	Description
`<color name="edit_text_underline_color">`	Color code for the Enter Message control underline color

Miscellaneous

Resource Name	Description
`<bool name="disableTTRPopup">`	Defines whether to disable the TTR snackbar popup (true=disable)
`<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.
`<string name="notification_large_icon_name">`	The name of an 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 uses 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 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 responsd. 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.

`<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 that case - it'll 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
`<string name="custom_button_icon_name">`	Deprecated

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_today	Today header in conversation	TODAY
lp_yesterday	Yesterday header in conversation	YESTERDAY
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_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_mark_as_resolved_dialog_header	Mark as resolved confirmation dialog header	Resolve Conversation
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_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 feedback dialog	Very Satisfied

lp_feedback_thank_you	Text displayed after the feedback dialog is submitted	Survey submitted successfully.\nThank you!
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_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_resend_failed_conversation_closed	Toas message displayed when trying to resend a failed message when conversation is already closed	This conversation has already ended.
lp_new_messages	Notification message displayed when there are multiple push messages	new messages

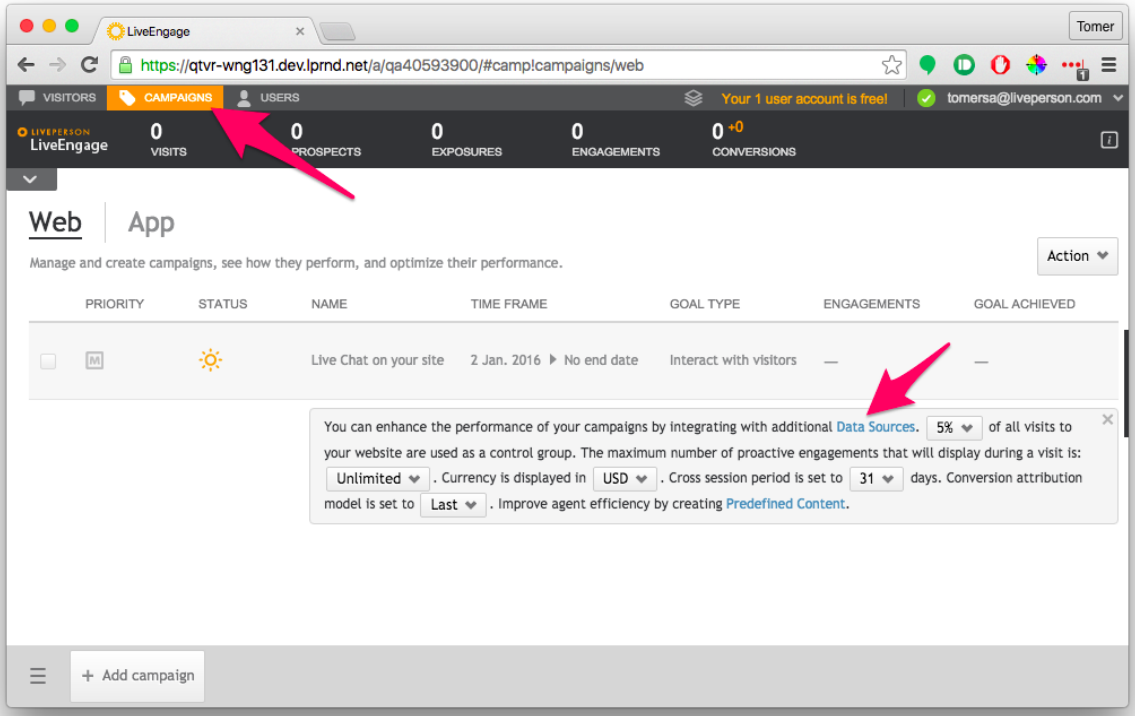
Modifying resources

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

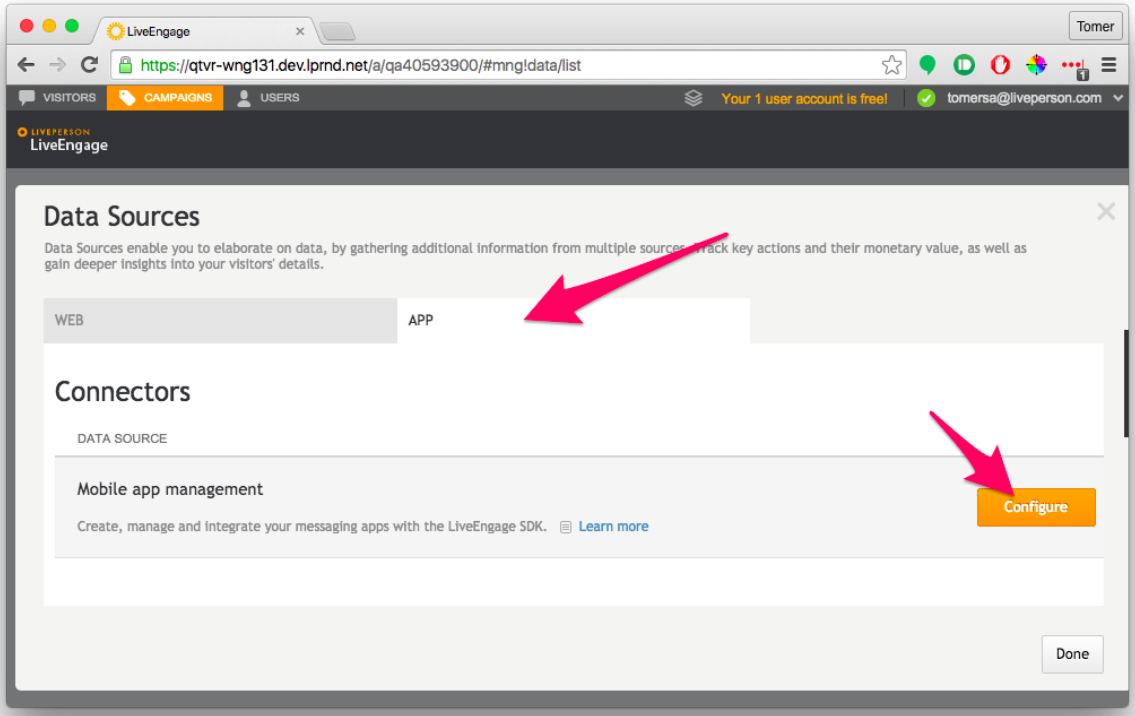
Description	Resources name	Size
Default brand avatar appearing on the action bar before an agent is assigned	lpmessaging_ui_ic_avatar	
Default agent avatar appearing next to an agent's bubble when no avatar URL is assigned on LiveEngage		

Set up your app key to enable push notifications

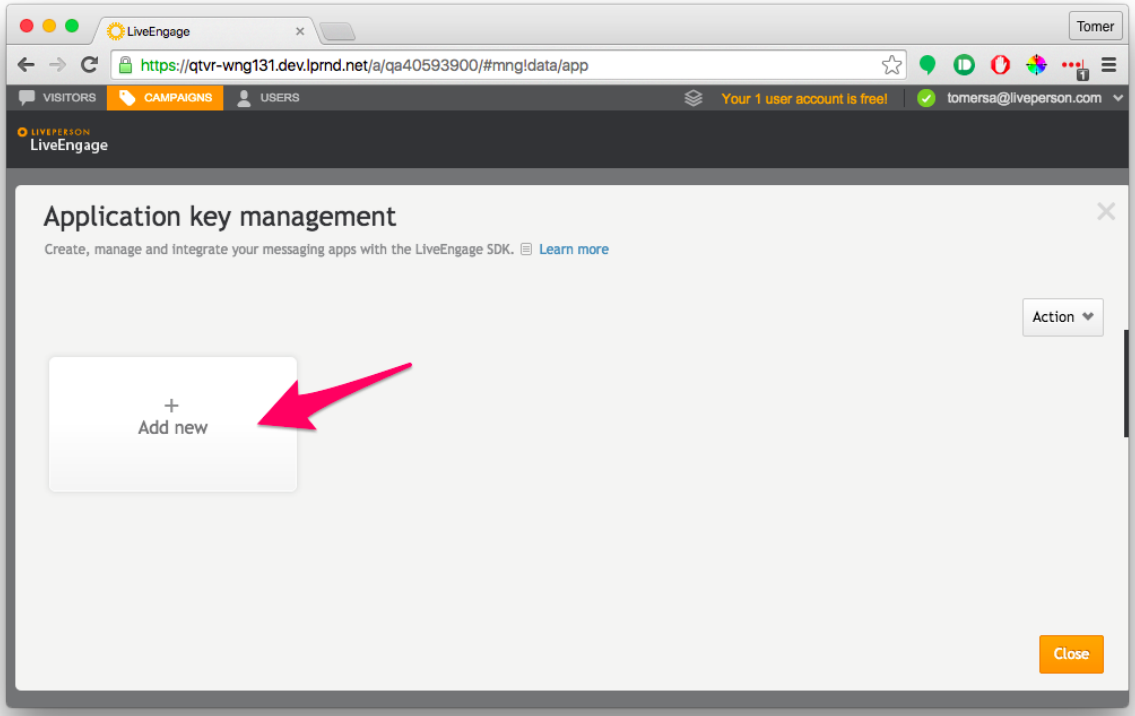
Log into your LiveEngage account using an administrator's credentials and navigate to Campaigns->Data Sources:



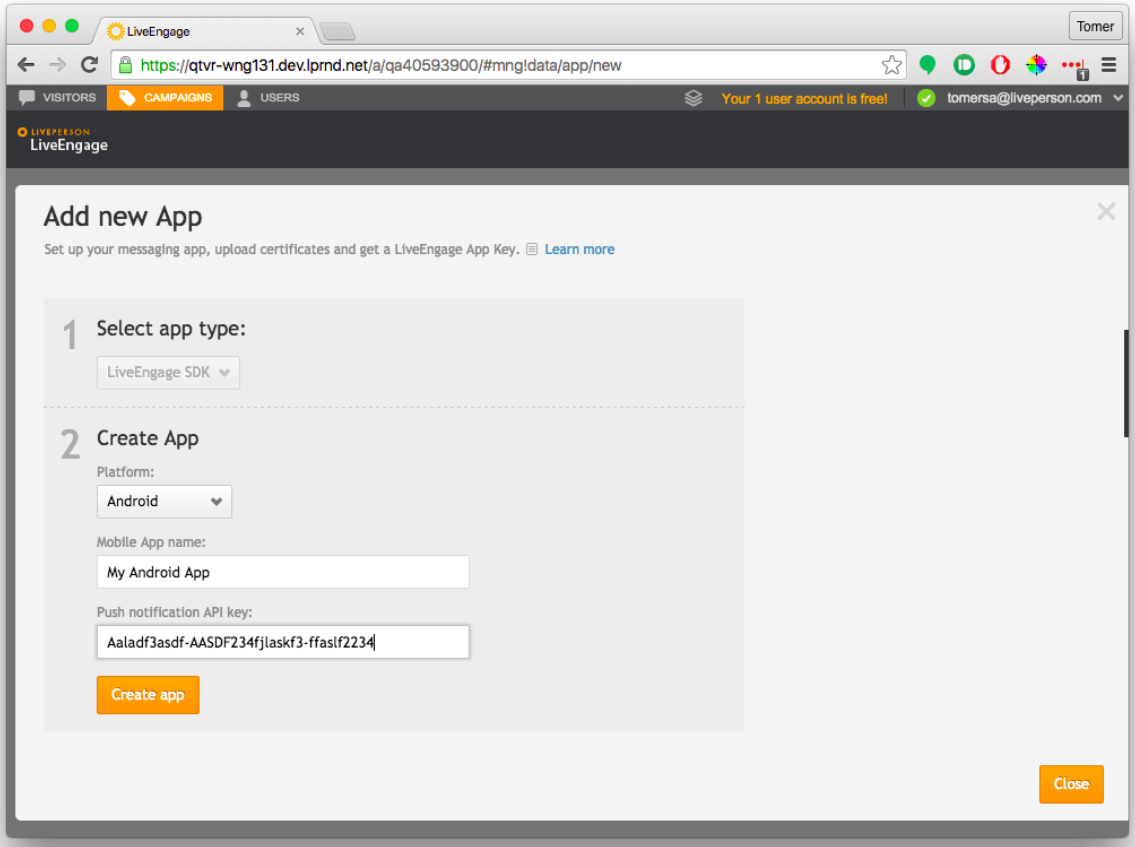
Then, select the “App” section and click “Configure”:



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


LiveEngage
Tomer
https://qtvr-wng131.dev.lprnd.net/a/qa40593900/#mng!data/app
VISITORS
CAMPAIGNS
USERS
Your 1 user account is free!
tomersa@liveperson.com
LIVEPERSON
LiveEngage
Application key management
Create, manage and integrate your messaging apps with the LiveEngage SDK.
Learn more
Action
+
Add new
Close

Select your platform as Android, enter your app’s name and your push notification API key in the appropriate fields. Then click ”Create App”:



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	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