LiveEngage Enterprise In-App Messenger SDK: iOS (swift)

Deployment Guide

Version 1.3
July 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 connection with their customers and increase app engagement and retention.

This Quick Start guide will get you up and running with a project powered by LivePerson as fast as possible. When you're done, you'll be able to send messages between an iOS Device and LiveEngage. You will need a LiveEngage account to complete this Quick Start guide, you can get the number and login information from your LivePerson account team.

The LiveEngage in-App messenger is supported on iOS 8, iOS 9 operating systems and on the following devices: iPhone 6s+, iPhone 6s, iPhone 6+, iPhone 6, iPhone 5s, iPhone 5, iPhone 4s.

Download and unzip the SDK

Click here to download the SDK package. Once downloaded, extract the ZIP file to a folder on your Mac.

Configure project settings to connect LiveEngage SDK

1. Copy the files into the project.
2. In project settings, navigate to the **General** tab, and add all Framework files to the Embedded Binaries section.


```
▼ Embedded Binaries

        📦 LPMessagingSDK.framework …in build/Debug-iphoneos
        📦 LPAMS.framework …in build/Debug-iphoneos
        📦 LPInfra.framework …in build/Debug-iphoneos
            +  −
```

3. In project settings, navigate to the **Build Phases** tab, and add with the '+' button "New Run Script Phase". Add the following script in order to loop through the frameworks embedded in the application and remove unused architectures (used for simulator).

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
```

```
   FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutable)
   FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
   echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

   EXTRACTED_ARCHS=()

   for ARCH in $ARCHS
   do
      echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
      lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o
"$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
      EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
   done

   echo "Merging extracted architectures: ${ARCHS}"
   lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
   rm "${EXTRACTED_ARCHS[@]}"

   echo "Replacing original executable with thinned version"
   rm "$FRAMEWORK_EXECUTABLE_PATH"
   mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```

Initialization

Now that you have the configuration file for your project, you're ready to begin implementing.
To initialize the SDK, you must have a LivePerson account number.

Inside AppDelegate, add:

```
import LPMessagingSDK
```

1. Initialization
   Inside **AppDelegate**, under **didFinishLaunchingWithOptions**, add the following code:

```
do {
    try LPMessagingSDK.instance.initialize()
} catch {
    // SDK has an initialization error...
    return
}
```

2. Setup and call the conversation view
   You'll need to provide your account number and a container view controller

```
let conversationQuery = LPMessagingSDK.instance.getConversationBrandQuery(accountNumber)
```

```
LPMessagingSDK.instance.showConversation(conversationQuery, authenticationCode:
    accountNumber, containerViewController: self)
```

3. In order to remove the conversation view when your container is deallocated, run the following code:

```
let conversationQuery = LPMessagingSDK.instance.getConversationBrandQuery(accountNumber)
LPMessagingSDK.instance.removeConversation(conversationQuery)
```

4. In general tab, make sure that the framework files are under 'Embedded Libraries'.

5. In build settings make sure "Embedded content contains Swift code" is set to "Yes"

Integrations

SDK Initialization and Lifecycle
In order to get started and initialize the in-app messaging SDK, utilize the following functions:

1. **public func** initialize(brandID: String? = **nil**) **throws**
2. **public func** showConversation(conversationQuery: ConversationParamProtocol, authenticationCode: String? = **nil**, containerViewController: UIViewController? = **nil**)
3. **public func** removeConversation(conversationQuery: ConversationParamProtocol)
4. <LPMessagingSDKdelegate> **optional func** LPMessagingSDKConnectionStateChanged(isReady: Bool, brandID: String)
5. <LPMessagingSDKdelegate> **func** LPMessagingSDKError(error: NSError)
6. <LPMessagingSDKdelegate> **optional func** LPMessagingSDKHasConnectionError(error: String?)

Supporting functions:

7. **public func** getConversationBrandQuery(brandID: String) -> ConversationParamProtocol
8. **public func** getConversationBrandAndSkillQuery(brandID: String, skillID: String) -> ConversationParamProtocol
9. **public func** getConversationConsumerQuery(consumerID: String?, brandID: String, agentToken: String) -> ConversationParamProtocol
10. **public func** isBrandReady(brandID: String) -> Bool

Authentication:
For users of oAuth2 for customer authentication, the following functions apply:

1. **public func** reconnect(conversationQuery: ConversationParamProtocol, authenticationCode: String)
2. <LPMessagingSDKdelegate> **func** LPMessagingSDKAuthenticationFailed(error: NSError)

3. <LPMessagingSDKdelegate>   **func** LPMessagingSDKTokenExpired(brandID: String)

UI:

To determine the layout of messaging within the app, you can utilize various actions to control the behavior and UI such as menus, custom buttons, typing indication, etc.

1. **public func** toggleChatActions(accountID: String, sender: UIBarButtonItem? = nil)
2. <LPMessagingSDKdelegate>   **optional func** LPMessagingSDKActionsMenuToggled(toggled: Bool)
3. **public func** setCustomButton(image: UIImage?)
4. <LPMessagingSDKdelegate>   **optional func** LPMessagingSDKCustomButtonTapped()
5. <LPMessagingSDKdelegate>   **optional func** LPMessagingSDKAgentIsTypingStateChanged(isTyping: Bool)
6. <LPMessagingSDKdelegate>   **optional func LPMessagingSDKOffHoursStateChanged(isOffHours: Bool, brandID: String)**

Conversation Lifecycle

During the course of the conversation, consumers can take several actions such as 'mark as urgent' to get faster service or 'resolved conversation' to let your agents know they have received their answers.

1. <LPMessagingSDKdelegate>   **optional func** LPMessagingSDKConversationStarted(conversationID: String?)
2. <LPMessagingSDKdelegate>   **optional func** LPMessagingSDKConversationEnded(conversationID: String?)
3. **public func** checkActiveConversation(conversationQuery: ConversationParamProtocol) -> Bool
4. **public func** markAsUrgent(conversationQuery: ConversationParamProtocol)
5. **public func** isUrgent(conversationQuery: ConversationParamProtocol) -> Bool
6. **public func** dismissUrgent(conversationQuery: ConversationParamProtocol)
7. **public func** resolveConversation(conversationQuery: ConversationParamProtocol)
8. <LPMessagingSDKdelegate> **optional func** LPMessagingSDKCSATScoreSubmissionDidFinish(accountID: String, rating: Int)
9. **public func** clearHistory(conversationQuery: ConversationParamProtocol) **throws**
10. **public func** logout()
11. **public func** destruct()

Notifications

Push and local notifications are a key factor that make the experience better for customers - they never have to stay on your app or keep the window open as they will get a proactive notification as soon as a reply or notice is available.

*Note: in order to enable push notification, you must configure it within LiveEngage UI as well. [See instructions](#).

1. **public func** handlePush(userInfo: [NSObject : AnyObject])
2. **public func** registerPushNotifications(token token: NSData, notificationDelegate: LPMessagingSDKNotificationDelegate? = **nil**, alternateBundleID: String? = **nil**)
3. <LPMessagingSDKNotificationDelegate> **optional func** LPMessagingSDKNotification (didReceivePushNotification notification: LPNotification)
4. <LPMessagingSDKNotificationDelegate> **optional func** LPMessagingSDKNotification(shouldShowPushNotification notification: LPNotification) -> Bool
5. <LPMessagingSDKNotificationDelegate> **optional func** LPMessagingSDKNotification(customLocalPushNotificationView notification: LPNotification) -> UIView
6. <LPMessagingSDKNotificationDelegate> **optional func** LPMessagingSDKNotification(notificationTapped notification: LPNotification)

User Data

Pass and display customer information to the agents, and agent information to the customers.

1. **public func** setUserProfile(lpuser: LPUser, brandID: String)
2. **public func** getAssignedAgent(conversationQuery: ConversationParamProtocol) -> LPUser?
3. <LPMessagingSDKdelegate> **optional func** LPMessagingSDKAgentDetails(agent: LPUser?)

Logs & info

Send logs from LiveEngage to your app. Logs include different severity levels of errors and warnings.

1. **public func** subscribeLogEvents(logLevel: LogLevel, logEvent: LogEventClosure)
2. <LPMessagingSDKdelegate> **func** LPMessagingSDKObseleteVersion(error: NSError)
3. **public func** logInvoked(text: String?, logLevel: LogLevel?, timestamp: String?, className: String?, funcName: String?)
4. <LPMessagingSDKdelegate> **optional func** L
5. **public func** getSDKVersion() -> String?

API Methods

Initialization

The SDK initialization is done only once, inside `appdelegate`. This function checks that the SDK has all mandatory preconditions. For example: it's able to find the bundle file, all the keys in the configFile.plist is available, etc.
If any of the preconditions are not met, an exception is thrown. Once an exception is thrown, you must not do any other call to the SDK.

```
do {
    try LPMessagingSDK.instance.initialize()
} catch {
    // SDK has an initialization error...
    return
}
```

showConversation

This method is used to open the conversation screen and accepts 3 parameters:

1. Conversation Query
The conversation query represents a 'filter' for the conversation screen, determining which of the conversations will be displayed in the following screens. The default conversationQuery sorts the conversations by account number.
See helpers methods below for how to generate a conversation query.

2. Authentication code
The SDK can enable code-flow SSO. If your account uses SSO, pass the auth-code here. Otherwise, you can skip this parameter.

3. Container View Controller
The SDK needs a container view controller. This can be done in two ways:
   a. View Controller mode
      If you provide a container viewController, the SDK will put itself inside as a child viewController. This mode allows you to keep your own navigation bar intact. Using this method, you can use the provided callbacks to retrieve data from the SDK and show it in the navigation bar (users profile data, avatar URL, calling menu items etc.)
   b. Window mode
      If you don't provide a container view controller, the SDK places its UI components on top of the app UI, including the navigation bar.

```
LPMessagingSDK.instance.showConversation(conversationQuery, authenticationCode:
accountNumber, containerViewController: self)
```

removeConversation

When navigating out of the conversation screen, remove the view controller from its container. This is done by calling remove conversation API. The method will remove the SDK UI and clean service or network operation that was running.

```
LPMessagingSDK.instance.removeConversation(conversationQuery)
```

reconnect

When using SSO in an authenticated connection, an auth-code is passed to the SDK (see `showConversation` API). The session in this case might have expiration date (see `LPMessagingSDKTokenExpired`). To reconnect with a new token, use the following 'reconnect' API and pass the new token.

```
LPMessagingSDK.instance.reconnect(query, authenticationCode: "Authentication code")
```

toggleChatActions

This API call is used to open or close the SDK menu.
● If you're using window mode, you won't need to utilize this method as the SDK will have a dedicated button in the navigation bar to toggle the menu.
● If you are using view controller mode - you may call this API to open the SDK menu, or use other APIs to build your own menu.

setCustomButton

In view controller mode only:
You can place a custom button in the SDK navigation bar. Call the API, passing a UIImage that will be placed in the navigation bar.
When the button is pressed, a dedicated callback will be invoked. (See LPMessagingSDKCustomButtonTapped for more information)

checkActiveConversation

Check if there is an active conversation by passing a conversation query.
- Note that conversation query defines a filter that fetches conversations which matche certain conditions. Each query can have one active conversation at most.
- Conversation is said to be active the moment an 'ack' is received from the server. It might not have an assigned agent yet.
- You may call this API only if you are sure that the SDK is in sync with the server, meaning LPMessagingSDKConnectionStateChanged was invoked and isReady is set to true.

```
LPMessagingSDK.instance.checkActiveConversation(conversationQuery)
```

markAsUrgent

A consumer can mark a conversation as 'urgent' in order to request faster response from the agent. You can call the API only if there's an active conversation, otherwise an alert would pop.
The conversation is marked as urgent only after an 'ack' is received from the server.

```
LPMessagingSDK.instance.markAsUrgent(conversationQuery)
```

isUrgent

Checks if the active conversation (if exists) is marked as urgent. Otherwise returns false.
Not that you must check that the SDK is in ready state before calling this method.

```
let isMarkedAsUrgent = LPMessagingSDK.instance.isUrgent(conversationQuery)
```

dismissUrgent

This API is used to cancel 'markAsUrgent' API. It will reset the SLA for the agent response back to default. This API can be called only for open conversations.

```
LPMessagingSDK.instance.dismissUrgent(conversationQuery)
```

resolveConversation

This API enables to resolve a conversation. The API will request the server to mark the active conversation as resolved. If there is not active conversation, alert view will appear.

```
LPMessagingSDK.instance.resolveConversation(conversationQuery)
```

clearHistory

This API may be used only when there is no active conversation.
This API clears the local database. The history is still available on the server, but won't be retrieved from the specific device, unless a fresh install was made.

```
do {
    try LPMessagingSDK.instance.clearHistory(conversationQuery)
} catch {
    // Clear history error...
}
```

Logout

This method is a destructive method that is typically used to clean a user's data before a second user logs into the same device.
This method conducts the following:
- Unregisters from the push notification service
- Clears all SDK persistent data
- Cleans running operations (see destruct)

```
LPMessagingSDK.instance.logout()
```

Destruct

This method is used to clean running operations of the SDK. Note that this method closes all active connections and remove conversations views.

```
LPMessagingSDK.instance.destruct()
```

handlePush

In order to receive all incoming push notifications in a single function and handle them, add the following method. This method cooperates with 2 other API methods:

- This method calls **shouldShowPushNotification method. If the host app returns false, the SDK will not show anything to the UI.**
- **Otherwise, the SDK will ask the host app to provide a view as an in-app notification. If the host app doesn't implement this method, the SDK will use its own implementation.**

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo:
    [NSObject : AnyObject], fetchCompletionHandler completionHandler:
    (UIBackgroundFetchResult) -> Void) {

LPMessagingSDK.instance.handlePush(userInfo)


    }
```

registerPushNotifications

Register to LPMessagingSDK push notifications with the following code in AppDelegate:

- LivePerson push service uses the bundle id to identify your app.
    - "alternateBundleID" is an optional value to use to have LivePerson pusher service identify your app with this identifier.
    - In debug mode the SDK appends "-dev" string to the bundle id.

- Note that push notifications must be pre-configured, and an APN certificate has to be uploaded to LiveEngage platform. See more info in 'how to configure push notifications'

```
func application(application: UIApplication,
    didRegisterForRemoteNotificationsWithDeviceToken deviceToken: NSData) {

LPMessagingSDK.instance.registerPushNotifications(token: deviceToken,
    notificationDelegate: self, alternateBundleID: "__alternateBundleID__")

}
```

didReceivePushNotification

When handling the push notifications, the SDK will form up a struct: LPNotification
And will pass it to the host app.

Add code snippet

shouldShowPushNotification

After calling handlePush, the SDK will ask the host app if it should show an in-app notification in the UI. (See handle push for the full description)

customLocalPushNotificationView

If shouldShowPushNotification is not implemented or it returns yes, the app can implement this method for showing an in-app notification in the UI. In case the method is not implemented, the SDK will provide and show its own view.

Add code snippet

notificationTapped

When tapping a local notification message bar, the following delegate is called:

```
  func LPMessagingSDKNotification(notificationTapped notification: LPNotification) {

        ((UIApplication.sharedApplication().delegate as!
 AppDelegate).mainViewController?.centerViewController as?
 LPNavigationController)?.navigateTo("chatView",data: ["brand":true])

   }
```

setUserProfile

Add custom parameters about the user and set them for the messaging agent.
Passed object is an instance of LPUser (a subclass of NSObject) with the following properties:

```
var firstName: String?
var lastName: String?
var profileImageURL: String?
var phoneNumber: String?
```

```
var employeeID: String?
var uid: String?
```

```
let user = LPUser(firstName: "John", lastName: "Doe", profileImageURL: "URL of image",
phoneNumber: "555-555555")
LPMessagingSDK.instance.setUserProfile(user)
```

getAssignedAgent

Get assigned agent details of the last or current conversation - depending on
'retrieve_assigned_agent_from_last_closed_conversation' in the configFile.plist.
You must check that the SDK is ready before calling this method.

```
let agent = LPMessagingSDK.instance.getAssignedAgent(conversationQuery)
```

subscribeLogEvents

Subscribe to log events (Trace, Debug, Info, Warning, Error) - Each time a log event with the
passed log level will occur, the callback will be invoked with the log object.
Passed object is an instance of LPLog (a subclass of NSObject) with the following properties:

```
var timestamp: String?
var className: String?
var funcName: String?
var text: String?
var logLevel: LogLevel?
```

LogLevel enum:

```
case TRACE
case DEBUG
case INFO
case WARNING
case ERROR
case OFF
public var description: String { get }
public init(string: String)
```

```
LPMessagingSDK.instance.subscribeLogEvents(LogLevel.INFO) { (log) -> () in
    NSLog(log.text)
}
```

logInvoked

After subscribing to log events, you'll get the logs inside this method.

getSDKVersion

Get current SDK version string

```
let sdkVersionString = LPMessagingSDK.instance.getSDKVersion()
```

Callbacks index

LPMessagingSDKConnectionStateChanged

Invoked when the connection state is changed.
- isReady: Bool - Set to true when the SDK is connected and in sync with the server.
- brandID: String - Brand account number

func LPMessagingSDKError(error: NSError)

Called when the SDK has a general error and cannot proceed.

LPMessagingSDKHasConnectionError(error: String?)

Called whenever the SDK received in connection error from the socket.

LPMessagingSDKAuthenticationFailed(error: NSError)

Called when the current session failed due to an authentication error.

LPMessagingSDKTokenExpired

Called when the current session failed due to an authentication error.

Stopped here

LPMessagingSDKCSATScoreSubmissionDidFinish

When a conversation is resolved, a feedback page is presented (CSAT - Customer Satisfaction). This delegate method is invoked after the CSAT is submitted. If the user chooses to skip the CSAT, the delegate method is called with score=0.

Add code snippet

LPMessagingSDKCustomButtonTapped

In window mode only, the app can place a custom button in the SDK UI. When the button is tapped the following delegate method is invoked:

```
LPMessagingSDK.instance.delegate = self
```

When this button is pressed, it will call the following delegate:

```
func LPMessagingSDKCustomButtonTapped() {
    UIApplication.sharedApplication().openURL(NSURL(string: "tel://55555555")!)
}
```

LPMessagingSDKAgentDetails(agent: LPUser?)

Called when the agent details were received or updated. For every message the SDK checks for agent details in order to determine whether the assigned agent was changed. If there is no assigned agent, agent will be nil.

LPMessagingSDKActionsMenuToggled(toggled: Bool)

Called when the action menu was toggled.

LPMessagingSDKHasConnectionError(error: String?)

Called whenever the SDK received in connection error from the socket.

LPMessagingSDKObseleteVersion(error: NSError)

Called when the SDK version is obsolete and needs to be updated.

LPMessagingSDKTokenExpired(brandID: String)

Called when the current session's token is expired.

LPMessagingSDKAgentIsTypingStateChanged(isTyping: Bool)

Called when the typing state of the agent was changed.

LPMessagingSDKConversationStarted(conversationID: String?)

Called when a new conversation was started.

LPMessagingSDKConversationEnded(conversationID: String?)

Called when an open conversation was ended by the user or by the agent.

LPMessagingSDKConnectionStateChanged(isReady: Bool, brandID: String)

Called when the SDK connection was changed. isReady will be true only after the socket is open and login and fetch history processes are done.

func LPMessagingSDKError(error: NSError)

Called when the SDK has a general error and cannot proceed.

LPMessagingSDKCSATScoreSubmissionDidFinish(accountID: String, rating: Int)

Called after the survey feedback has been submitted (rating 1 to 5) or skipped (rating = 0)

LPMessagingSDKCSATCustomTitleView(brandID: String) -> UIView

Delegate method which allows you to set custom title view in the survey navigation

LPMessagingSDKOffHoursStateChanged(isOffHours: Bool, brandID: String)
Delegate which called when an off hours state changed

LPMessagingSDKNotification(didReceivePushNotification notification: LPNotification)

LPMessagingSDKNotification(shouldShowPushNotification notification: LPNotification) -> Bool

LPMessagingSDKNotification(customLocalPushNotificationView notification: LPNotification) -> UIView

LPMessagingSDKNotification(notificationTapped notification: LPNotification)

Config index

The SDK allows you to configure the look and feel of your app using **LPConfig** object. To apply a custom look and feel, create your own configuration instance and assign the attributes you want to customize.
The most suitable place to customize configuration is right after the SDK initialization and before calling showConversation().

To get the default configuration:

```
let configuration = LPConfig.defaultConfiguration
```

To print all configurable attributes and their default values call:

```
LPConfig.printAllConfigurations()
```
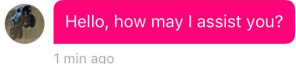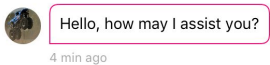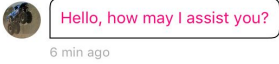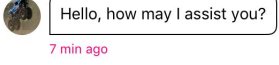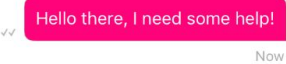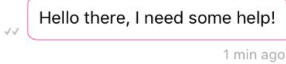
To customize an attribute, follow this example:
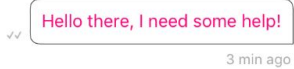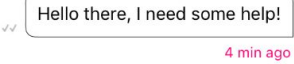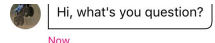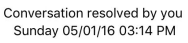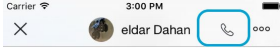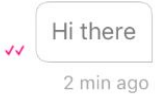
```
configuration.agentBubbleBackgroundColor = UIColor.purpleColor()
configuration.brandName = "Brand Name"
configuration.agentBubbleBorderWidth = 0.5
```
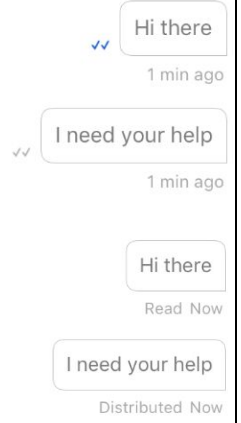
Updating an existing version of the SDK

In case your SDK is using "**configFile.plist**" you should use the following map in order to set new configurations:
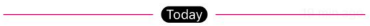Old ConfigFile to new LPConfig Map
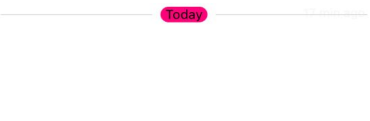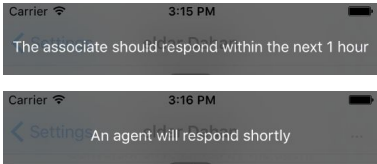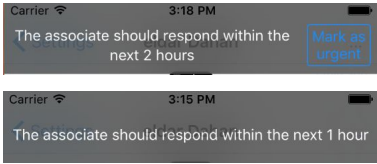
Attribute name	Type	Description	Example
remoteUserBubbleBackgroundColor	UIColor	Color code for the background of the remote user bubble.	Hello, how may I assist you? 1 min ago
remoteUserBubbleBorderColor r	UIColor	Color code for the outline color.	Hello, how may I assist you? 4 min ago
remoteUserBubbleLinkColor	UIColor	Color code for links in the text of the remote user bubble.	Hi, please visit www.liveperson.com Now
remoteUserBubbleTextColor	UIColor	Color code for the text of the remote user bubble.	Hello, how may I assist you? 6 min ago
remoteUserBubbleBorderWidth h	Double	Double number for the outline width.	
remoteUserBubbleTimestampColor olor	UIColor	Color code for the timestamp of the remote user bubble.	Hello, how may I assist you? 7 min ago
remoteUserTypingTintColor	UIColor	Color of the remote user typing bubbles animation	
userBubbleBackgroundColor	UIColor	Color code for the background of the visitor bubble.	Hello there, I need some help! ✓✓ Now
userBubbleBorderColor	UIColor	Color code for the outline color.	Hello there, I need some help! ✓✓ 1 min ago
userBubbleLinkColor	UIColor	Color code for links in the text of the visitor bubble.	www.liveperson.com ✓✓ Now

userBubbleTextColor	`UIColor`	Color code for the text of the visitor bubble.	Hello there, I need some help! ✓✓ 3 min ago
userBubbleBorderWidth	`Double`	Double number for the outline width.	
userBubbleTimestampColor	`UIColor`	Color code for the timestamp of the visitor bubble.	✓✓ Hello there, I need some help! 4 min ago
systemBubbleTextColor	`UIColor`	Color code for the text of the system messages.	Hi, what's you question? Now Conversation resolved by you Sunday 05/01/16 03:14 PM
customButtonIconName	`String`	Custom button icon filename without extension. This will be displayed on the navigation bar.	Carrier 📶 3:00 PM ✕ eldar Dahan 📞 ∘∘∘
customButtonDescription	`String`	Accessibility voiceover string for the custom button.	
checkmarkVisibility	`Checkmar ksState (Integer Enum)`	Checkmark visibility of the following options (type `CheckmarksState):` `SentOnly` - Show checkmarks for only Sent messages. `SentAndAccepted` - Show checkmarks for only Sent and Accepted messages. `All` - Show checkmarks for Sent, Accepted and Read messages.	
checkmarkColor	`UIColor`	Color of read indication signs	✓✓ Hi there 2 min ago

readReceiptTextSent	**String**	Text for sent indication	
readReceiptTextDistributed	**String**	Text for distributed indication	
readReceiptTextRead	**String**	Text for read indication.	
isReadReceiptTextMode	**Bool**	Two options for read indication: ● Read Receipt with Text Mode ● Read Receipt with Icon Mode If the parameter set as true the mode will be Text. If the parameter set as false the mode will be Icon. Default value is true.	
csatSubmitButtonCornerRadius	**Double**	Corner radius of the Submit button	
csatSubmitButtonBackgroundColor	**UIColor**	Background color code of the Submit button	
csatSubmitButtonTextColor	**UIColor**	Text color code of the Submit button	
csatRatingButtonSelectedColor	**UIColor**	Background Color code of the rating buttons	
csatResolutionButtonSelectedColor	**UIColor**	Color code for the resolution confirmation buttons (YES/NO) when selected	
csatResolutionFeedbackText	**String**	Text for the feedback label	
csatResolutionQuestionText	**String**	Text for the resolution confirmation question	

csatAllTitlesTextColor	`UIColor`	Titles text colors for all labels	
csatResolutionHidden	`Bool`	Hides the yes/no question	
csatNavigationBackgroundColor	`UIColor`	Background color of the navigation of the survey	
csatNavigationTitleColor	`UIColor`	Text color of the the title in the survey navigation	
csatSkipButtonColor	`UIColor`	Skip survey button color	
csatUIStatusBarStyleLightContent	`Bool`	Allow the UI status bar to take the color of the survey navigation bar color	
maxConversationToFetch	`UInt`	The amount of conversations to fetch on loading	
maxPreviousConversationToPresent	`UInt`	Amount of conversations to show in advance	
deleteClosedConversationOlderThanMonths	`UInt`	Upon SDK initialization, all closed conversation with end date older than X months, will get deleted from the database. Setting 0 will delete all closed conversation.	
country	`String`	Country code. If no value is provided, the SDK will use the country according to the device's locale.	
language	`String`	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.	

brandName	String	The brand name will be shown as a title on toolbar when there is no active conversation.	
conversationBackgroundColor	UIColor	Color code for the entire view background. Default is white color.	
dateSeparatorTitleBackgroundColor	UIColor	Color code for date separator title background color	
dateSeparatorTextColor	UIColor	Color code for date separator text color	
dateSeparatorLineBackgroundColor	UIColor	Color code for date separator line background color	
dateSeparatorBackgroundColor	UIColor	Color code for date separator background color	
sendButtonDisabledTextColor	UIColor	Color code for 'Send' button title in disabled mode	
sendButtonEnabledTextColor	UIColor	Color code for 'Send' button title in enabled mode	
editTextUnderlineColor	UIColor	Color code for underline background in 'edit' view	
retrieveAssignedAgentFromLastClosedConversation	Bool	When using "getAssignedAgent" method, this option let you decide whether to get assigned agents from active conversation only, or also from the last closed conversation in case there is no active conversation.	

		If not assigned agent is available this method will return nil.	
notificationShowDurationInSeconds	Double	Display duration of the local notification in seconds. Such as: TimeToRespond notification, local notification etc.	
ttrFirstTimeDelay	Double	TTR - Time To Respond Number of seconds before the first TTR notification appears	
ttrShouldShowTimestamp	Bool	TTR - Time To Respond **Enable:** Shows a time stamp in the TTR notification. **Disable:** Shows: "An agent will respond shortly"	
showUrgentButtonInTTRNotification	Bool	TTR - Time To Respond Enable presentation of 'Urgent' button in the TTR notification	
showOffHoursBanner	Bool	Show offline banner	
ttrBannerBackgroundColor	UIColor	Color of background for banner	
ttrBannerTextColor	UIColor	Text color of the banner	
ttrBannerOpacityAlpha	Double	Opacity level of the banner background (values: 0.0 - 1.0)	
offHoursTimeZoneName	String	Off Hours time zone name string based on [NSTimeZone knownTimeZoneNames].	

		If sending empty string, the local timezone will be used (Server sends UTC time).	
csdsDomain	String	CSDS Domain URL. For brands that need to control the URL that is the gateway for LivePerson services, use this key to set a URL of your choice.	
remoteUserAvatarBackground Color	UIColor	Background color of the remote user's avatar	
remoteUserAvatarIconColor	UIColor	Icon's avatar color within the avatar's background	
brandAvatarImage	UIImage?	Set avatar image for brand. This is an optional UIImage that if is set to nil a default avatar will be presented. Image ratio should be 1:1 (square) and at least 50x50 pixels	
enableClientOnlyMasking	Bool	Determines whether to enable using regular expression to control which part of the text to mask, all masked data will appear as asterisks, will be saved to local db masked and will be sent to the server unmasked. Default = false	
enableRealTimeMasking	Bool	Determines whether to enable using regular expression to control which part of the text to mask, all masked data will appear as asterisks, will be saved to local db masked and sent to	

		the server masked. Default is false	
clientOnlyMaskingRegex	String	Regular expression string to control which part of the text to mask, all masked data will appear as asterisks, will be saved to local db masked and will be sent to the server unmasked. Default is empty, means no regex.	
realTimeMaskingRegex	String	Regular expression string to control which part of the text to mask, all masked data will appear as asterisks, will be saved to local db masked and will be sent to the server unmasked. Default is empty, means no regex.	

String localization in SDK

The SDK allows you to override the string localization in any supported language you like.
To apply a custom localization file with your own strings, use your existing localization files or if needed, create a new .strings file with the desired language as file name in your app and just override the values with the SDK keys.

For example, overriding the SDK string of 'send' in English:

```
    1.  Create/Use existing English.strings file in your app.
    2.  Add new key: "send" = "<ANY NEW VALUE>";

Your 'send' string implementation will override the localization in English in the SDK
```

In order to all the localized keys in the SDK call:

```
LPMessagingSDK.instance.printAllLocalizedKeys()
```

In order to get all localized supported languages in the SDK call:

```
LPMessagingSDK.instance.printSupportedLanguages()
```

iOS Certificate creation

In order for the push notification to work you'll need 2 .pem files:

1. A certificate file stored using a pem forma.
2. A Key file stored using a pem format without a password.

Creating a Certificate Signing Request file. (with this file we'll create both .p12 file and .crt file)

To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac. To create a CSR file, follow the instructions below to create one using Keychain Access.

In the Applications folder on your Mac, open the Utilities folder and launch Keychain Access.

Within the Keychain Access drop down menu, select Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority.

- In the Certificate Information window, enter the following information:
    - In the User Email Address field, enter your email address.
    - In the Common Name field, create a name for your private key (e.g., John Doe Dev Key).
    - The CA Email Address field should be left empty.
    - In the "Request is" group, select the "Saved to disk" option.
- Click Continue within Keychain Access to complete the CSR generating process.

Finally, download the certificate and run it, which should add it to your Keychain, paired with a private key:

Creating a key .p12 file

Right-click on your new push certificate and choose **Export**:

Save it as pushNotification.p12 as a .p12 file:
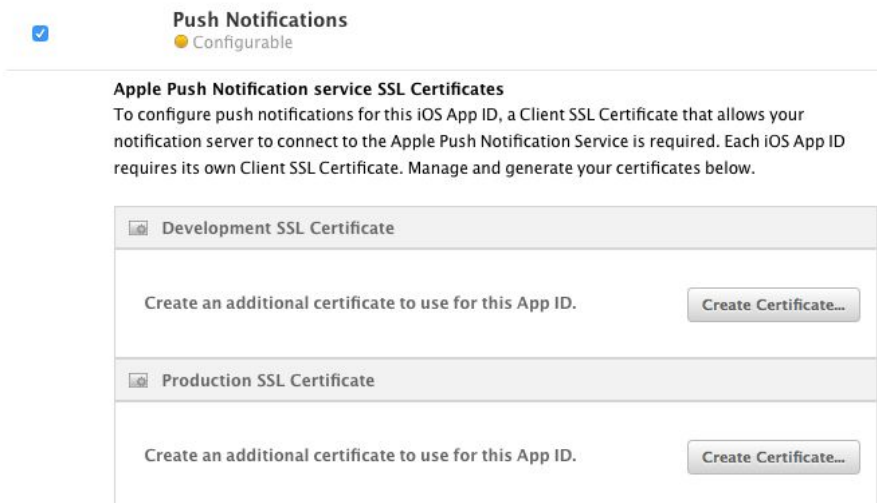
You will be prompted to enter a password for the p12. You can either leave this blank or enter a password of your choosing.

Creating a push notification certificate. (der format .crt file)

- Go to your app area in iOS member area.
- Press create certificate under push notification area (Developer or Production).

- Choose your .csr file that you have created in the previous stage.

Upload CSR file.
Select .certSigningRequest file saved on your Mac.

Choose File...

- Download the file, we will use this file to create the .pem format certificate.

Download, Install and Backup
Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.

Name: Apple Development iOS Push Services: com.liveperson.com.ios.Theo
Type: APNs Development iOS
Expires: May 23, 2017

Download

Creating both key.pem file and cert.pem file (Use them when configuring LiveEngage Push Notification)
- Open the terminal and locate the folder you save the file to.

- create cert pem:
  openssl x509 -in aps_development.cer -inform der -out cert.pem

- Convert the private key's .p12 file into a .pem file:
  openssl pkcs12 -nocerts -out keyWithPassword.pem -in key.p12
  You will be prompt to enter a passphrase for this file. Enter anything and remember it for the next step.

- RSA .pem key (no password)
  openssl rsa -in hostkey.pem -out hostkey.pem
  You will be prompt to enter a passphrase, enter the same passphrase you used in the previous step.

LiveEngage Configuration

Push Notifications

Set up your app key to enable push notifications.
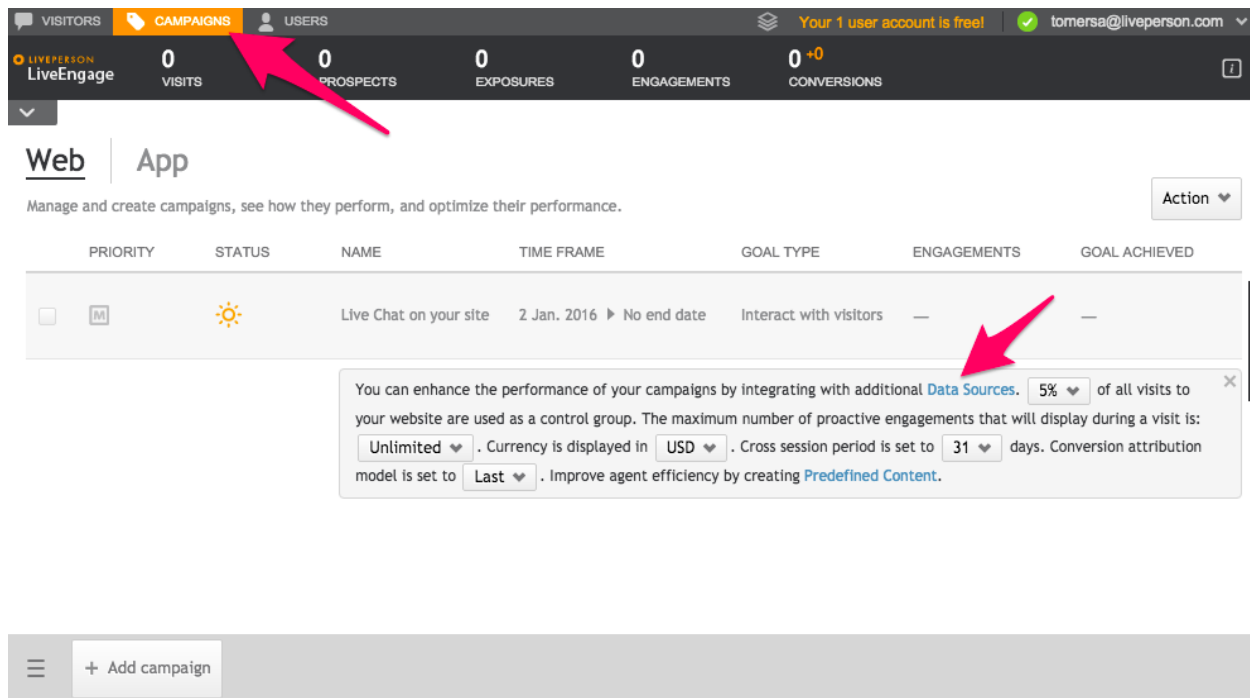
Before you begin the setup, you must ensure your LiveEngage account is configured and connected to the SDK.

Enter your LiveEngage account through this [Login URL](#)
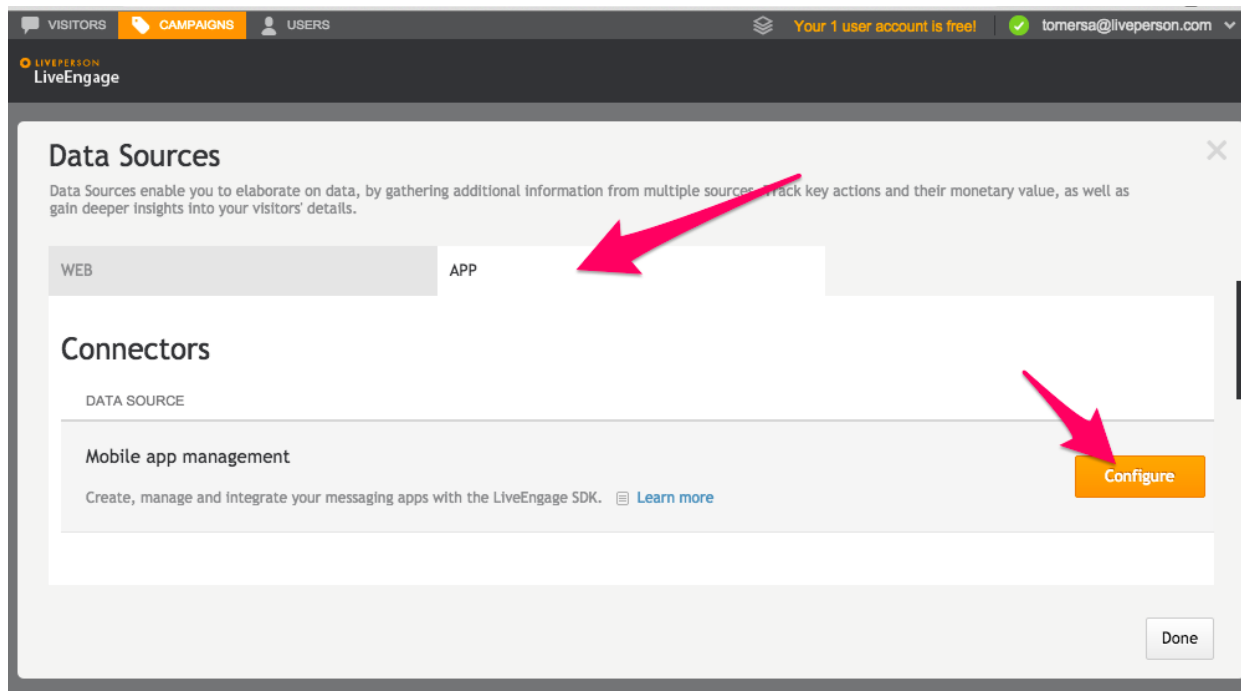You will need the following info from your LivePerson account team:
- LiveEngage account number
- User name (must be an administrator user)
- password

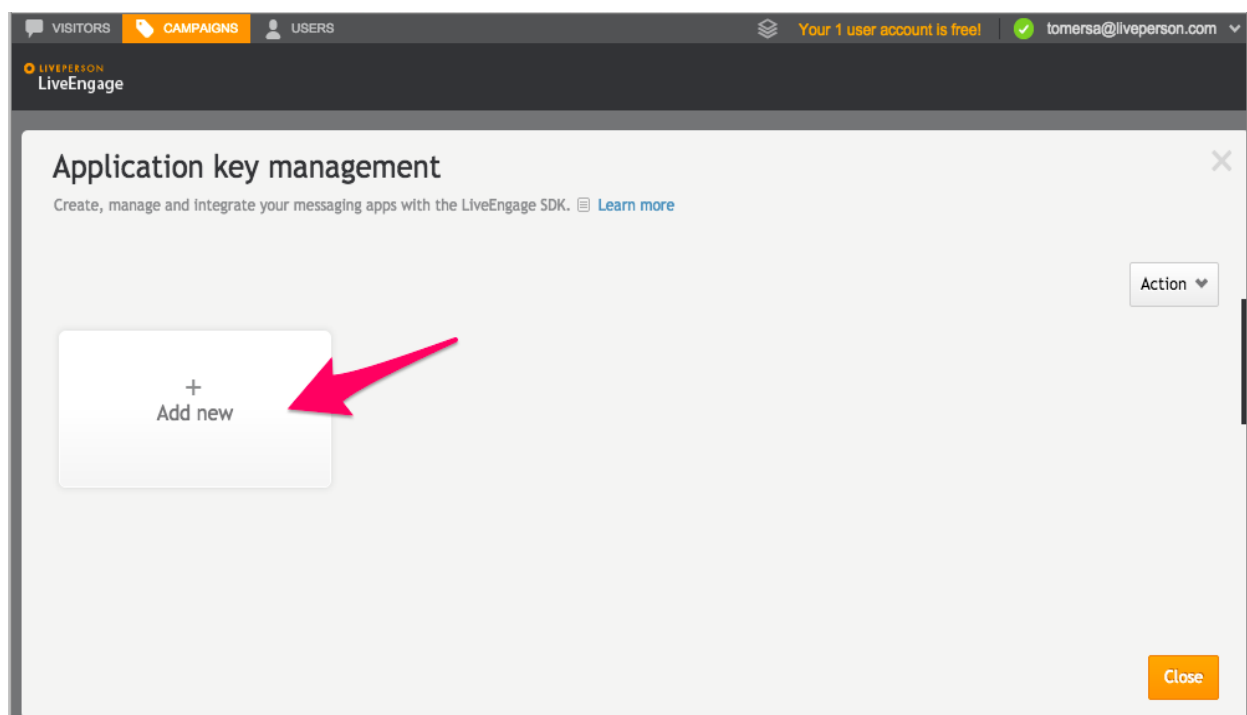Within LiveEngage, navigate to Campaigns->**Data Sources**:

Then, select the "App" section and click "Configure":

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

Select your platform as iOS, enter your app's name and click "Create app". Then, upload your app certificate and key file in the appropriate locations:

**Important - If you are using a development certificate you should uncheck the Production checkbox and add "-Dev" postfix to the Mobile app name (For example if your app bundle id is AppId your mobile app name should be "AppId-Dev".
If you are using a production certificate you should leave the production checkbox checked and insert to the Mobile App name your App bundle Id as it is.

Click "Close" to finish the process.

Open source list

The following open source code is used within the LiveEngage SDK. Licensing terms for use of this code require you to mention the list of these sources in the end customer product or documentation. No additional fees or costs are associated with use of these sources.

Name	Licence
Reachability	Apple inc
Starscream	Apache
UIRefreshControl+UITableView	MIT
TTTAttributedLabel	Apache
NSDate+Extension	License