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

See Objective-C

Deployment Guide

Version 1.5.1

November 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 iOS 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.

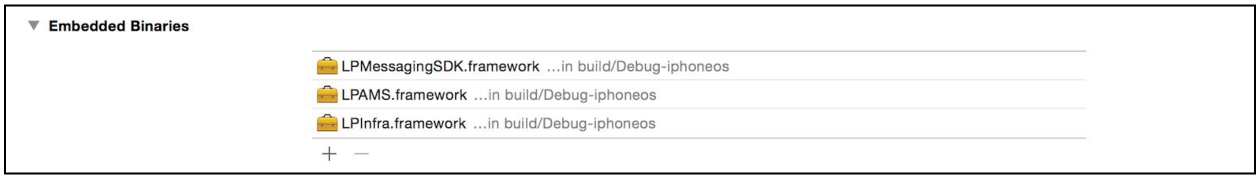
The LiveEngage In-App Messenger is supported on iOS operating systems 8 and above, and on iPhone devices 4S and above.

Step 1: Download and unzip the SDK

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

Step 2: 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. If you are using Xcode 8 with iOS 10 (or later): In order to use LiveEngage SDK on an iOS 10 simulator, you must have at least one Project Capability enabled, such as Keychain Sharing or even Push Notifications.
   To enable Capabilities: In project setting, navigate to **Capabilities** and enable **Keychain Sharing**. LiveEngage iOS SDK uses keychain to store sensitive settings and data. This step is a workaround for an open Apple bug that fails to use keychain store in Xcode 8 and iOS 10: https://openradar.appspot.com/27422249**.**
4. In project settings, navigate to the **Build Phases** tab, and click the **+** button to add a **New Run Script Phase**. Add the script below in order to loop through the frameworks embedded in the application and remove unused architectures (used for simulator). This step is a workaround for

a known iOS issue http://www.openradar.me/radar?id=6409498411401216 and is necessary for archiving your app before publishing it to the App Store.

```
bash "${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH}/LPInfra.framework/frameworks-strip.sh"
```

Step 3: 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
import LPInfra
import LPAMS
```

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

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

2. Set up and call the conversation view. You'll need to provide your LivePerson 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 the **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**.

Advanced Configurations

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:

1. `public func getConversationBrandQuery(brandID: String) -> ConversationParamProtocol`
2. `public func getConversationBrandAndSkillQuery(brandID: String, skillID: String) -> ConversationParamProtocol`
3. `public func getConversationConsumerQuery(consumerID: String?, brandID: String, agentToken: String) -> ConversationParamProtocol`
4. `public func isBrandReady(brandID: String) -> Bool`

SDK delegates

The SDK uses 2 delegates:

1. LPMessagingSDKdelegate - for lifecycle and connectivity events.
2. LPMessagingSDKNotificationDelegate - for handling push and in app notifications.

You should implement and set the delegate for the above in order to receive notification from the SDK.

Authentication

For users of OAuth 2.0 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. `<LPMessagingSDKdelegate>   optional func LPMessagingSDKCustomButtonTapped()`
4. `<LPMessagingSDKdelegate>   optional func LPMessagingSDKAgentIsTypingStateChanged(isTyping: Bool)`
5. `<LPMessagingSDKdelegate>   optional func LPMessagingSDKOffHoursStateChanged(isOffHours: Bool, brandID: String)`
6. `<LPMessagingSDKdelegate>   optional func LPMessagingSDKConversationViewControllerDidDismiss()`

Conversation Lifecycle

During the course of the conversation, consumers can take several actions such as Mark as urgent to receive a faster service, or Resolve 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 LPMessagingSDKConversationCSATDismissedOnSubmittion(conversationID: String?)`
9. `<LPMessagingSDKdelegate>  optional func LPMessagingSDKCSATScoreSubmissionDidFinish(brandID: String, rating: Int)`
10. `public func clearHistory(conversationQuery: ConversationParamProtocol) throws`
11. `public func logout()`
12. `public func destruct()`

Notifications

Push and local notifications are a key factor that make the experience better for consumers - they never have to stay in 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 notifications, you must also configure them within the LiveEngage UI. 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 consumer information to agents, and agent information to consumers.

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

Logs and Info

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

1. `public func subscribeLogEvents(logLevel: LogLevel)`
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

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

API Name	Purpose
Initialization	Initialize the resources required by the SDK.
showConversation	Open the conversation screen.
removeConversation	Remove the SDK UI and clean the service or network operation that was running.
reconnect	Reconnect with new authentication key.
toggleChatActions	Open or close the SDK menu.
checkActiveConversation	Check whether there is an active conversation.
markAsUrgent	Mark the current conversation as urgent.
isUgent	Check whether the current conversation is marked as urgent.
dismissUrgent	Cancel the markAsUrgent API.
resolveConversation	Resolve the current conversation.
clearHistory	Clear all conversations from device.
logOut	Logout from the SDK - when all user data should be removed.
destruct	Close all active connections and remove conversation view.
handlePush	Receive all incoming push messages in a single function.
registerPushNotifications	Register to LPMessagingSDK push notifications.
setUserProfile	Take custom parameters about the consumer as an input, set them for the messaging agent, and attach them to the transcript.
getAssignedAgent	Get assigned agent details of last / current conversation.
subscribeLogEvents	Subscribe to log events.
logInvoked	Invoked when subscribed to LogsManager.instance.subscribeLogEvents.
getSDKVersion	Get current SDK version string.

Initialization

The SDK initialization is done only once, inside **`appdelegate`**. This function checks that the SDK has all mandatory preconditions. For example, it is able to find the bundle file, verify that all the pre-defined configurations are valid, and more. 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(“Your Account ID”)
} catch {
    // SDK has an initialization error...
    return
}
```

showConversation

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

- **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 above for how to generate a conversation query.

- **Authentication code**

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

- **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 the 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 an 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.

checkActiveConversation

Check if there is an active conversation by passing a conversation query.

- Note that conversation query defines a filter that fetches conversations which match 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 may not yet have an assigned agent.
- 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 a faster response from the agent. You can call the API only if there's an active conversation, otherwise an alert will be triggered. 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 existing) is marked as urgent. Otherwise returns false. Note 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 the 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 a conversation to be resolved. The API will request the server to mark the active conversation as resolved. If there is no active conversation, an alert will be displayed.

```
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 installation Is 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 removes conversation 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 two other API methods:

- This method calls the 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 that can be used so that the LivePerson pusher service identifies 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 the LiveEngage platform. See more info on how to configure push notifications.

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

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

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 retrieveAssignedAgentFromLastClosedConversation in the LPConfig defaultConfiguration.
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 occurs, 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 will get the logs inside this method.

getSDKVersion

Get current SDK version string.

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

Callbacks index

The SDK uses 2 delegates:

1. LPMessagingSDKdelegate - for lifecycle and connectivity events
2. LPMessagingSDKNotificationDelegate - for handling push and in app notifications

LPMessagingSDKdelegate Methods

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.
If there is an SDK initialization error, the SDK can not proceed, and you should not call any other SDK API.
There are also other possible errors such as send message error.

LPMessagingSDKHasConnectionError(error: String?)

Called whenever the SDK receives a connection error from the socket.

LPMessagingSDKAuthenticationFailed(error: NSError)

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

LPMessagingSDKTokenExpired

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

LPMessagingSDKActionsMenuToggled(toggled: Bool)

Called when the action menu is toggled.

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")!)
}
```

LPMessagingSDKAgentIsTypingStateChanged(isTyping: Bool)

Called when the typing state of the agent is changed.

LPMessagingSDKOffHoursStateChanged(isOffHours: Bool, brandID: String)

Delegate which is called when an off hours state changes.

LPMessagingSDKConversationViewControllerDidDismiss()

Delegate which is called when the conversation viewcontroller is dismissed (both for window mode and viewController mode).

LPMessagingSDKConversationStarted(conversationID: String?)

Called when a new conversation is started.

LPMessagingSDKConversationEnded(conversationID: String?)

Called when an open conversation is ended by the consumer or by the agent.

LPMessagingSDKConversationCSATDismissedOnSubmittion(conversationID: String?)

Called after CSAT screen is dismissed by clicking **Submit**.

LPMessagingSDKCSATScoreSubmissionDidFinish(brandID: String, rating: Int)

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.

LPMessagingSDKAgentDetails(agent: LPUser?)

Called when agent details are 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, for instance, when the consumer is returned to queue.

LPMessagingSDKObseleteVersion(error: NSError)

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

LPMessagingSDKNotificationDelegate Methods

LPMessagingSDKNotification(didReceivePushNotification notification: LPNotification)

Called when handling the push notifications, the SDK will form up a struct: LPNotification, which will then be passed to the host app.

LPMessagingSDKNotification(shouldShowPushNotification notification: LPNotification) -> Bool

Called after calling handlePush, the SDK will ask the host app if it should display an in-app notification in the UI. (See handlePush for the full description).

LPMessagingSDKNotification(customLocalPushNotificationView notification: LPNotification) -> UIView

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.

LPMessagingSDKNotification(notificationTapped notification: LPNotification)

Called when tapping a local notification message bar when a remote push notification received. You should implement this delegate method if you wish to navigate and show the conversation screen.

```
func LPMessagingSDKNotification(notificationTapped notification: LPNotification) {
    // Navigate to a desired view controller
  }
```

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 time 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()
```

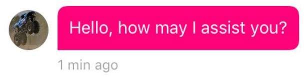
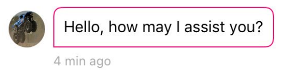
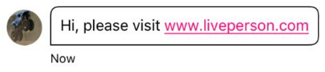
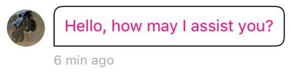
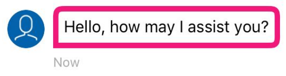
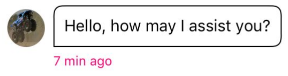
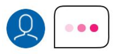
To customize an attribute, follow this example:

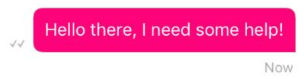
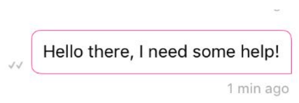
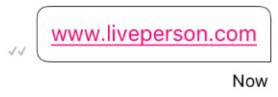
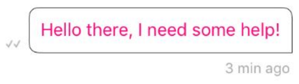
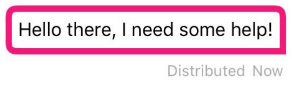
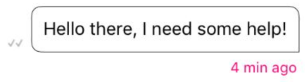
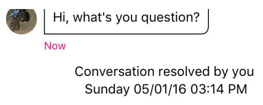
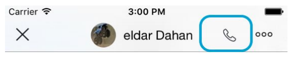
```
configuration.remoteUserBubbleBackgroundColor = UIColor.purpleColor()
configuration.brandName = “Brand Name”
configuration.remoteUserBubbleBorderWidth = 0.5
```

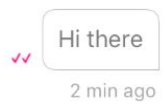
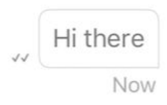
Updating an existing version of the SDK

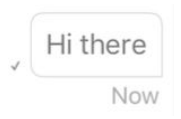
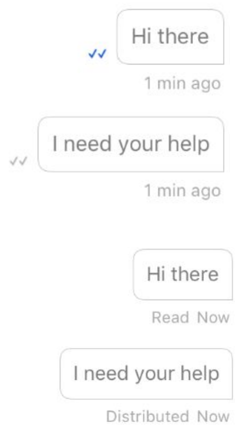
Attributes

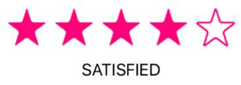
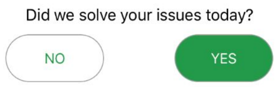
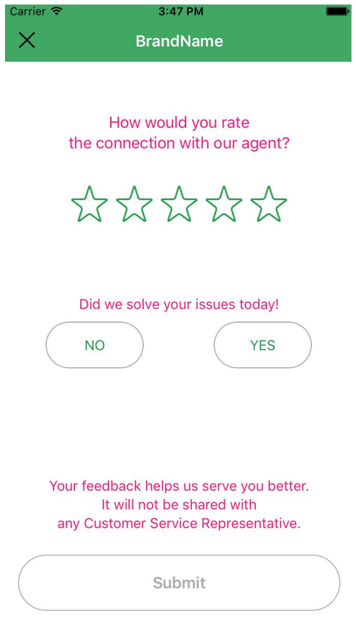
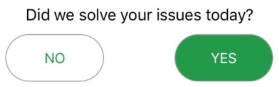
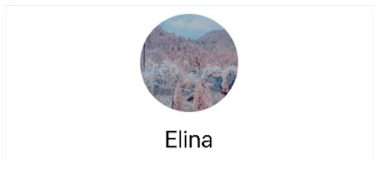
The table below lists the available attributes which can be used to personalize your app. The **Default / Customer notes** column includes space for you to add your own branding.

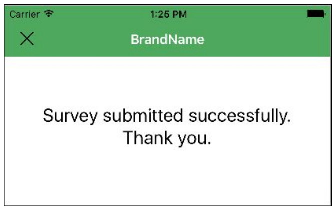
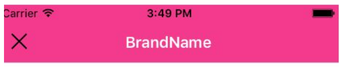
Attribute name	**Type**	**Description**	**Example**	**Default / Customer notes**
Consumer bubble				
`remoteUserBubbleBackgroundColor`	`UIColor`	Color code for the background of the remote user bubble.	 Hello, how may I assist you? 1 min ago	#0362AC
`remoteUserBubbleBorderColor`	`UIColor`	Color code for the outline color.	 Hello, how may I assist you? 4 min ago	#0362AC
`remoteUserBubbleLinkColor`	`UIColor`	Color code for links in the text of the remote user bubble.	 Hi, please visit www.liveperson.com Now	#FFFFFF
`remoteUserBubbleTextColor`	`UIColor`	Color code for the text of the remote user bubble.	 Hello, how may I assist you? 6 min ago	#FFFFFF
`remoteUserBubbleBorderWidth`	`Double`	Double number for the outline width.	 Hello, how may I assist you? Now	2
`remoteUserBubbleTimestampColor`	`UIColor`	Color code for the timestamp of the remote user bubble.	 Hello, how may I assist you? 7 min ago	#AAAAAA
`remoteUserTypingTintColor`	`UIColor`	Color of the remote user typing bubbles animation.		#FFFFFF

userBubbleBackgroundColor	UIColor	Color code for the background of the visitor bubble.	 Hello there, I need some help! Now	#FFFFFF
userBubbleBorderColor	UIColor	Color code for the outline color.	 Hello there, I need some help! 1 min ago	#BEBEBE
userBubbleLinkColor	UIColor	Color code for links in the text of the visitor bubble.	 www.liveperson.com Now	#0000ee
userBubbleTextColor	UIColor	Color code for the text of the visitor bubble.	 Hello there, I need some help! 3 min ago	#7C7C7C
userBubbleBorderWidth	Double	Double number for the outline width.	 Hello there, I need some help! Distributed Now	1
userBubbleTimestampColor	UIColor	Color code for the timestamp of the visitor bubble.	 Hello there, I need some help! 4 min ago	#AAAAAA
System messages				
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	#000000
Window mode				
customButtonImage	UIImage	[In window mode]() only:	 Carrier 3:00 PM eldar Dahan	nil

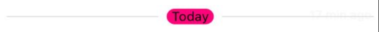
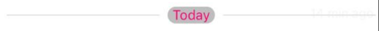
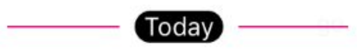
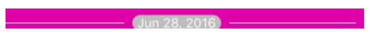
		Custom button icon image. This will be displayed on the navigation bar. When the button is pressed, a dedicated callback will be invoked. (See LPMessagingSDKCustomButtonTapped for more information).		
Delivery notifications				
`checkmarkVisibility`	**`CheckmarksState (Integer Enum)`**	Checkmark visibility 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.	###	Checkmarks State .All
`checkmarkReadColor`	**`UIColor`**	Color of checkmark indication signs of Read messages.	 Hi there 2 min ago	#004dc9
`checkmarkDistributedColor`	**`UIColor`**	Color of checkmark indication signs of	 Hi there Now	#808080

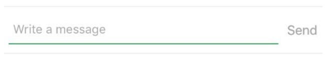
		Distributed messages.		
checkmarkSentColor	UIColor	Color of checkmark indication signs of Sent messages.		#808080
isReadReceiptTextMode	Bool	Two options for read indication: ● Read Receipt with Text Mode ● Read Receipt with Icon Mode If the parameter set as true the mode will be Text. If the parameter set as false the mode will be Icon. Default value is true.	 Hi there 1 min ago I need your help 1 min ago Hi there Read Now I need your help Distributed Now	true
Submit / Rating / Resolution buttons				
csatSubmitButtonCornerRadius	Double	Corner radius of the Submit button.		30
csatSubmitButtonBackgroundColor	UIColor	Background color code of the Submit button.		#229A49
csatSubmitButtonTextColor	UIColor	Text color code of the Submit button.		#FFFFFF

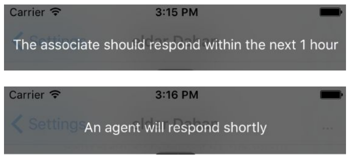
csatRatingButtonSelectedColor	UIColor	Background Color code of the rating buttons.		#229A49
csatResolutionButtonSelectedColor	UIColor	Color code for the resolution confirmation buttons (YES/NO) when selected.		#229A49
CSAT				
csatAllTitlesTextColor	UIColor	Title text color for all labels.	 Carrier 3:47 PM BrandName How would you rate the connection with our agent? Did we solve your issues today! NO YES Your feedback helps us serve you better. It will not be shared with any Customer Service Representative. Submit	#000000
csatResolutionHidden	Bool	Hides the yes/no question.		false
csatAgentViewHidden	Bool	Hides the view of agent avatar and name.		true

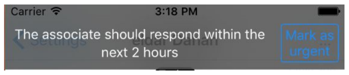
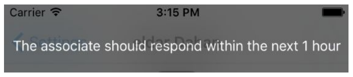
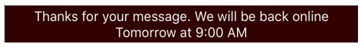
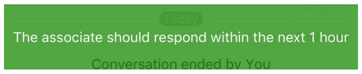
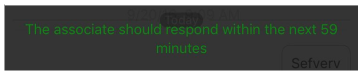
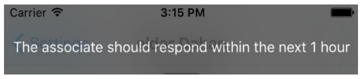
csatThankYouScreenHidden	Bool	Hides the Thank You screen after tapping Submit button.	 Survey submitted successfully. Thank you.	false
csatNavigationBackgroundColor	UIColor	Background color of the navigation of the survey.		#229A49
csatNavigationTitleColor	UIColor	Text color of the title in the survey navigation.		#FFFFFF
csatSkipButtonColor	UIColor	Skip survey button color.		#000000
csatUIStatusBarStyleLightContent	Bool	Allow the UI status bar to take the color of the survey navigation bar color.		true
csatShowSurveyView	Bool	Hides the whole survey view and disables it.		true
Conversations				
maxPreviousConversationToPresent	UInt	Number of conversations to show in advance.		2
deleteClosedConversationOlderThanMonths	UInt	Upon SDK initialization, all closed conversations with an end date older than X months will be deleted from the		13

		database. Setting 0 will delete all closed conversations.		
Localization				
`country`	`String?`	Country code: When it is not nil, it will be combined with 'language' ("language_country", for example: en_US) and used instead of device default locale when formatting date and time. If no value is provided, the SDK will use the country according to the device's locale.	###	nil
`language`	LPLanguage	Language that will be used instead of default device language. Its type is LPLanguage enum that contains all the languages that are supported by MessagingSDK. It will affect the following areas: 1. Will be used when getting localized strings 2. Will be combined with 'country' ("language_country", for example: en_US) and used instead of default device locale when	###	DeviceLanguage

		formatting time and date. If no value is provided, the SDK will use the device's language as default.		
Brand				
`brandName`	`String`	The brand name will be shown as a title on toolbar when there is no active conversation.		"BrandName "
`conversationBackgroundColor`	`UIColor`	Color code for the entire view background.		#FFFFFF
Date separator				
`dateSeparatorTitleBackgroundColor`	`UIColor`	Color code for date separator title background color.		#BEBEBE
`dateSeparatorTextColor`	`UIColor`	Color code for date separator text color.		#FFFFFF
`dateSeparatorLineBackgroundColor`	`UIColor`	Color code for date separator line background color.		#DCDCDC
`dateSeparatorBackgroundColor`	`UIColor`	Color code for date separator background color.		#FFFFFF

Send button				
`sendButtonDisabledTextColor`	`UIColor`	Color code for Send button title in disabled mode.		#AAAAAA
`sendButtonEnabledTextColor`	`UIColor`	Color code for Send button title in enabled mode.		#0362AC
Underline background				
`editTextUnderlineColor`	`UIColor`	Color code for underline background in 'edit' view.		#C4C4C4
Agent assignment				
`retrieveAssignedAgentFromLastClosedConversation`	`Bool`	When using "getAssignedAgent" method, this option lets you decide whether to get assigned agents from active conversations 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.		true
Duration of local notifications				

`notificationShowDurationInSeconds`	`Double`	Display duration of the local notification in seconds. Examples: TimeToRespond notification, local notification, etc.		3
Response time and Off hours				
`ttrShowShiftBanner`	`Bool`	Ability to enable/disable shift toaster ('An agent will respond...')		true
`ttrFirstTimeDelay`	`Double`	TTR - Time To Respond. Number of seconds before the first TTR notification appears.		10
`ttrShouldShowTimestamp`	`Bool`	TTR - Time To Respond. **<u>Enable:</u>** Displays a time stamp in the TTR notification. **<u>Disable:</u>** Displays: "An agent will respond shortly".	 The associate should respond within the next 1 hour An agent will respond shortly	false
`ttrShowFrequencyInSeconds`	`UInt`	Controls the TTR frequency: Don't show the TTR more than once in X seconds.		8

showUrgentButtonInTTRNotification	Bool	TTR - Time To Respond Enable presentation of Urgent button in the TTR notification.	 Carrier 3:18 PM The associate should respond within the next 2 hours Mark as urgent  Carrier 3:15 PM The associate should respond within the next 1 hour	false
showOffHoursBanner	Bool	Ability to disable/enable the off-hours toaster.	 Thanks for your message. We will be back online Tomorrow at 9:00 AM	true
ttrBannerBackgroundColor	UIColor	Color of background for banner.	 The associate should respond within the next 1 hour	#52A742
ttrBannerTextColor	UIColor	Text color of the banner.	 The associate should respond within the next 59 minutes	#52A742
ttrBannerOpacityAlpha	Double	Opacity level of the banner background (values: 0.0 - 1.0).	 Carrier 3:15 PM The associate should respond within the next 1 hour	0.8
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).		""
Date and Time				
lpDateFormat	String?	Custom formatting for date string (day, year..), for example: 'd MMM'. If not defined, one of the default styles will be used (see	###	nil

		timestamps formatting).		
`lpTimeFormat`	`String?`	Custom formatting for time string (hours, lpDateTimeFormat minutes..), for example: 'hh:mm a'. If not defined, one of the default styles will be used (see timestamps formatting).	###	nil
`lpDateTimeFormat`	`String?`	Custom formatting for date and time string, for example: 'EEEE MM/dd/YY hh:mm a'. If not defined, one of the default styles will be used (see timestamps formatting).	###	nil
Toast notifications				
`toastNotificationsEnabled`	`Bool`	Used for local system notifications such as 'no network'. **True:** Enable toast notifications such as offline notifications. **False:** Disable toast notifications.		false

`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.	"https://adminlogin.liveperson.net/csdr/account/%@/service/baseURI.json?version=1.0"	
User avatar				
`remoteUserAvatarBackgroundColor`	`UIColor`	Background color of the remote user's avatar.		
`remoteUserAvatarIconColor`	`UIColor`	Icon color of default remoteUser avatar.	#0362AC	#FFFFFF
`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.		nil
`csatAgentAvatarBackgroundColor`	`UIColor`	Background color of agent's default avatar in CSAT.		#0362AC
`csatAgentAvatarIconColor`	`UIColor`	Icon color of agent's default avatar in CSAT.		#FFFFFF

Data masking				
`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.		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.		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, meaning no regex. The regular expression patterns and behavior are based on **Perl's** regular expressions. See Apple Reference.		
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, meaning no regex. The regular expression patterns and behavior are based on **Perl's** regular expressions. See Apple Reference.		""
Navigation				
conversationNavigationBackgroundColor	UIColor	Background color of navigation bar in conversation screen.		#0362AC
conversationNavigationTitleColor	UIColor	Navigation title color in conversation screen.		#FFFFFF

`conversationStatusBarStyle`	`UIStatusBarStyle`	Status bar style in conversation screen.		.LightContent

Deprecated Attributes

Listed below are attributes that have been deprecated. These keys should no longer be used as LPConfig files, but rather as Localised strings keys. The names of these keys in the Localisation files remain the same as in the LPConfig files.

In order to configure different string for this key, see String localization in SDK.

Note: By default, these keys are configured with Localization keys values, and are sensitive to any language changes. These keys must be reconfigured each time a language is changed.

Attribute name	**Type**	**Description**
`customButtonDescription`	`String`	Accessibility voiceover string for the custom button.
`readReceiptTextDistributed`	`String`	Text for distributed indication.
`csatResolutionFeedbackText`	`String`	Text for the feedback label.
`csatResolutionQuestionText`	`String`	Text for the resolution confirmation question.
`readReceiptTextSent`	`String`	Text for sent indication.
`readReceiptTextRead`	`String`	Text for read indication.

String Localization in SDK

The SDK contains language folders for each language supported. For a list of supported languages, see LiveEngage System Requirements and Language Support. Each folder contains LPLocalizable.strings file, where all strings are located for a specific language.

The example below contains four language folders:

- en.lproj: strings in English. Used in case the host app/device language is not supported by us.
- fr.lproj: strings in French.
- pt.lproj: strings in Portuguese.
- pt-PT.lproj: strings in Portuguese (Portugal).

The SDK allows you to override the string localization of any supported language in LiveEngage. To apply a custom localization files with your own strings, create LPLocalizable.strings files for relevant languages inside your app, which will include the keys you would like to override.

Example: Overriding the SDK string of ‘send’ in English:

```
    1.  Create in your app new localization Base file called LPLocalizable.strings which will
        include your supported language: ‘New file…’ -> ‘Strings file’ -> ‘Create’
    2.  Add new key: “send” = “<ANY NEW VALUE>”;
    3.  Mark this LPLocalizable.strings file as localized: Tap on LPLocalizable.strings file ->
        Open the file inspector -> Go to ‘Localization’ section -> Tap ‘Localize…’
    4.  Adding a new language support
        - Open project settings
        - Info tab
        - Under Localizations, press ‘+’ sign
        - Choose the new language you would like to support.

        Attach a Strings file to existing supported languages:
        - Choose the desired strings file
        - Open the file inspector
        - Under localizations, choose the relevant languages.

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

In order to print all localized keys with a default English value in the SDK, call the following:

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

In order to get or print all localized supported languages in the SDK, call the following:

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

In order to get all supported languages by key-value (key is locale key and value is language description), call the following:

```
LPMessagingSDK.instance.getAllSupportedLanguages()
```

Note: The SDK uses native iOS control for the Copy/Paste menu. The language of the menu control is defined by the current device language if the host app is localized for this language. If the host app does not have this language, the last menu control language will be used.

Localization Keys

Listed below are all localized keys with the English localization string.

Attribute name	**String Localization**
`customButtonAccessibilityDescription`	"Call"
`readReceiptTextDistributed`	“Distributed”
`conversationEndedByAgent`	"Conversation resolved by %@ \n %@"
`conversationEndedByAgentWithoutName`	"Conversation resolved by Agent \n %@"
`conversationEndedByYou`	"Conversation resolved by You \n %@"
`cancel`	"Cancel"
`hiMessage`	"How can I help you today?"
`sentMessageErrorTitle`	"Message was not sent"
`retry`	"Retry"
`sentMailErrorTitle`	"Mail not sent"
`sentMailErrorMessage`	"Email could not be sent. Please try again later."
`callTo`	"Call %@"
`sendMessage`	"Send Message"
`facetimeAudio`	"FaceTime Audio"
`sendEmailTitle`	"Send Email"

sendEmailMessage	"Would you like to send an email to %@?"
notNow	"Not now"
yes	"Yes"
feedbackYesButtonTitle	"Yes"
feedbackNoButtonTitle	"No"
cantSendMail	"Can't send email"
noEmailOnDeviceError	"No email is defined on this device"
dismissUrgent	"Dismiss urgency"
dismissUrgentConfirmation	"Are you sure you want to mark this conversation as not urgent?"
dismiss	"Dismiss"
markAsUrgent	"Mark as urgent"
markAsUrgentConfirmation	"Are you sure you want to mark this conversation as urgent?"
urgent	"Urgent"
noInternetConnection	"No Internet Connection"
ok	"OK"
markAsEndedConfirmation	"Are you sure this topic is resolved?"
endTheConversation	"Resolve the conversation"
ttrResponseMessage	"An agent will respond within the next %@"
rateConnectionWithAgent	"How would you rate your\nconnection with our agent?"
rateConnectionWithName	"How would you rate your\nconnection with %@?"
veryDissatisfied	"Very Dissatisfied"
dissatisfied	Dissatisfied
neither	"Neither"
satisfied	"Satisfied"
verySatisfied	"Very Satisfied"
writeMessage	"Write a message"

`olderMessages`	"Older messages"
`numberOfDays`	"%d days"
`day`	"1 day"
`numberOfHours`	"%d hours"
`oneHour`	"1 hour"
`oneMinute`	"1 minute"
`numberOfMinutes`	"%d minutes"
`minutesAgo`	"%d min ago"
`send`	"Send"
`submit`	"Submit"
`today`	"Today"
`noSocketConnectivity`	"No connectivity to server\nPlease try again"
`conversationAlreadyEnded`	"This conversation has already been resolved."
`loadingHistoryMessages`	"Loading..."
`and`	"and"
`anAgentWillRespondShortly`	"An agent will respond shortly"
`nowTimestamp`	"Now"
`surveySubmittedSuccessfullyThankYou`	"Survey submitted successfully.\nThank you!"
`ttrOffHoursResponseMessageToday`	"Thanks for your message. We will be back online Today at %@"
`ttrOffHoursResponseMessageTomorrow`	"Thanks for your message. We will be back online Tomorrow at %@"
`ttrOffHoursResponseMessageDate`	"Thanks for your message. We will be back online %@"
`maskedMessageResendError`	"Message failed to send. Please re-enter message and send again."
`systemMessageRealTimeMasked`	"Your personal data has been masked to protect your security and cannot be read by the agent."

`systemMessageClientOnlyMasked`	"Your personal data has been masked to protect your security. Only the agent can read it."
`offlineNotification`	"Offline. Check your internet connection."
`resolve`	"Resolve"

More information regarding strings with parameters can be found in Appendix A: Localization Strings.

Locale - Timestamps Formatting

Time/Date Styles

The iOS platform provides four different default types of date and time styles:
SHORT is completely numeric.
MEDIUM is longer and contains the first 3 letters of the month.
LONG is longer and contains the full month name.
FULL specifies the complete time and date.

iOS examples:

SHORT 12/13/52 or 3:30 PM

MEDIUM Jan 12, 1952

LONG January 12, 1952 or 3:30:32 PM

FULL Tuesday, April 12, 1952 AD or 3:30:42 PM PST.

The LPMessagingSDK uses default styles. Each feature has its own style. The style is flexible and adapts the 'locale' configuration of the device.
Example: US locale SHORT date is displayed as "9/25/16", whereas Japanese locale SHORT date is displayed as "2016/9/25".

A specific 'locale' which is different from the device locale can be set through the language and country configurations.
Example: country: String?, language: String?

Time/Date formats:

It is also possible to configure a specific format for time and date instead of style.
If the host app has configured its own formatting, this formatting will be used instead of style (and therefore will not be affected by 'locale').

To configure the formatting by the host app, three configurable formatting resources have been added:

1. For date only (separator):

   lpDateFormat: String?

2. For time only (bubble's timestamp & off hours time in case of today/tomorrow):

   lpTimeFormat: String?

3. For date & time together (resolve message & off hours time in case of other date):

   lpDateTimeFormat: String?

Off Hours

Date & Time

- The Today and tomorrow off hours message uses the default SHORT time without date according to the locale (default or custom).

If the device is set to 12 hour format :
“Thanks for your message. We will be back online today/tomorrow at *3:30 PM*”
If the device is set to 24 hour format :
“Thanks for your message. We will be back online today/tomorrow at *15:30*”

To use a different time format, you can use `lpTimeFormat: String?` with any time format.
Example: “hh:mm a”, “HH:mm”, etc.

- The Date off hours message (not today/tomorrow) uses default LONG date and SHORT time according to the locale (default or custom).

If the device is set to 12 hour format :
“Thanks for your message. We will be back online *January 12, 2017* at *3:30 PM*”
If the device is set to 24 hour format :
“Thanks for your message. We will be back online *January 12, 2017* at *15:30*”

To use a different date/hour format, you can use `lpDateFormat: String?` with any date & time format.
Example: “MMM d, yyyy hh:mm a”, “EEEE dd/mm/yy HH:mm” etc.

Timezone

Off hours can appear in different time zones with this resource ID :
`offHoursTimeZoneName: String = ""`
A list of timezone IDs can be found here.
Example: ”US/Pacific”, “Europe/Berlin”.

Bubble timestamp

The Bubble timestamp contains the time only in the SHORT time format, according to the locale (default or custom) and to the device setting.

If the device is set to 12 hours format : “3:30 PM”
If the device is set to 24 hours format : ”15:30”

If you wish to configure this time format, override the resource ID `lpTimeFormat: String?` with any time format.
Example: “hh:mm a”, “HH:mm” etc.
Note: This applies to all bubble timestamps.

Separator timestamp

The Separator timestamp contains the date only in the 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 the resource ID `lpDateFormat: String?` with any date format.
Example: “MMM d, yyyy”, “EEEE dd/mm/yy” etc.

Resolve message

The Resolve message uses the default SHORT date and SHORT time according to the locale (default or custom) and to the device setting.

If the device is set to 12 hour format (US locale):
    “Conversation resolved by [agent name] \n 9/25/16, 3:30 PM”
If the device is set to 24 hour format (US locale):
    “Conversation resolved by [agent name] \n 9/25/16, 15:30”

To use a different date/hour format, you can use `lpDateTimeFormat: String?` with any date & time format.
Example: “MMM d, yyyy hh:mm a”, “EEEE dd/mm/yy HH:mm” etc.
Limitation: If the formatting is changed after the Resolve messages already appears, this change will take no effect. This will be fixed in future versions.

OS Certificate Creation

In order for the push notification to work, you will need two .pem files:

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

Creating a Certificate Signing Request file

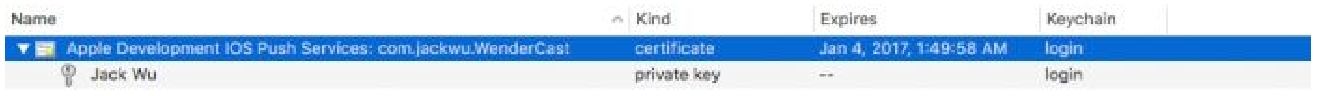
With this file, we will create both a .p12 file and a .crt file.

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.

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

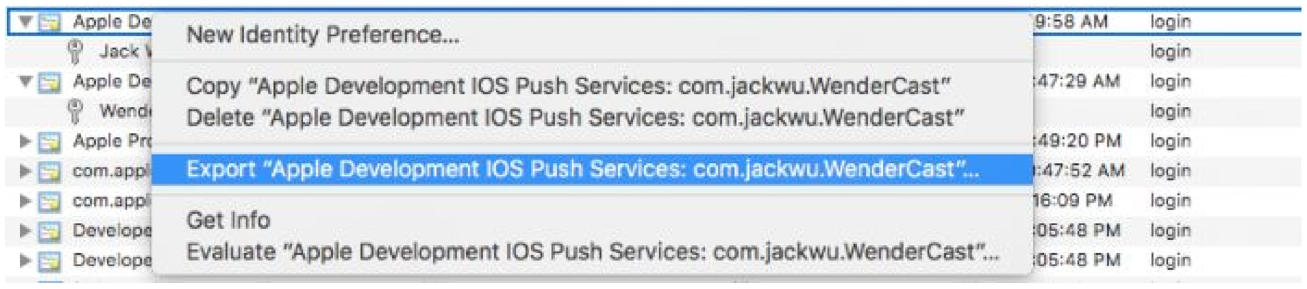
Within the Keychain Access dropdown menu, select **Keychain Access** > **Certificate Assistant** > **Request a Certificate from a Certificate Authority.**

2. 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, for example,John Doe Dev Key.
    - The CA Email Address field should be left empty.
    - In the Request is group, select the Saved to disk option.
3. In Keychain Access, click **Continue** to complete the CSR generating process.
4. Download and run the certificate. The certificate is now added to your Keychain, paired with a private key:

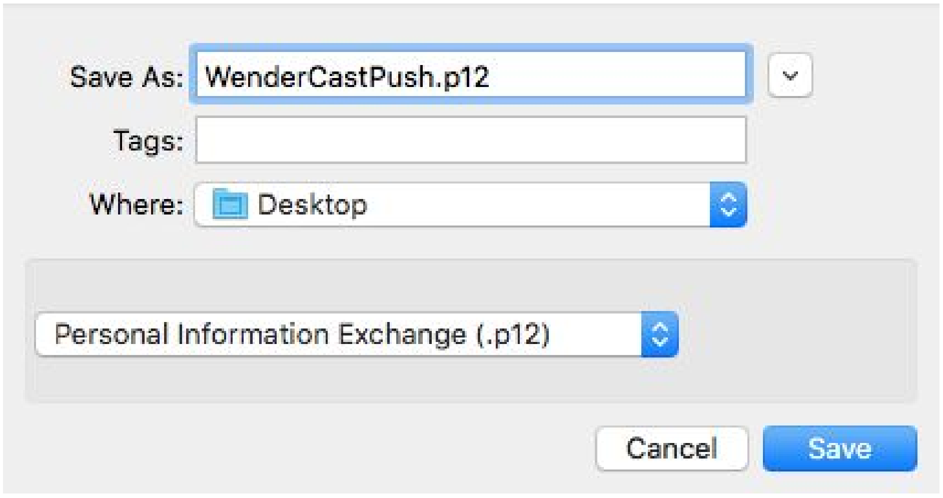


Name	Kind	Expires	Keychain
Apple Development IOS Push Services: com.jackwu.WenderCast	certificate	Jan 4, 2017, 1:49:58 AM	login
Jack Wu	private key	--	login

Creating a key .p12 file


New Identity Preference...
Copy "Apple Development IOS Push Services: com.jackwu.WenderCast"
Delete "Apple Development IOS Push Services: com.jackwu.WenderCast"
Export "Apple Development IOS Push Services: com.jackwu.WenderCast"...
Get Info
Evaluate "Apple Development IOS Push Services: com.jackwu.WenderCast"...

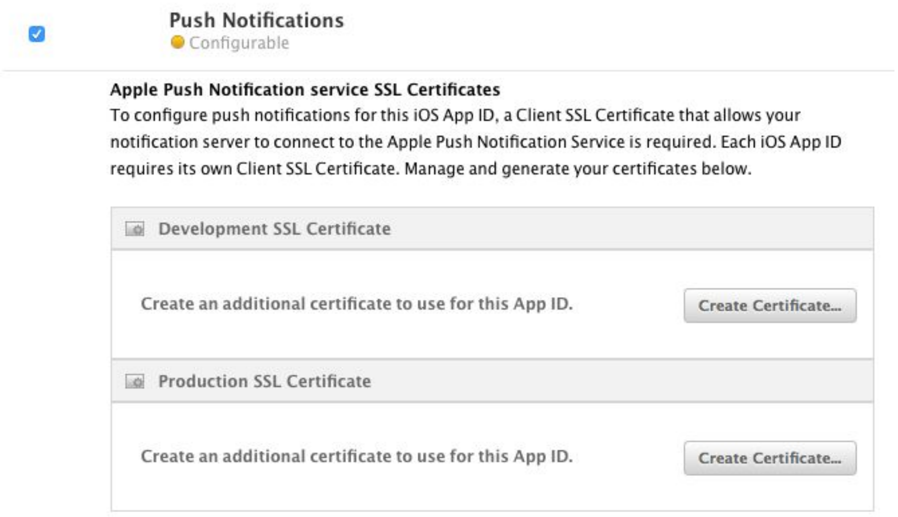
1. Right-click on your new push certificate and choose **Export**.


Save As: WenderCastPush.p12
Tags:
Where: Desktop
Personal Information Exchange (.p12)
Cancel
Save

2. Save the certificate as pushNotification.p12 as a .p12 file.
3. 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)

1. In the iOS member area, go to your app area.



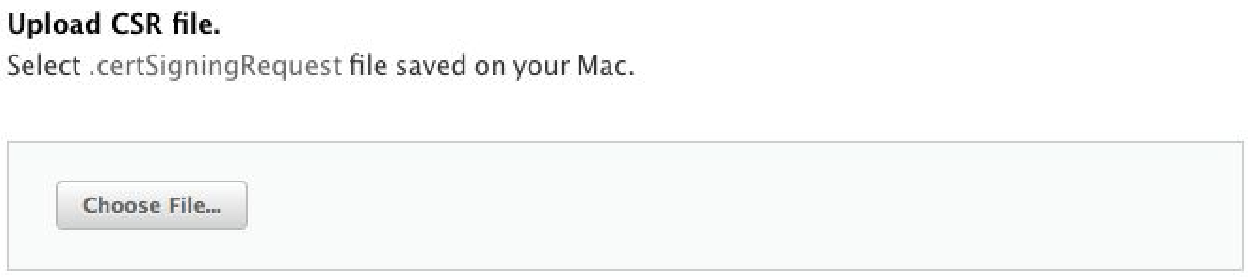
Push Notifications
Configurable

Apple Push Notification service SSL Certificates
To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

Development SSL Certificate
Create an additional certificate to use for this App ID.

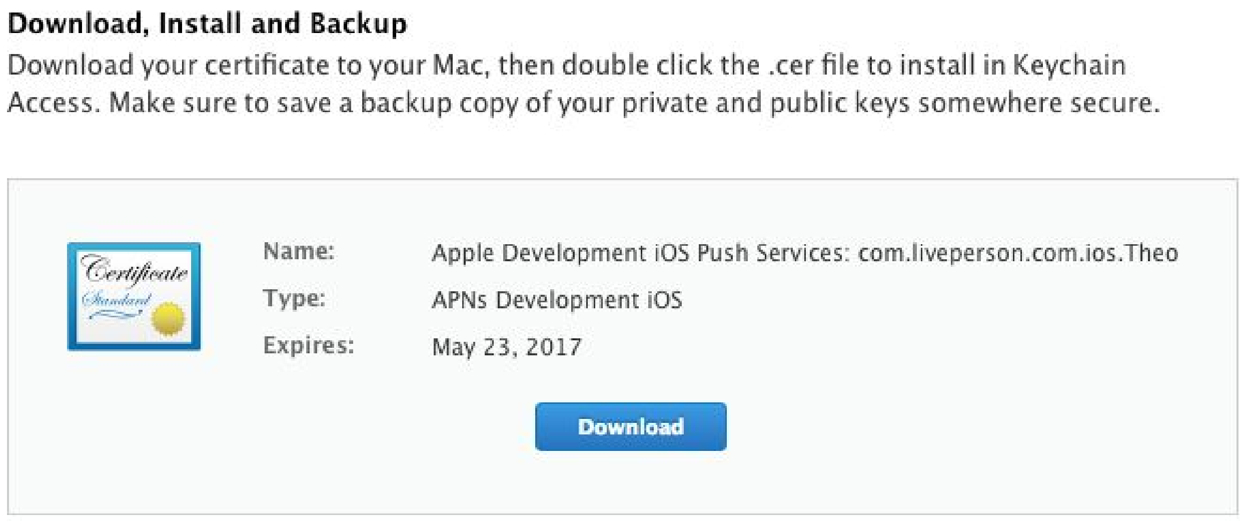
Production SSL Certificate
Create an additional certificate to use for this App ID.

2. Under Push Notification service SSL Certificates, select **Create Certificate** (Developer or Production).


Upload CSR file.

Select .certSigningRequest file saved on your Mac.

3. Choose the .csr file that you created in the previous stage.


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

4. Download the file.This file will be used to create the .pem format certificate.

Creating both key.pem file and cert.pem file

Note: This is used when configuring LiveEngage Push Notification.

1. Open the terminal and locate the folder in which you would like to save the file.

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

3. Convert the private keys .p12 file into a .pem file:
   openssl pkcs12 -nocerts -out keyWithPassword.pem -in key.p12
   You will be prompted to enter a passphrase for this file. Enter any password and remember it for the next step.

4. RSA .pem key (no password)

openssl rsa -in hostkey.pem -out hostkey.pem
You will be prompted to enter a passphrase. Enter the same passphrase you used in the previous step.

LiveEngage Configuration

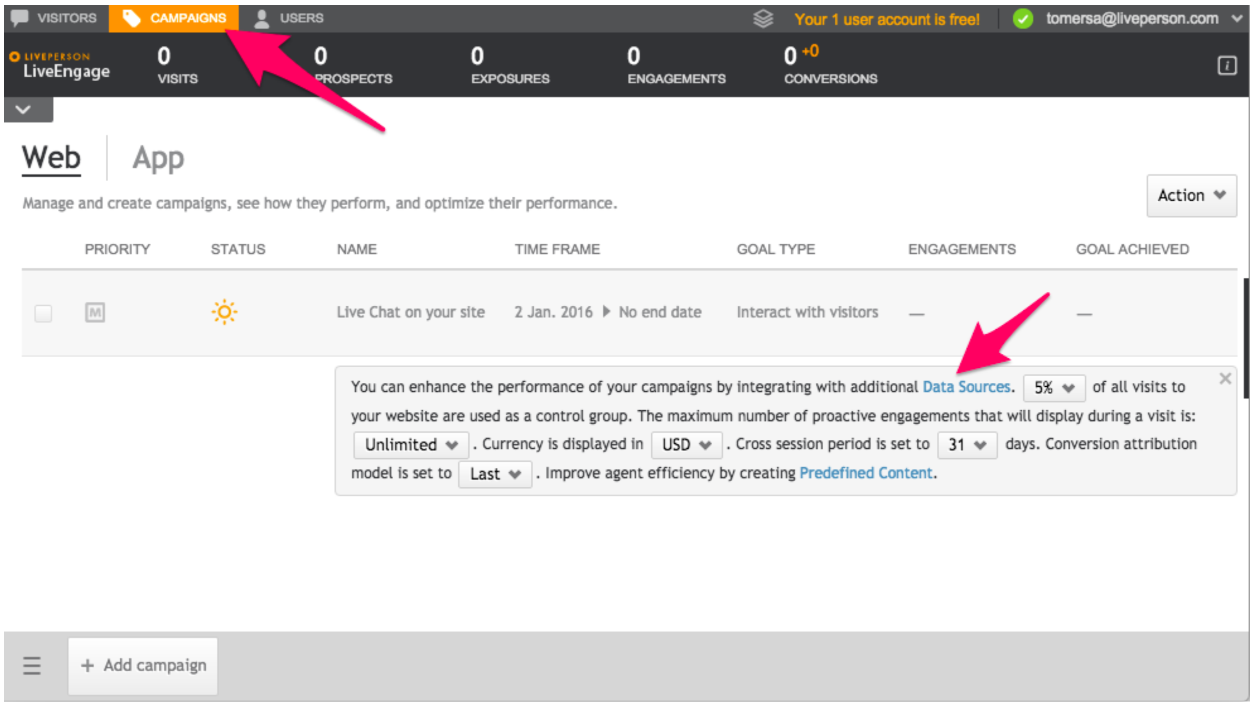
Push Notifications

Follow the instructions below to set up your app key to enable push notifications.

Note: Before you begin the setup, you must ensure your LiveEngage account is configured and connected to the SDK.

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

2. Within LiveEngage, navigate to **Campaigns.**

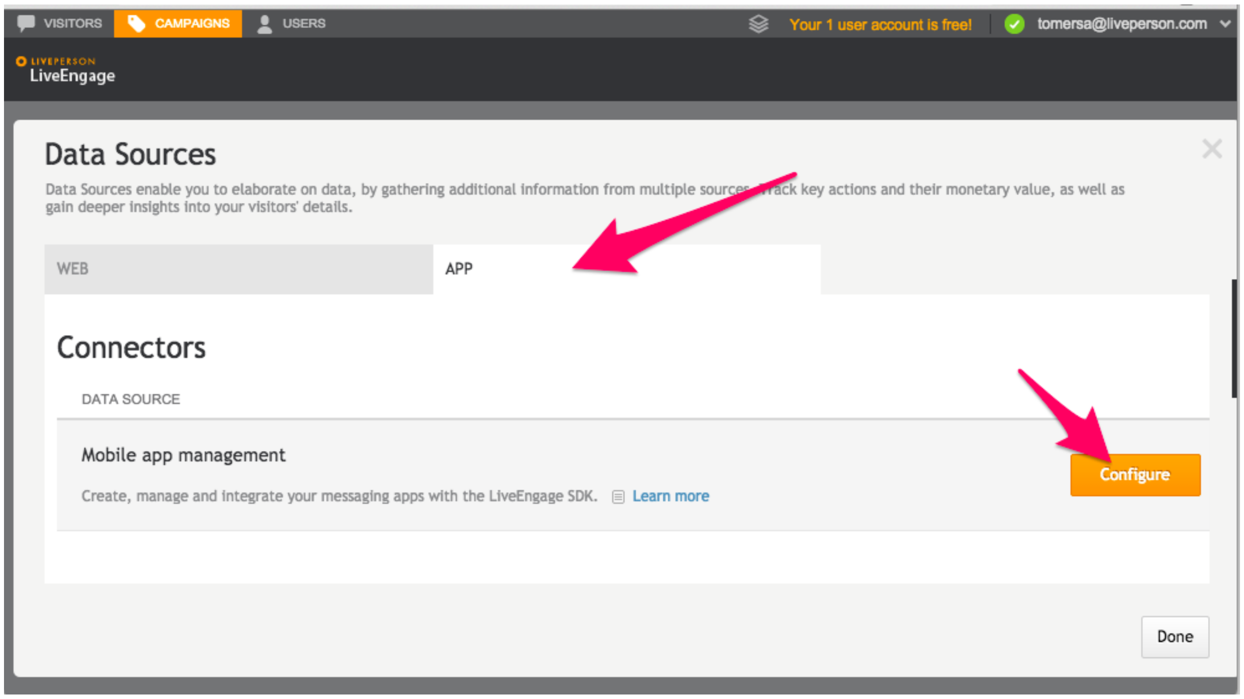


Manage and create campaigns, see how they perform, and optimize their performance.

PRIORITY	STATUS	NAME	TIME FRAME	GOAL TYPE	ENGAGEMENTS	GOAL ACHIEVED
		Live Chat on your site	2 Jan. 2016 ▶ No end date	Interact with visitors	—	—

You can enhance the performance of your campaigns by integrating with additional Data Sources. 5% of all visits to your website are used as a control group. The maximum number of proactive engagements that will display during a visit is: Unlimited. Currency is displayed in USD. Cross session period is set to 31 days. Conversion attribution model is set to Last. Improve agent efficiency by creating Predefined Content.

3. Click **Data Sources**, and then select **App**.



Data Sources

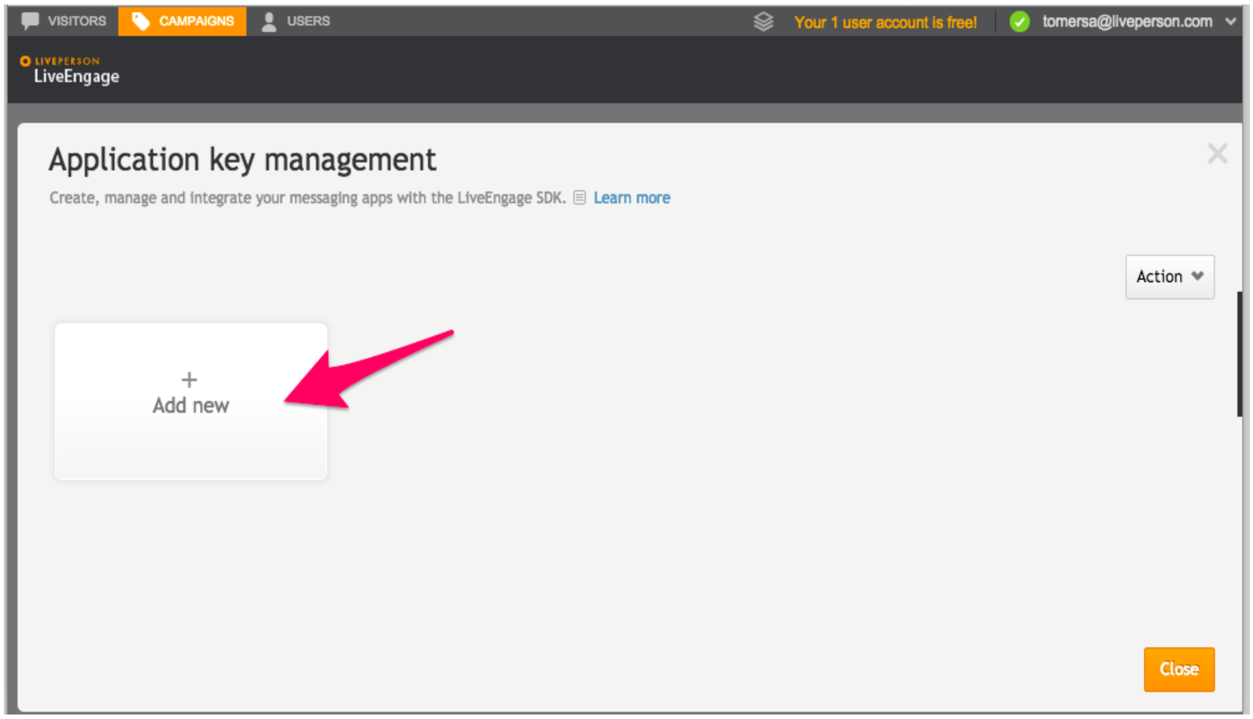
Data Sources enable you to elaborate on data, by gathering additional information from multiple sources. Track key actions and their monetary value, as well as gain deeper insights into your visitors' details.

Connectors

Mobile app management

Create, manage and integrate your messaging apps with the LiveEngage SDK. Learn more

4. Click **Configure**.



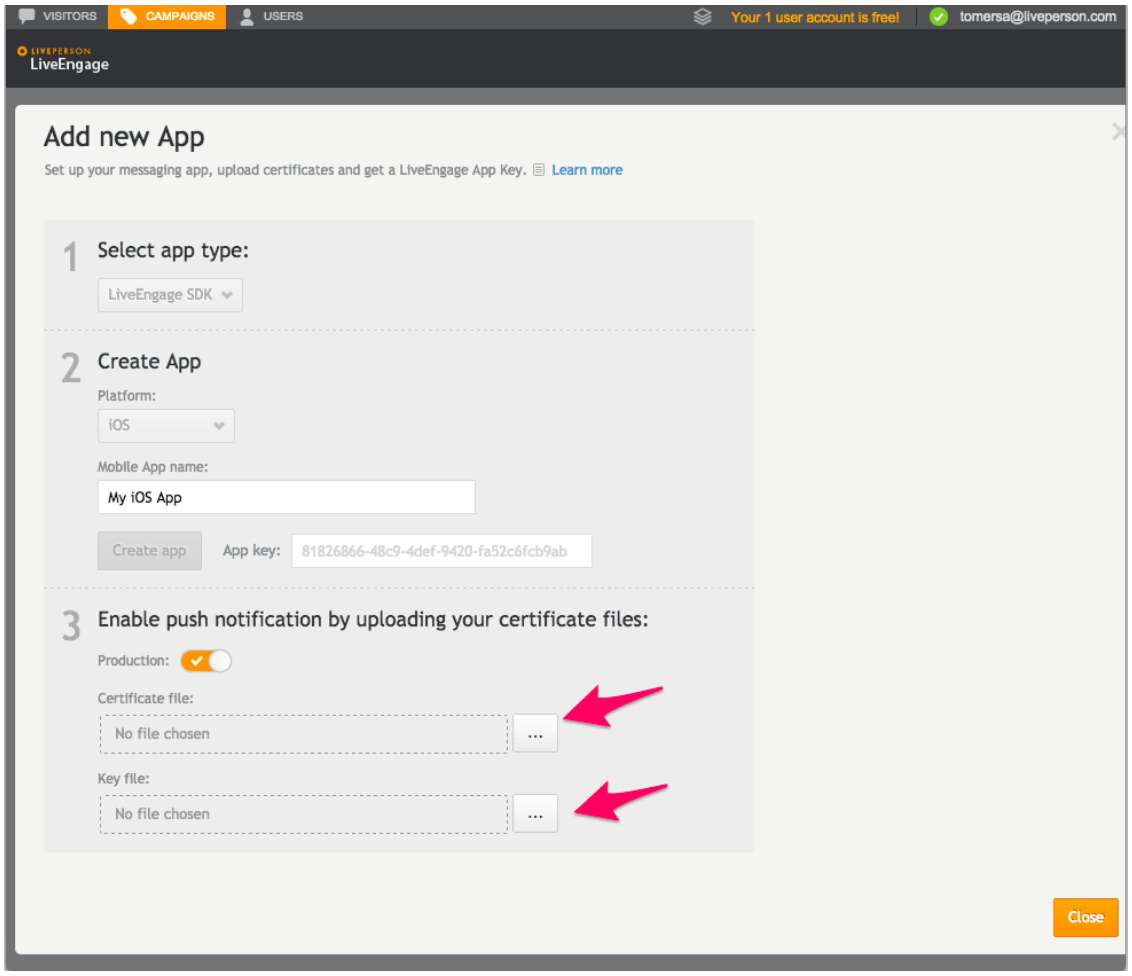
Application key management

Create, manage and integrate your messaging apps with the LiveEngage SDK. Learn more

5. Click **Add new** to associate your app with the LiveEngage account.

6. 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.

Note: 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.



Add new App

Set up your messaging app, upload certificates and get a LiveEngage App Key. Learn more

1 Select app type:

LiveEngage SDK

2 Create App

Platform: iOS

Mobile App name: My iOS App

Create app App key: 81826866-48c9-4def-9420-fa52c6fcb9ab

3 Enable push notification by uploading your certificate files:

Production: ✓

Certificate file: No file chosen ...

Key file: No file chosen ...

Close

7. Click Close to complete 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
SocketRocket	BSD
UIRefreshControl+UITableView	MIT
TTTAttributedLabel	Apache
NSDate+Extension	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.

Appendix A: Localization Strings

Strings with parameters

"**ttrResponseMessage**"
"The associate should respond within the next %@";

- Message which appears inside the toast when sending first message.
- Parameter container will be filled with the date which is combined from the following string keys:
  "numberOfDays" = "%d days";
  "day" = "day";
  "numberOfHours" = "%d hours";
  "oneHour" = "1 hour";
  "oneMinute" = "1 minute";
  "numberOfMinutes" = "%d minutes";
  "and" = "and";

Examples:
"The associate should respond within the next 9 hours and 30 minutes“.
"The associate should respond within the next 2 days 2 hours and 30 minutes“.
"The associate should respond within the next 2 hours“.
"The associate should respond within the next day 1 hour and 1 minute“.

"**ttrOffHoursResponseMessageToday**"
"Thanks for your message. We will be back online Today at %@";

- Message which appears inside the toast when agents are in off hours.
- Parameter container will be filled with the time according to device/custom formatting.

Example: “Thanks for your message. We will be back online Today at 2:48 PM”.

"**ttrOffHoursResponseMessageTomorrow**"
"Thanks for your message. We will be back online Tomorrow at %@";

- Message which appears inside the toast when agents are in off hours.
- Parameter container will be filled with the time according to device/custom formatting.

Example: “Thanks for your message. We will be back online Tomorrow at 05:48 PM”.

"**ttrOffHoursResponseMessageDate**"
"Thanks for your message. We will be back online %@";

- Message which appears inside the toast when agents are in off hours.
- Parameter container will be filled with the date and time according to device/custom formatting.

Example: “Thanks for your message. We will be back online September 21, 2016 at 2:48 PM”.

"**conversationEndedByAgent**"
"Conversation ended by %@ \n %@";

- System message which is created when the conversation was resolved and includes two lines of text.
- First parameter: Agent’s name as received from server.
- Second parameter: Date and time according to device/custom formatting.

Example: “Conversation ended by Elina \n 9/12/16, 5:05 PM”.

"**conversationEndedByYou**"
"Conversation ended by You \n %@";

- The same as ‘conversationEndedByAgent’, except the conversation was resolved by the user and not by the agent.

"**conversationEndedByAgentWithoutName**"
"Conversation ended by Agent \n %@";

- System message which is created when the conversation was resolved and includes two lines of text, but the name of the agent is not available.
- Parameter: Date and time according to device/custom formatting.

Example: “Conversation ended by Agent \n 9/12/16, 5:05 PM”.

"**callTo**"
"Call %@";

- Parameter: Phone number.

"**sendEmailMessage**"
"Would you like to send an email to %@?";

- Parameter: Email address.

"**rateConnectionWithName**"
"How would you rate\nthe connection with %@?";

- Message which appears in CSAT and includes two lines of text.
- Parameter: Agent’s name as received from the server.

"**minutesAgo**"
"%d min ago";

- Text which appears under the user’s message bubble and was posted less than an hour previously.
- Parameter: Number of minutes.

Example: “3 min ago”

Strings without parameters

"nowTimestamp"
"Now";

"readReceiptTextSent"
"Sent";

- Text which appears in the label under the user’s message bubble if it is configured to display as text and not as a UI checkmark.
- If the message was sent less than one minute previously, this label will be near the label which includes text from the key “nowTimestamp”.
  Example: ‘Sent Now’.
- If the message was sent less than one hour previously, this label will be near the label which includes text from the key “minutesAgo”.
  Example: ‘Sent 5 min ago’.

"readReceiptTextDistributed"
"Distributed";

- Same logic as “readReceiptTextSent”.

"readReceiptTextRead"
"Read";

- Same logic as “readReceiptTextSent”.