LiveEngage Enterprise
In-App Messenger SDK: iOS

Deployment Guide

Version 1.1.85

2016

Table of Contents

Introduction

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

Platform Support

- **Supported OS**: iOS 8+
- **Certified devices**: iPhone 6s+, iPhone 6s, iPhone 6+, iPhone 6, iPhone 5s, iPhone 5, iPhone 4s

Deployment:

- Embeddable library for iOS: Xcode
- Installers: Manual

Security

Security is a top priority and key for enabling trusted, meaningful engagements. LivePerson's comprehensive security model and practices were developed based on years of experience in SaaS operations, close relationships with Enterprise customers' security teams, frequent assessments with independent auditors, and active involvement in the security community.

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

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

Deploying the App Messaging SDK

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

- Download and unzip the SDK
- Set up the SDK package in Xcode
- Configure project settings
- Initialization
- Objective-C configuration
- Build and test the SDK

To deploy the App Messaging SDK:

Download and unzip the SDK

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

Set up the SDK package in Xcode

1. In Xcode, from the menu, select **File** > **New** >**New project**.
2. From the list of templates, select **Single View Application**, and then click **Next**.
3. Complete the following fields:
    - Product Name
    - Organization Identifier
    - Select Swift or Objective-C
4. Click **Next.**
5. Save the project to a folder of your choosing.
6. On the project explorer pane, navigate to the main folder for your project. Right-click it, and select "**Add Files to**..." Navigate to the folder where you extracted the SDK package files, and then add the files to the lib subfolder in your project.

Configure project settings

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

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.

To initialize the SDK using Objective-C:

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

```
[[LPMessagingSDK instance] initialize];
```

2. In order to create/view the conversation page, run the following code:

```
id <ConversationParamProtocol> conversationQuery = [[LPMessagingSDK instance]
getConversationBrandQuery:accountNumber];
[[LPMessagingSDK instance] showConversation:conversationQuery authenticationCode:@"Your
authentication code" containerViewController:self];
```

3. Inside `AppDelegate, add:`

```
#import <LPMessagingSDK/LPMessagingSDK.h>
#import <LPAMS/LPAMS.h>
#import <LPInfra/LPInfra.h>
```

4. In build settings, make sure of the following:
    - “Embedded content contains Swift code” is set to Yes.

5. In general tab, make sure that the framework files are under ‘Embeded Libraries’.

To initialize the SDK using Swift:

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

```
LPMessagingSDK.instance.initialize()
```

2. In order to create/view the conversation page, run the following code:

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

3. Inside AppDelegate add:

```
import LPMessagingSDK
```

Objective-C configuration

1. In your app delegate:

```
#import <LPMessagingSDK/LPMessagingSDK.h>
#import <LPInfra/LPInfra.h>
#import <LPAMS/LPAMS.h>
```

2. In build settings:
   - Make sure “Embedded content contains Swift code” is set to Yes.

3. In general tab, make sure that the framework files are under ‘Embeded Libraries’.

Build and test the SDK

That’s it! You are now ready to run the app with a basic implementation of our SDK. Keep reading to find out what else you can do with our SDK.

Advanced options

Push registration

1. Register to LPMessagingSDK push notification with the following code in `AppDelegate`:

Objective-C:

```
[[LPMessagingSDK instance] registerPushNotifications:deviceToken
notificationDelegate:self];
```

Swift:

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

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

}
```

2. Handle remote notifications as follows:

Objective-C:

```
[[LPMessagingSDK instance] handlePush:userInfo];
```

Swift:

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

LPMessagingSDK.instance.handlePush(userInfo)

    }
```

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

Objective-C:

```
[((AppDelegate *) [[UIApplication sharedApplication]
delegate]).mainViewController.centerViewController navigateTo:@"chatView"
data:@{@"brand":@YES}];
```

Swift:

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

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

    }
```

API Methods

Check if the SDK is ready (connected to internet and connected to server)

Objective-C:

```
[[LPMessaging instance] isSDKReady];
```

Swift:

```
LPMessagingSDK.instance.isSdkReady()
```

Set a user profile

Objective-C:

```
    LPUser *user = [[LPUser alloc] initWithFirstName:@”First name” lastName:@”Last name” uid:nil profileImageURL:@"Image url" phoneNumber:@"000-0000000"];
    [[LPMessagingSDK instance] setUserProfile:user accountID:@”Account ID”];
```

Swift:

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

Reconnect when token expires

Objective-C:

```
id <ConversationParamProtocol> query = [[LPMessagingSDK instance] getConversationBrandQuery:account];
[[LPMessagingSDK instance] reconnect:conversationQuery authenticationCode:@"Your authentication code"];
```

Swift:

```
let query = LPMessagingSDK.instance.getConversationBrandQuery("brandID")
LPMessagingSDK.instance.reconnect(query, authenticationCode: "Authentication code")
```

Set a custom button that will call a delegate to your project (usually used to call a client center)

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

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

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

Subscribe to log events (Trace, Debug, Info, Warning, Error)

Objective-C:

```
[[LPMessagingSDK instance] subscribeLogEvents:LogLevelINFO logEvent:^(LPLog *log){

}];
```

Swift:

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

Callbacks

1. protocol LPMessagingSDKdelegate
    a. LPMessagingSDKCustomButtonTapped()
    b. LPMessagingSDKAgentDetails(agent: LPUser)
    c. LPMessagingSDKActionsMenuToggled(toggled: Bool)
    d. LPMessagingSDKHasConnectionError(error: String?)
    e. LPMessagingSDKObseleteVersion(error: NSError)
    f. LPMessagingSDKAuthenticationFailed(error: NSError)
    g. LPMessagingSDKTokenExpired(brandID: String)
    h. LPMessagingSDKAgentIsTypingStateChanged(isTyping: Bool)

2. protocol LPMessagingSDKNotificationDelegate

a. LPMessagingSDKNotification(didReceivePushNotification notification: LPNotification)
b. LPMessagingSDKNotification(shouldShowPushNotification notification: LPNotification) -> Bool
c. LPMessagingSDKNotification(customLocalPushNotificationView notification: LPNotification) -> UIView
d. LPMessagingSDKNotification(notificationTapped notification: LPNotification)

Configuring the SDK

The SDK allows you to configure the look and feel of your app with your `configFile.plist` file. This file MUST contain all the exact resource names as listed below:

General

Resource name	Description
`brand_name`	The brand name will be shown as a title on toolbar when there is no active conversation.
`language`	The language is defined by a two-letter ISO 639-1 language code, for example, “en” for English. If no value is provided, the SDK will use the language according to the device's locale.
`country`	Country code. If no value is provided, the SDK will use the country according to the device's locale.
`conversation_background`	Color code for the entire view background.
`TTRfirstTimeDelay`	TTR - Time To Respond Number of seconds before the first TTR notification appears
`show_urgent_button_in_ttr_notification`	TTR - Time To Respond Enable presentation of ‘Urgent’ button in the TTR notification

URLs

Resource name	Description
`csds_domain`	CSDS Domain URL

Custom Button

Resource name	Description
`custom_button_icon_description`	Accessibility voiceover string for the custom button.

`custom_button_icon_name`	Custom button icon filename without extension. This will be displayed on the navigation bar.

Agent Message Bubble

Resource name	Description
`agent_bubble_stroke_width`	Int number for the outline width.
`agent_bubble_stroke_color`	Color code for the outline color.
`agent_bubble_message_text_color`	Color code for the text of the agent bubble.
`agent_bubble_message_link_text_color`	Color code for links in the text of the agent bubble.
`agent_bubble_timestamp_text_color`	Color code for the timestamp of the agent bubble.
`agent_bubble_background_color`	Color code for the background of the agent bubble.

Visitor Message Bubble

Resource name	Description
`visitor_bubble_message_text_color`	Color code for the text of the visitor bubble.
`visitor_bubble_stroke_width`	Int number for the outline width.
`visitor_bubble_stroke_color`	Color code for the outline color.
`visitor_bubble_message_link_text_color`	Color code for links in the text of the visitor bubble.
`visitor_bubble_timestamp_text_color`	Color code for the timestamp of the visitor bubble.
`visitor_bubble_background_color`	Color code for the background of the visitor bubble.

System messages

Resource name	Description
`system_bubble_text_color`	Color code for the text of the system messages.

Checkmarks Visibility

Resource name	Description
`message_receive_icons`	Int number representing number of read indications
`checkmarks_color`	Color of read indication signs
`readReceipt_distributed`	Text for distributed indication
`readReceipt_read`	Text for read indication
`readReceipt_sent`	Text for sent indication
`read_receipt_mode`	Two options for read indication: • read_receipt_mode_text • read_receipt_mode_icon

Customer Satisfaction

Resource name	Description
`csat_submit_button_corner_radius`	Corner radius of the Submit button
`csat_submit_button_background_color`	Background color code of the Submit button
`csat_submit_button_text_color`	Text color code of the Submit button
`csat_rating_button_selected_color`	Background Color code of the rating buttons
`csat_resolution_button_selected_color`	Color code for the resolution confirmation buttons (YES/NO) when selected
`csat_resolution_feedback_text`	Text for the feedback label

`csat_resolution_question_text`	Text for the resolution confirmation question
`csat_all_titles_text_color`	Titles text colors for all labels

Conversation Settings

Resource name	Description
`max_conversations_to_fetch`	The amount of conversations to fetch on loading
`max_previous_conversations_to_present`	Amount of conversations to show in advance

Open source list

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