SALESmanago

Setting up the Mobile Push channel

Integrating SALESmanago with Your Mobile App: iOS

Contents

1. Prerequisites

When integrating your mobile app with SALESmanago, bear in mind the following requirements:

- **Minimum supported OS Version:** The minimum iOS version supported by the SALESmanago SDK is iOS 15.0 (available for iPhone versions 6s and above).
- **Permissions:** To be able to send notifications to your app users, you need two types of permissions, both of which must be requested by your app:
    - System permission
    - Marketing consent

  The status of these two permissions must be transferred to SALESmanago (for more details, see Section 3 below).

NOTE: *Currently, there is **no option to reassign** a device to a different Contact (different email address).*

EXAMPLE: *After using a development version of your mobile app, you want to test notifications for a different user. The execution of the* `updateUserProperties` *or* `updateUserContactData` *methods with a different email address will not assign the device to the new email address.*

2. SDK installation and initialization

You can install the SALESmanago SDK for the Mobile Push channel in two ways:

- **Automatically (using Swift Package Manager)** – Add the following link:

  https://bitbucket.org/benhauerdev/salesmanago-mobile-push-sdk-ios.git

  to your Xcode project settings, in Package Dependencies.

- **Manually:**

  1. Download SalesmanagoSDK.xcframework from the SALESmanago repository: **https://bitbucket.org/benhauerdev/salesmanago-mobile-push-sdk-ios/src/main**
  2. In your Xcode project's editor, select the app target and open the *General* tab.
  3. Expand the *Frameworks, Libraries, and Embedded Content* section.
  4. Click the add button (+), select *Add Other*, and then *Add Files* to locate your static framework. Click Open.
  5. Choose the *Embed & Sign* option from the *Embed value* list for the static framework.

To initialize the SALESmanago SDK, call the `initialize()` method, passing the SALESmanago API key. This method must be called at the app's startup, usually in the `configurationForConnecting` method of the `AppDelegate` class.

The required API key can be generated in SALESmanago (**Menu > Channels > Mobile Push > Settings > Integration settings > *API keys* tab**).

Swift

```
Unset
import UIKit
import SalesmanagoSDK

func application(
    _ application: UIApplication,
    configurationForConnecting connectingSceneSession: UISceneSession,
    options: UIScene.ConnectionOptions
) -> UISceneConfiguration
    Salesmanago.initialize(apiKey: <SALESmanago API key>)
}
```

Objective-C

```
Unset
#import <UIKit/UIKit.h>
#import <SalesmanagoSDK/SalesmanagoSDK-Swift.h>

- (UISceneConfiguration *)application:(UIApplication *)application
configurationForConnectingSceneSession:(UISceneSession
*)connectingSceneSession options:(UISceneConnectionOptions *)options
{
    [Salesmanago initializeWithApiKey:<SALESmanago API key>];
}
```

3. Mobile Push and In-App setup

Push notifications are sent using the Apple Push Notification service (APNs).

3.A. Managing consents

SALESmanago expects your application to transfer two distinct permissions for Push notifications: system permission and marketing consent. This configuration must be performed within the mobile app's code.

NOTE:

- *The order in which these permissions are requested is not important and totally up to you. However, users who have already granted marketing consent may be more likely to give system permission as well.*
- *The system permission status is transferred together with the marketing consent, using the* `Salesmanago.updateMobilePushOptIn` *method.*

System permission

Your application must request system permission to display notifications. The moment the permission request is shown to the user can be configured in the app's code.

If a user **dismisses** the permission request, no information is transferred to SALESmanago or iOS and the request **can** be shown to this user again.

If a user **denies** the permission request, it **cannot** be shown to this user again.

The status of the system permission is transferred to SALESmanago **together with the current status of the marketing consent**, using the `Salesmanago.updateMobilePushOptIn` method.

If a user denies the request, you can still try to obtain the permission by encouraging them to change the device's notification settings for your app. The mobile app can be configured to open the iOS notification settings. Once the user returns to the application, use the `updateMobilePushOptIn` or `sendDeviceTokenData` methods to update the permission status. Both methods are provided below.

Swift

```
Unset
import UserNotifications
import UIKit

...

let granted = try await UNUserNotificationCenter
    .current()
    .requestAuthorization(
    options: UNAuthorizationOptions(arrayLiteral: .alert, .badge, .sound)
    )

if granted {
    UIApplication.shared.registerForRemoteNotifications()
}
```

Objective-C

```
Unset
#import <UserNotifications/UserNotifications.h>
#import <UIKit/UIKit.h>

...

[[UNUserNotificationCenter currentNotificationCenter]
requestAuthorizationWithOptions:(UNAuthorizationOptionBadge |
UNAuthorizationOptionSound | UNAuthorizationOptionAlert)
completionHandler:^(BOOL granted, NSError * _Nullable error) {
    if (granted) {
        [[UIApplication sharedApplication] registerForRemoteNotifications];
    }
}];
```

In the `didRegisterForRemoteNotificationsWithDeviceToken` function of the `UIApplicationDelegate` protocol, call the `sendDeviceTokenData` method:

Swift

```
Unset
func application(
    _ application: UIApplication,
    didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data
) {
    Salesmanago.sendDeviceTokenData(deviceToken)
}
```

Objective-C

```
Unset
- (void)application:(UIApplication *)app
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [Salesmanago sendDeviceTokenData:deviceToken];
}
```

Marketing consent

In addition to system permission, SALESmanago requires obtaining marketing consent for displaying Mobile Push and In-App notifications. The way this consent is acquired (for example, its format, appearance, and display time) is fully configurable on your side.

The marketing consent request can be shown to the same user multiple times. Its status should be transferred to SALESmanago for both Contacts and anonymous app users, using the `updateMobilePushOptIn` method.

NOTE:

- *The system permission status is transferred along with the marketing consent status, using the same method –* `updateMobilePushOptIn`.
- *If you plan to display the marketing consent as a pop-up and want to prevent it from reappearing after consent has been given, ensure that the information about its acceptance (consent status:* `GRANTED`*) is stored by your mobile app. This must be configured on your side.*

Swift

```
Unset
Salesmanago.updateMobilePushOptIn(.granted)
```

Objective-C

```
Unset
[Salesmanago updateMobilePushOptIn:OptInOptionGranted];
```

3.B. Handling notifications

Implement the methods below to ensure that your notifications work as intended.

Push notifications

If your mobile app is currently not opened, clicking a link in a notification should launch the app. The methods provided below allow the transferring of data from that notification to SALESmanago.

If your application uses only the `UIApplicationDelegate` protocol and no `UIWindowSceneDelegate` protocol, implement the following method:

Swift

```
Unset
func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]? = nil
) -> Bool {
    Salesmanago.application(didFinishLaunchingWithOptions: launchOptions)
    return true
}
```

Objective-C

```
Unset
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary<UIApplicationLaunchOptionsKey,id
> *)launchOptions
{
    [Salesmanago
applicationWithDidFinishLaunchingWithOptions:launchOptions];
    return YES;
}
```

If your application uses the `UIWindowSceneDelegate` protocol, implement the following method:

Swift

```
Unset
func scene(
    _ scene: UIScene,
    willConnectTo session: UISceneSession,
    options connectionOptions: UIScene.ConnectionOptions
) {
    Salesmanago.scene(willConnectWithOptions: connectionOptions)
}
```

Objective-C

```
Unset
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession
*)session options:(UISceneConnectionOptions *)connectionOptions
{
    [Salesmanago sceneWithWillConnectWithOptions:connectionOptions];
}
```

In-App notifications

InApp notifications are sent with the `apns-push-type` header set to `background`. To enable the handling of In-App notifications by your mobile app, implement the `didReceiveRemoteNotification` method in the `UIApplicationDelegate` protocol:

Swift

```
Unset
func application(
    _ application: UIApplication,
    didReceiveRemoteNotification userInfo: [AnyHashable: Any],
    fetchCompletionHandler completionHandler: @escaping
(UIBackgroundFetchResult) -> Void
) {
    Salesmanago.didReceiveRemoteNotification(userInfo: userInfo)
    completionHandler(.noData)
}
```

Objective-C

```
Unset
- (void)application:(UIApplication *)application
didReceiveRemoteNotification:(NSDictionary *)userInfo
fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
    [Salesmanago didReceiveRemoteNotificationWithUserInfo:userInfo];
    completionHandler(UIBackgroundFetchResultNoData);
}
```

3.C. Implementing deep links

Unlike standard links that lead to webpages, **deep links** redirect users to specific locations within your application (for example, a product view). If you want to use deep links in your Mobile Push and/or In-App notifications, you need to register one or more URL schemes in Xcode.

Read more in Apple's documentation >>

URL scheme registration

1. Open the *Info* tab of your project settings.
2. In the *URL Types* section, declare all URL schemes supported by your app. Click the plus button and provide the required details.
    - The **identifier** is your mobile app's bundle ID (also used when integrating SALESmanago with iOS).
    - The **URL scheme** must be unique. To ensure that your URL scheme is unique, we recommend using your brand or application name in it, for example: *benhauer-salesmanago*.

After registering at least one URL scheme, implement the deep link handling method(s) provided below, depending on the protocol you use.

i. UIApplicationDelegate

If your application uses only the `UIApplicationDelegate` protocol and no `UIWindowSceneDelegate` protocol, you need to implement one `UIApplicationDelegate` method for handling deep links:

Swift

```
Unset
func application(
    _ app: UIApplication,
    open url: URL,
    options: [UIApplication.OpenURLOptionsKey: Any] = [:]
) -> Bool {
    // Handle `url`
    return true
}
```

Objective-C

```
Unset
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url
options:(NSDictionary<UIApplicationOpenURLOptionsKey, id> *)options
{
    if (url) {
    // Handle `url`
    }
}
```

ii. UIWindowSceneDelegate

If your application uses the `UIWindowSceneDelegate` protocol, you need to implement two `UIWindowSceneDelegate` methods for handling deep links.

Swift

```
Unset
func scene(
    _ scene: UIScene,
    willConnectTo session: UISceneSession,
    options connectionOptions: UIScene.ConnectionOptions
) {
    windowScene = scene as? UIWindowScene

    if let url: URL = connectionOptions.urlContexts.first?.url {
    // Handle `url`
    }
}

func scene(
    _ scene: UIScene,
    openURLContexts urlContexts: Set<UIOpenURLContext>
) {
    if let url: URL = urlContexts.first?.url {
    // Handle `url`
    }
}
```

Objective-C

```
Unset
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession
*)session options:(UISceneConnectionOptions *)connectionOptions
{
    NSURL *url = connectionOptions.URLContexts.allObjects.firstObject.URL;
    if (url) {
    // Handle `url`
    }
}

- (void)scene:(UIScene *)scene openURLContexts:(NSSet<UIOpenURLContext *>
*)URLContexts
{
    NSURL *url = URLContexts.allObjects.firstObject.URL;
    if (url) {
    // Handle `url`
    }
}
```

4. Contact data

The main entity in SALESmanago is a "**Contact**". A Contact represents a customer/user/website visitor who has provided their email address (the email address is required to create a "Contact Card").

Contacts can be described using a number of properties ("Contact data") useful for various marketing activities, including personalization, segmentation, and targeting. This section describes a number of methods that can be used to transfer Contact data from your mobile app to SALESmanago.

NOTE: *When transferring Contact data, the only mandatory field is the* ***email address.*** *If the email address is not transferred, SALESmanago can only store the system permission and the marketing consent described in Section 3.B. above, and the app user is considered anonymous.*

The following Contact properties can be transferred via the SALESmanago SDK:

- **Contact data:** name, email address, phone number, standard details (see Sections 4.A and 4.B below)
- **Marketing consents:**
    - Mobile Push and In-App consent (see Section 3 above)
    - Email Marketing consent (see Section 4.C below)
    - Mobile Marketing consent (see Section 4.C below)
- **Custom consents** (see Section 4.D below)
- **Contact tags** (see Section 4.E below)

The respective data fields available in the SALESmanago SDK are described in the table in Section 4.F.

The SDK enum class `OptInOption` represents the possible **statuses** for both marketing consents and custom consents:

- `GRANTED`—Contact has given the consent.
- `DENIED`—Contact has not given or has withdrawn the consent.
- `NO_ANSWER`—Contact has neither given nor rejected the consent. The current status will remain unchanged. If there is no status yet, the status will be set to `DENIED`.

4.A. Updating all Contact properties at once

Use the `Salesmanago.updateUserProperties` method to update multiple types of Contact properties at once.

All properties are optional. If you do not want to transfer any of them, use named parameters in Swift or set these parameters to nil in Objective-C. However, the first time you want to transfer Contact data, you need to include the **email address** (required to create a Contact Card in SALESmanago).

Swift

```
Unset
Salesmanago.updateUserProperties(
    contactData: UserContactData(
    name: "John Doe",
    email: "john.doe@email.com",
    phone: "+44123456789",
    standardDetails: [
        "ecoprogrammember": "yes",
        "size": "XL",
    ]
    ),
    marketingConsents: MarketingConsents(
    email: .granted,
    mobile: .noAnswer
    ),
    additionalConsents: [
    AdditionalConsent(
        name: "custom consent",
        status: .granted
    )
    ],
    tagsToAdd: ["tag_to_add"],
    tagsToRemove: ["tag_to_remove"]
)
```

Objective-C

```
Unset

[Salesmanago updateUserPropertiesWithContactData:[[UserContactData alloc]
        initWithName:@"John Doe"
        email:@"john.doe@email.com"
        phone:@"48123456789",
        standardDetails: @[
            @"ecoprogrammember": @"yes",
            @"size": @"XL"
        ]]
    marketingConsents:[[MarketingConsent alloc]
        initWithEmail: OptInOptionDenied
        mobile: OptInOptionGranted
        monitoring: OptInOptionNoAnswer //Website monitoring*
    ]
    additionalConsents:@[[[AdditionalConsent alloc]
        initWithName:@"custom consent"
        status:OptInOptionDenied]
    ]
    tagsToAdd:@[@"tagsToAdd"]
    tagsToRemove:@[@"tagsToRemove"]
];
```

*This field, corresponding to the user's consent to website monitoring (the storage of a SALESmanago cookie in their browser), may be removed from the SDK after the beta phase.

To ensure compatibility with the SDK for Android, SALESmanago provides you with additional methods for updating only specific types of Contact properties (Sections 4.B–4.E). Additionally, these methods allow you to avoid writing lengthy code blocks with nil parameters in Objective-C.

4.B. Updating basic Contact data

You can use a dedicated method to add or edit only basic Contact properties: name, address, phone number, and standard details.

Additionally, this method passes the email address, meaning it can be used to **create new Contacts**. However, the **email address cannot be edited** using this method. If you provide a different email address when updating Contact data, this field will be ignored.

To update a Contact's email address, use the **`contact/upsert` method >>**.

Swift

```
Unset
Salesmanago.updateUserContactData(
    name: "John Doe",
    email: "john.doe@email.com",
    phone: "+48123456789",
    standardDetails: [
    "ecoprogrammember": "yes",
    "size": "XL",
    ]
)
```

Objective-C

```
Unset
[Salesmanago updateUserContactDataWithName:@"John Doe
    email:@"john.doe@email.com"
    phone:@"+48123456789",
    standardDetails: @[
        @"ecoprogrammember": @"yes",
        @"size": @"XL"
    ]
];
```

4.C. Updating Email Marketing and Mobile Marketing consent status

The **Email Marketing consent** is required to send marketing emails (newsletter) to your Contacts.

The **Mobile Marketing consent** is required to send marketing text messages (SMS, WhatsApp, Viber) to your Contacts.

Both these consent types can be requested via your mobile app and transferred to SALESmanago.

NOTE: *This method can only be used for **Contacts**, i.e., after the user's email address has been transferred to SALESmanago using the* `Salesmanago.updateUserProperties` *or* `Salesmanago.updateUserContactData` *method.*

Swift

```
Unset
Salesmanago.updateUserMarketingConsents(
    email: .granted,
    mobile: .granted
)
```

Objective-C

```
Unset
[Salesmanago updateUserMarketingConsentsWithEmail:OptInOptionGranted
    mobile:OptInOptionGranted
    monitoring:OptInOptionGranted //Website monitoring*
];
```

*This field, corresponding to the user's consent to website monitoring (the storage of a SALESmanago cookie in their browser), may be removed from the SDK after the beta phase.

4.D. Updating custom consents

Custom consents are consents other than marketing consents (for example, consent to the processing for personal data for the purposes of an agreement). They are defined individually by each SALESmanago Client.

Custom consents are created and managed in **Menu → Audiences → Contacts → Custom consents.**

NOTE: *This method can only be used for* ***Contacts****, i.e., after the user's email address has been transferred to SALESmanago using the* `Salesmanago.updateUserProperties` *or* `Salesmanago.updateUserContactData` *method.*

Swift

```
Unset
Salesmanago.updateUserAdditionalConsents([
    AdditionalConsent(
    name: "Demo custom consent",
    status: .granted
    )
])
```

Objective-C

```
Unset
[Salesmanago updateUserAdditionalConsents:@[
    [[AdditionalConsent alloc] initWithName:@"Demo custom consent"
        status: OptInOptionGranted
    ]
]];
```

4.E. Adding and removing Contact tags

Tags are labels assigned to Contacts to enable their segmentation and precise targeting.

Tags can only consist of letters, digits, underscores (_), and dashes (-). Spaces will be converted to underscores. The minimum length is 3 characters, and the maximum length is 255 characters.

NOTE: *This method can only be used for* ***Contacts****, i.e., after the user's email address has been transferred to SALESmanago using the* `Salesmanago.updateUserProperties` *or* `Salesmanago.updateUserContactData` *method.*

Swift

```
Unset
Salesmanago.addTags(["only_tag_to_add"])
Salesmanago.removeTags(["only_tag_to_remove"])
```

Objective-C

```
Unset
[Salesmanago addTags: @[@"only_tag_to_add"]];
[Salesmanago addTags: @[@"only_tag_to_remove"]];
```

4.F. Data field specification

The data fields available in the SALESmanago SDK are described in the table below.

The same fields are used in all five methods for transferring Contact properties (Sections 4.A–4.E above).

Fields available in SALESmanago SDK

Object	Field	Limits	Description
`contactData`	`email`	RFC882	Required to create a Contact Card and store Contact data in SALESmanago. App users who have not provided an email address are referred to as “anonymous” and the only data that can be stored for them is system permission status and marketing consent status.
	`name`	255	Contact’s name, for example, first and last name.
	`phone`	255 Only digits and plus character	Contact’s phone number. It should start with + followed by the country code.
	`standardDetails` (object)	16x255	Object containing key-value pairs. Standard details can be used to store additional information about Contacts.
`marketingConsents`	`emailOptIn`	SDK enum class `OptInOption`	Email Marketing consent status: • `GRANTED`—Contact has given the consent. • `DENIED`—Contact has not given or has withdrawn the

			consent. • `NO_ANSWER`—Contact has neither given nor rejected the consent. The current status will remain unchanged. If there is no status yet, the status will be set to `DENIED`.
	`mobileOptIn`	SDK enum class `OptInOption`	Mobile Marketing consent status: • `GRANTED`—Contact has given the consent. • `DENIED`—Contact has not given or has withdrawn the consent. • `NO_ANSWER`—Contact has neither given nor rejected the consent. The current status will remain unchanged. If there is no status yet, the status will be set to `DENIED`.
`additionalConsents`	array of objects	255 SDK enum class `OptInOption`	Array of objects containing the name of a custom consent and its status: • `GRANTED`—Contact has given the consent. • `DENIED`—Contact has not given or has withdrawn the consent. • `NO_ANSWER`—Contact has neither given nor rejected the consent. The current status will remain unchanged. If there is no status yet, the status will be set to `DENIED`. Custom consents can be created in **Menu → Audiences → Contacts → Custom consents**.

`tagsToAdd`	array	3-255 per tag a-zA-Z0-9_ and - Max. 16 tags per request	Array of tags to be assigned to the Contact. Use ASCII characters only.
`tagsToRemove`	array	3-255 per tag a-zA-Z0-9_ and - Max. 16 tags per request	Array of tags to be removed from the Contact. Use ASCII characters only. **NOTE:** Tags present in both the `addTags` and `removeTags` arrays will be removed only. In that event, Automation Processes based on assigning a tag or increasing a tag scoring will not be triggered.

5. Adding events

The SALESmanago SDK enables tracking user activity by transferring predefined events to SALESmanago. The events are recorded for both Contacts and anonymous app users.

`EventType` is an SDK enum class representing the possible event types. At present, the following event types are available:

- `LOGIN`—Occurs when the user logs in to your mobile app.

Swift

```
Unset
Salesmanago.addEvent(EventType.LOGIN)
```

Objective-C

```
Unset
[Salesmanago addEventWithEventType: EventTypeLogin];
```

If you have any questions or doubts concerning the configuration of the Mobile Push channel, or if you would like to have your setup verified by our Support specialist, please contact us at:

support@salesmanago.com
