

Integrate **Manago AI** with your **Mobile App**

iOS
(Native)

Contents

1. Prerequisites	2
2. SDK installation and initialization	3
2.A. Installation	3
2.B. Initialization	4
2.C. Initialization completion callback	5
2.D. Reinitialization during runtime	6
3. Mobile Push and In-App setup	8
3.A. Enabling Xcode extension	8
3.B. In-App Queue Interval	10
3.C. Managing consents	11
3.D. Handling notifications	14
3.E. Implementing deep links	16
4. Contact data	20
4.A. Updating all Contact properties at once	20
4.B. Updating basic Contact data	23
4.C. Updating Email Marketing and Mobile Marketing consent status	24
4.D. Updating custom consents	25
4.E. Adding and removing Contact tags	26
4.F. Data field specification	27
5. Adding events	30
5.A. App-level events	30
5.B. External Events (transactions)	30

1. Prerequisites

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

- **Minimum supported OS Version:** The minimum iOS version supported by the Manago AI 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 Manago AI (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 `updateContactProperties` or `updateContactData` methods with a different email address will not assign the device to the new email address.

2. SDK installation and initialization

2.A. Installation

You can install the Manago AI 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 Manago AI 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.

2.B. Initialization

To initialize the Manago AI SDK, call the `initialize()` method, passing the Manago AI 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 Manago AI (**Menu > Channels > Mobile Push > Settings > Integration settings > API keys tab**).

Swift

```
import UIKit
import SalesmanagoSDK

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

Objective-C

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

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

2.C. Initialization completion callback

If you need to call another SDK method right after initialization, use the completion callback and continue only after success. Initialization completion is delivered on the MainActor / main thread, so this callback is the recommended place to chain dependent SDK calls.

The completion can return failure if initialization is called in an invalid state (for example, when the SDK is already initialized) or when a runtime initialization error occurs (for example, invalid API key).

Swift

```
Salesmanago.initialize(apiKey: <API key>) { isSuccess, errorMessage in
    guard isSuccess else {
        print(errorMessage ?? "Initialization failed")
        return
    }
    Salesmanago.updateContactData(email: "john.doe@email.com")
}
```

Objective-C

```
[Salesmanago initializeWithApiKey:<API key>
                    completion:^(BOOL isSuccess, NSString * _Nullable
errorMessage) {
    if (!isSuccess) {
        NSLog(@"%@", errorMessage ?: @"Initialization failed");
        return;
    }
    [Salesmanago updateContactDataWithName:nil
                                email:@"john.doe@email.com"
                                phone:nil
                                userId:nil
                                company:nil
                                state:ContactStateUnknown
                                birthday:nil
                                streetAddress:nil
                                city:nil
                                zipCode:nil
                                province:nil
                                country:nil
                                doubleOptInEmailId:nil
                                doubleOptInLanguage:nil
                                standardDetails:nil
                                numberDetails:nil
                                dateDetails:nil];
}];
```

2.D. Reinitialization during runtime

The SDK supports runtime reinitialization with a new API key. This allows switching between different Manago AI API keys during the application's lifecycle without restarting the app. This lets you change the Manago AI account to which the data is sent, which is especially useful for apps supporting multiple regions.

The SDK switches to the new API key, reinitializes all necessary components, and performs an initial synchronization. The SDK keeps track of existing Mobile App User IDs and tries to re-use the same IDs when switching back to a previously used API key.

To reinitialize the SDK with a new API key, call the `reInit(apiKey:)` method:

Swift

```
Salesmanago.reInit(apiKey: <new API key>)
```

Objective-C

```
[Salesmanago reInitWithApiKey:<new API key>];
```

Important: Switching the API key does not automatically opt out Contacts from Mobile Push notifications. If you want to send Mobile Push notifications only from the Manago AI account associated with the new API key, consider revoking the consent for marketing notifications first, before executing the API key swap. This way, after changing the API key, Contacts will no longer receive marketing Mobile Push notifications; however, the transactional ones, such as order confirmations, will still be delivered.

Reinitialization with a callback

Reinitialization completion callback is delivered on the MainActor / main thread and returns success/failure with an optional error message:

Swift

```
Salesmanago.reInit(apiKey: <new API key>) { isSuccess, errorMessage in
    guard isSuccess else {
        print(errorMessage ?? "Reinitialization failed")
        return
    }

    Salesmanago.updateContactData(email: "john.doe@email.com")
}
```

Objective-C

```
[Salesmanago reInitWithApiKey:<new API key>
    completion:^(BOOL isSuccess, NSString * _Nullable errorMessage) {
    if (!isSuccess) {
        NSLog(@"%@", errorMessage ?: @"Reinitialization failed");
        return;
    }

    [Salesmanago updateContactDataWithName:nil
        email:@"john.doe@email.com"
        phone:nil
        userId:nil
        company:nil
        state:ContactStateUnknown
        birthday:nil
        streetAddress:nil
        city:nil
        zipCode:nil
        province:nil
        country:nil
        doubleOptInEmailId:nil
        doubleOptInLanguage:nil
        standardDetails:nil
        numberDetails:nil
        dateDetails:nil];
}];
```

3. Mobile Push and In-App setup

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

3.A. Enabling Xcode extension

To use Rich Mobile Push notifications (notifications including images), you must enable Notification Service Extension in Xcode settings:

1. In Xcode, go to File > New > Target...
2. Select Notification Service Extension and click Next
3. Enter a product name (e.g. *RichPushNotificationsExtension*) and click Finish
4. When prompted to activate the scheme, click Cancel (you don't need to activate it)

Important: Make sure the extension's Bundle Identifier is a child of your main app's bundle identifier (e.g. if your app is `com.example.myapp`, the extension should be `com.example.myapp.RichPushNotificationsExtension`).

Next, you must add SalesmanagoSDK framework to the extension target

1. Select your project in the Project Navigator
2. Select the Notification Service Extension target
3. Go to the General tab
4. Under Frameworks and Libraries, click +
5. Add SalesmanagoSDK.xcframework
6. Make sure the framework is set to Embed & Sign

Finally, replace the entire contents of the generated `NotificationService.swift` file in your Notification Service Extension target with the implementation below.

Note: Swift implementation is recommended, even for apps written in Objective-C. The extension can be written in Swift regardless of the main app language.

Swift

```
//
// NotificationService.swift
// RichPushNotificationsTestExtension
//

import SalesmanagoSDK
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(
        _ request: UNNotificationRequest,
        withContentHandler contentHandler: @escaping (UNNotificationContent) -> Void
    ) {
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as?
UNMutableNotificationContent)
        guard let content = bestAttemptContent else {
            contentHandler(request.content)
            return
        }

        Salesmanago.didReceive(request)

        guard let imageURL = URL(string: content.launchImageName) else {
            contentHandler(content)
            return
        }

        downloadImage(from: imageURL) { attachment in
            if let attachment {
                content.attachments = [attachment]
            }
            contentHandler(content)
        }
    }

    override func serviceExtensionTimeWillExpire() {
        // Called just before the extension will be terminated by the system.
        // Use this as an opportunity to deliver your "best attempt" at modified content,
otherwise the original push
        // payload will be used.
        if let contentHandler, let bestAttemptContent {
            contentHandler(bestAttemptContent)
        }
    }
}
```

```
private extension NotificationService {
    func downloadImage(from url: URL, completion: @escaping (UNNotificationAttachment?) -> Void) {
        let task = URLSession.shared.dataTask(with: url) { data, _, error in
            guard let data, error == nil else {
                completion(nil)
                return
            }

            do {
                let tempDirectory = FileManager.default.temporaryDirectory
                let fileURL = tempDirectory.appendingPathComponent(url.lastPathComponent)
                try data.write(to: fileURL)

                let attachment = try UNNotificationAttachment(identifier: "imageAttachment",
url: fileURL, options: nil)
                completion(attachment)
            } catch {
                completion(nil)
            }
        }
        task.resume()
    }
}
```

3.B. In-App Queue Interval

The SDK enforces a minimum delay between consecutive in-app message displays (default: 60 seconds). You can configure this interval at runtime using the `configure()` method. The value is specified in seconds.

Setting the interval to 0 disables the delay, allowing in-app messages to display back-to-back. If a shorter interval is set while a display is already scheduled, and the new interval has already elapsed, the next in-app will be shown immediately.

Swift

```
Salesmanago.configure(inAppQueueInterval: 60)
```

Objective-C

```
[Salesmanago configureWithInAppQueueInterval:60];
```

3.C. Managing consents

Manago AI 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 Manago AI 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 Manago AI **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
import UserNotifications
import UIKit
...

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

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

Objective-C

```
#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

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

Objective-C

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

Marketing consent

In addition to system permission, Manago AI 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 Manago AI 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

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

Objective-C

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

3.D. 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 Manago AI.

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

Swift

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

Objective-C

```
- (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

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

Objective-C

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

Web links

By default, HTTP(S) URLs contained in a push notification are opened in the device's external browser (Safari). Applications can instead configure the SDK to open these links in an in-app browser based on SFSafariViewController, giving a more integrated experience without leaving the app.

Swift

```
Salesmanago.configure(urlPresentation: .inAppBrowser)
Salesmanago.configure(urlPresentation: .externalBrowser)
```

Objective-C

```
[Salesmanago configureWithUrlPresentation:URLPresentationInAppBrowser];
[Salesmanago configureWithUrlPresentation:URLPresentationExternalBrowser];
```

In-App notifications

In-App 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

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

Objective-C

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
```

```
{  
    [Salesmanago didReceiveRemoteNotificationWithUserInfo:userInfo];  
    completionHandler(UIBackgroundFetchResultNoData);  
}
```

3.E. 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 Manago AI 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.

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

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

Objective-C

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

UIWindowSceneDelegate

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

Swift

```
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

```
- (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 Manago AI 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 Manago AI.

NOTE: When transferring Contact data, the only mandatory field is the **email address**. If the email address is not transferred, Manago AI 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 Manago AI 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 Manago AI 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.updateContactProperties` 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 Manago AI).

Swift

```
Salesmanago.updateContactProperties(  
    contactData: ContactData(  
        name: "John Doe",  
        email: "john.doe@example.com",  
        newEmail: "john.doe.new@example.com",  
        phone: "+15554443322",  
        userId: "U123456",  
        company: "Acme Inc.",  
        state: .customer,  
        birthday: "19870605",  
        streetAddress: "Main Street 1",  
        city: "San José",  
        zipCode: "90210",  
        province: "California",  
        country: "US",  
        standardDetails: [  
            "eco_program_member": "yes",  
            "t_shirt_size": "XL",  
        ],  
        numberDetails: [  
            "shoe_size": 10  
        ],  
        dateDetails: [  
            "member_since": 1459938888  
        ],  
        doubleOptIn: ContactData.DoubleOptIn(  
            emailId: "a2b2-...-c3d4",  
            language: "es" //Include either the language or emailId  
        )  
    ),  
    marketingConsents: MarketingConsents(  
        email: .granted,  
        mobile: .noAnswer  
        monitoring: .granted  
    ),  
    additionalConsents: [  
        AdditionalConsent(  
            name: "TERMS_OF_SERVICE",  
            status: .granted  
        )  
    ],  
    tagsToAdd: ["T_SHIRTS", "SWEATERS", "UPSELL"],  
    tagsToRemove: ["FIRST_PURCHASE"]  
)
```

Objective-C

```
[Salesmanago updateContactPropertiesWithContactData:[(ContactData alloc)
    initWithName:@"John Doe"
    email:@"john.doe@example.com"
    newEmail:@"john.doe.new@example.com"
    phone:@"+15554443322"
    userId:@"U123456"
    company:@"Acme Inc."
    state:ContactStateCustomer
    birthday:@"19870605"
    streetAddress:@"Main Street 1"
    city:@"San José"
    zipCode:@"90210"
    province:@"California"
    country:@"US"
    doubleOptIn:[(DoubleOptIn alloc) initWithEmailId:@"a2b2-...-c3d4"
        language:@"es"] //Include either the language or emailId
    standardDetails:@{
        @"eco_program_member": @"yes",
        @"t_shirt_size": @"XL"
    }
    numberDetails:@{ @"shoe_size": @10 }
    dateDetails:@{ @"member_since": @1459938888 }
    marketingConsents:[(MarketingConsents alloc)
        initWithEmail: OptInOptionDenied
        mobile: OptInOptionGranted
        monitoring: OptInOptionNoAnswer //Website monitoring. This field may be removed in
the future
    ]
    additionalConsents:@[[AdditionalConsent alloc]
        initWithName:@"TERMS_OF_SERVICE"
        status:OptInOptionDenied]
    ]
    tagsToAdd:@[@"T_SHIRTS", @"SWEATERS", @"UPSELL"]
    tagsToRemove:@[@"FIRST_PURCHASE"]
];
```

Methods for transferring partial data

To ensure compatibility with the SDK for Android, Manago AI 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, e.g. name, address, phone number, and standard details.

Additionally, this method passes the email address, meaning it can be used to **create new Contacts**.

Swift

```
Salesmanago.updateContactData(  
    name: "John Doe",  
    email: "john.doe@example.com",  
    newEmail: "john.doe.new@example.com",  
    phone: "+15554443322",  
    userId: "U123456",  
    company: "Acme Inc.",  
    state: .customer,  
    birthday: "19870605",  
    streetAddress: "Main Street 1",  
    city: "San José",  
    zipCode: "90210",  
    province: "California",  
    country: "US",  
    standardDetails: [  
        "eco_program_member": "yes",  
        "t_shirt_size": "XL",  
    ],  
    numberDetails: [  
        "shoe_size": 10  
    ],  
    dateDetails: [  
        "member_since": 1459938888  
    ],  
    doubleOptIn: ContactData.DoubleOptIn(  
        emailId: "a2b2-...-c3d4",  
        language: "es" //Include either the language or emailId  
    )  
)
```

Objective-C

```
[Salesmanago updateContactDataWithName:@"John Doe"
    email:@"john.doe@example.com"
    newEmail:@"john.doe.new@example.com"
    phone:@"+15554443322"
    userId:@"U123456"
    company:@"Acme Inc."
    state:ContactStateCustomer
    birthday:@"19870605"
    streetAddress:@"Main Street 1"
    city:@"San José"
    zipCode:@"90210"
    province:@"California"
    country:@"US"
    doubleOptIn:[DoubleOptIn alloc] initWithEmailId:@"a2b2-...-c3d4"
    language:@"es"] //Include either the language or emailId
standardDetails: @{
    @"eco_program_member": @"yes",
    @"t_shirt_size": @"XL"
}
numberDetails:@{ @"shoe_size": @10 }
dateDetails:@{ @"member_since": @1459938888 }
];
```

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 Manago AI.

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

Swift

```
Salesmanago.updateContactMarketingConsents(
    email: .granted,
    mobile: .granted,
    monitoring: .granted
)
```

Objective-C

```
[Salesmanago updateContactMarketingConsentsWithEmail:OptInOptionGranted
    mobile:OptInOptionGranted
```

```
monitoring:OptInOptionGranted //Website monitoring*  
];
```

*This field, corresponding to the user's consent to website monitoring (the storage of a Manago AI 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 Manago AI 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 Manago AI using the `Salesmanago.updateContactProperties` or `Salesmanago.updateContactData` method.

Swift

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

Objective-C

```
[Salesmanago updateContactAdditionalConsents:@[  
    [[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 Manago AI using the `Salesmanago.updateContactProperties` or `Salesmanago.updateContactData` method.

Swift

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

Objective-C

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

4.F. Data field specification

The data fields available in the Manago AI 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 Manago AI SDK

Object	Field	Limits	Description
contactData / First parameter	name	255	Contact's name, for example, first and last name.
	email	RFC822	Required to create a Contact Card and store Contact data in Manago AI. 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.
	newEmail	RFC882	New email address of the Contact. The specified email address cannot be already assigned to another Contact. In Java implementation, leave this field as null to keep the existing email address of the Contact.
	phone	Plus sign followed by 11 digits	Contact's phone number. It should start with + followed by the country code.
	userId	255	Optional field for user identification on supported Manago AI accounts
	company	255	Company the Contact is associated with. In Manago AI, you can list Contacts associated with a given company.
	state	enum	Contact's state (CUSTOMER, PROSPECT, PARTNER, OTHER, UNKNOWN)
	birthday	YYYYMMDD or MMDD	Contact's birthday (used for birthday emails).
	streetAddress	255	Street and apartment number. Additional data is allowed.
	city	255	City. Special characters are allowed.
	zipCode	255	Zip code (postcode). Special characters are allowed.
	province	255	Administrative division within a country. Special characters are allowed.

	country	255	Country. Special characters are allowed.
	standardDetails (object)	16×255	Object containing key-value pairs. Standard details can be used to store additional information about Contacts.
	numberDetails (object)	16×255	Object containing key-value pairs. Number details can be used to store additional information about Contacts.
	dateDetails (object)	16×255	Object containing key-value pairs. Date details can be used to store additional information about Contacts or trigger automations at a specific time.
	doubleOptIn (object)	emailId: UUID language: ISO 639-1	Either the ID of the email to be sent or the Contact's language to be used in the double opt-in email
marketingConsents / Second parameter	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 / Third parameter	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 / Fourth parameter	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 / Fifth parameter	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.

[illegible]

```
@ "P234567" ]

Card"

products:@["@P012345", @ "P123456",

value:@99.99
location:@ "examplecom"
externalId:@ "ID-123456"
detail1:@ "Payment method: Credit

detail2:@ "Shipping: Express"
detail3:@ "Discount: 10%"
detail4:@ "Customer segment: Premium"
detail5:@ "Campaign: Summer Sale"
detail6:nil
detail7:nil
detail8:nil
detail9:nil
detail10:nil
detail11:nil
detail12:nil
detail13:nil
detail14:nil
detail15:nil
detail16:nil
detail17:nil
detail18:nil
detail19:nil
detail20:nil
description:@ "Example purchase

transaction"];
```

Updating and deleting External Events

While adding a new external event, the SDK generates an eventId (UUID) and returns it for further managing the event via other methods. This can be used to update or delete the event via `updateExternalEvent` and `deleteExternalEvent` methods:

```
Swift

Salesmanago.updateExternalEvent(
    eventId: eventId,
    eventTime: "1772700397",
    eventType: .cancellation,
    products: ["P012345", "P123456", "P234567"],
    value: 1,
    detail1: "Updated detail",
    description: "Updated description"
)
```

Objective-C

```
[Salesmanago updateExternalEventWithEventId:eventId
                      eventTime:@"1772700397"
                      eventType:ExternalEventTypeCancellation
                      products:@[@"P012345", @"P123456", @"P234567"]
                      value:@1
                      location:nil
                      externalId:nil
                      detail1:@"Updated detail"
                      detail2:nil
                      detail3:nil
                      detail4:nil
                      detail5:nil
                      detail6:nil
                      detail7:nil
                      detail8:nil
                      detail9:nil
                      detail10:nil
                      detail11:nil
                      detail12:nil
                      detail13:nil
                      detail14:nil
                      detail15:nil
                      detail16:nil
                      detail17:nil
                      detail18:nil
                      detail19:nil
                      detail20:nil
                      description:@"Updated description"];
```

Method `deleteExternalEvent` uses `eventId` or `externalId` to identify the event. At least one of the params has to be specified. If you specify both, the `eventId` will take priority.

Swift

```
Salesmanago.deleteExternalEvent(
    eventId: eventId,
    externalId: "ID-123456"
)
```

Objective-C

```
[Salesmanago deleteExternalEventWithEventId:eventId externalId:@"ID-123456"];
```

We are here to support you

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@manago.ai