SALESmanago

Setting up the Mobile Push channel

Integrating SALESmanago with Your Mobile App:

REACT NATIVE

Contents

1. Prerequisites

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

- **Minimum supported versions:**
    - Android: min. 7.0 Nougat (API level 24)
    - iOS: min. 15.0 (iPhone versions 6s and above)
    - React Native: min. 0.68.0
- **Supported React Native architectures:**
    - Old architecture: YES
    - New architecture (Fabric): YES
- **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 (all iOS versions, for Android – versions 13 [API level 33] and above)
    - Marketing consent

  The status of these two permissions must be transferred to SALESmanago (for more details, see Section 4 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* `Salesmanago.updateContactProperties` *or* `Salesmanago.updateContactData` *methods with a different email address will not assign the device to the new email address.*

2. SDK installation and initialization

To install the `@salesmanago/mobile-push-sdk-react-native` library, execute the following methods:

```
Unset
# using npm
npm install @salesmanago/mobile-push-sdk-react-native

# OR using Yarn
yarn add @salesmanago/mobile-push-sdk-react-native
```

To initialize the SALESmanago SDK, call the `init()` method, which passes the `applicationContext` object and SALESmanago API key. This method must be called at the app's startup.

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

```
Unset
import { Salesmanago } from '@salesmanago/mobile-push-sdk-react-native';

Salesmanago.init(<API key>)
```

3. Mobile Push and In-App setup

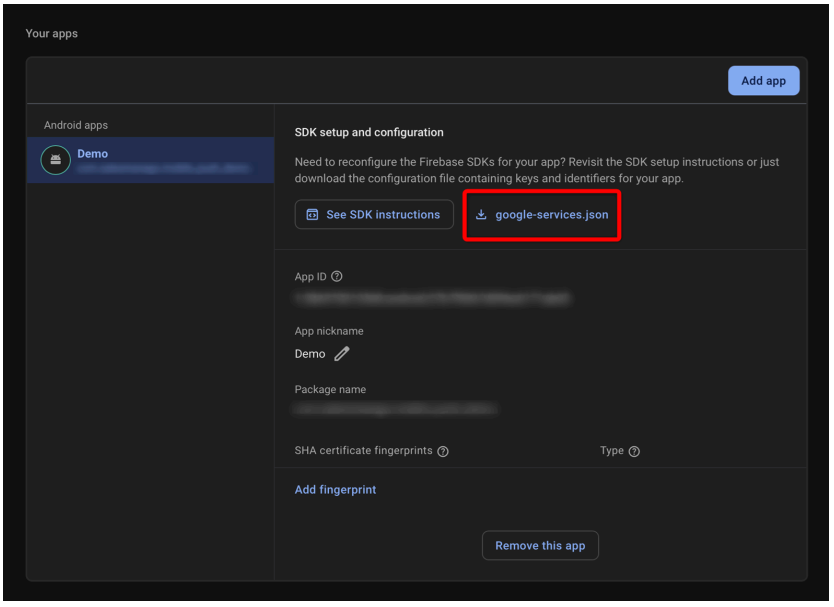
Push notifications are sent via Firebase Cloud Messaging on Android and the Apple Push Notification service (APNs) on iOS. To ensure that your notifications function as intended, perform the configuration steps described below.

3.A. Android: Connecting SALESmanago to Firebase

To integrate SALESmanago with your mobile app, include your google-services.json file (required to initialize the SALESmanago SDK) in the app-level root directory.

To obtain the google-services.json file:

1. Open the Firebase Console and select your project.
2. Click the gear icon next to *Project Overview* and select *Project Settings*.
3. Scroll down to the *Your apps* section, select your Android app, and click the download button for the google-services.json file.



Note that you can also obtain your Android app's config file via REST API (**projects.androidApps.getConfig >>**).

Place the downloaded JSON file in the app-level directory of your app. Make sure that you only have the most recently downloaded config file in your app.

At this stage, add the Gradle plugin named “Google Services” (for example, from the **MVN Repository >>**) to your **project-level** `build.gradle` file:

Groovy

```
Unset
plugins {
    // ...

    // dependency for the Google services Gradle plugin
    id("com.google.gms.google-services") version "<version>" apply false
}
```

and to the **app-level** `build.gradle` file:

Groovy

```
Unset
plugins {
    id("com.android.application")

    // Google services Gradle plugin
    id("com.google.gms.google-services")

    ...
}
```

3.B. Android: Handling In-App notifications

To make sure In-App notifications are displayed correctly in your React Native app on Android, you need to properly configure your main activity.

Check your `AndroidManifest.xml` file and look at the `<activity>` tag for your main activity. Make sure the `android:taskAffinity` attribute is not set to an empty string. If `taskAffinity` is empty, Android may block notifications from appearing properly.

Correct:

```
Unset
<activity
    android:name=".MainActivity"
    android:taskAffinity="com.example.myapp" />
```

Incorrect:

```
Unset
<activity
    android:name=".MainActivity"
    android:taskAffinity="" />
```

If you do not set `android:taskAffinity` at all, Android will automatically assign a default value based on your app's package name, which works correctly for displaying In-App notifications.

3.C. iOS: Handling 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,
    didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data
) {
    Salesmanago.sendDeviceTokenData(deviceToken)
}
```

Objective-C

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

3.D. Deep links

Just like standard links 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 Mobile Push and/or In-App notifications, you need to perform additional configuration for both Android and iOS.

Android

For Android, each deep link needs to be declared in your app.

The `AndroidManifest.xml` file declares your main activity (usually `MainActivity`) within the `<activity>` tag. To enable this activity to handle deep links, declare your `scheme` in a separate `<intent-filter>`. For instance, if you create a notification in SALESmanago with the following deep link: `salesmanago://main`, the `<activity>` tag should contain the following declarations:

```
Unset
<activity
    ...
    ...

    <intent-filter>
        ...
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="salesmanago" />
    </intent-filter>
</activity>
```

iOS

For iOS, the deep link scheme has to be declared in your `Info.plist` file. For instance, if you create a notification in SALESmanago with the following deep link: `salesmanago://main`, your `Info.plist` file should contain the following declarations:

```
Unset
<key>CFBundleURLTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>Editor</string>
        <key>CFBundleURLName</key>
        <string>YOUR APP BUNDLE IDENTIFIER</string>
        <key>CFBundleURLSchemes</key>
        <array>
            <string>salesmanago</string>
        </array>
    </dict>
</array>
```

You can also 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*.

4. 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. However, users who have already granted marketing consent may be more likely to give system permission as well.*

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.

When the system permission is granted or denied, this status must be explicitly transferred to SALESmanago using the `Salesmanago.updatePushNotificationSystemPermissions();` method.

If a user **dismisses** the permission request, no information is transferred to SALESmanago or Android 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.

If you attempt to obtain the permission by opening the device's notification settings for your app, once the user returns to the application, use the `Salesmanago.updatePushNotificationSystemPermissions()` method to update the permission status.

Additionally, every time you call the `init` method in Android, the permission status is automatically updated in SALESmanago (for example, when a user blocks notifications on the list of notifications or changes the permission status in Android's notification settings).

In iOS, the `init` method does not update the system permission status.

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 `Salesmanago.updateMobilePushOptIn(OptInOption)` method. An example is provided below, and the **possible statuses are described in Section 5**.

NOTE: *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.*

```
Unset
Salesmanago.updateMobilePushOptIn(OptInOption.GRANTED);
```

5. 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 4. 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 5.A and 5.B below)
- **Marketing consents:**
  - Mobile Push and In-App consent (see Section 4 above)
  - Email Marketing consent (see Sections 5.A and 5.C below)
  - Mobile Marketing consent (see Sections 5.A and 5.C below)
- **Custom consents** (see Sections 5.A and 5.D below)
- **Contact tags** (see Sections 5.A and 5.E below)

The respective data fields available in the SALESmanago SDK are described in the table in Section 5.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`.

5.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, specify only the desired ones using named parameters. 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).

```
Unset
Salesmanago.updateContactProperties({
  contactData: {
    name: "John Doe",
    email: "john.doe@example.com",
    phone: "+48123456789",
    standardDetails: { "custom_field": "custom_value" }
  },
  marketingConsents: {
    email: OptInOption.GRANTED,
    mobile: OptInOption.DENIED,
    monitoring: OptInOption.NO_ANSWER
  },
  additionalConsents: [
    { name: "custom consent", status: OptInOption.DENIED }
  ],
    tagsToAdd: ["tag_to_add"],
    tagsToRemove: ["tag_to_remove"]
});
```

To facilitate the development of your mobile app, SALESmanago provides you with additional methods for updating only specific types of Contact properties (Sections 5.B–5.E).

These methods are particularly useful when migrating your native Android and/or iOS application(s) to React Native.

5.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 >>**.

Unset

```
 Salesmanago.updateContactData({
  name: "John Doe",
  email: "john.doe@example.com",
  phone: "+48123456789",
  standardDetails: { "custom_field": "custom_value" }
});
```

5.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 after the user's email address has been transferred to SALESmanago using the* `Salesmanago.updateContactProperties` *or* `Salesmanago.updateContactData` *method.*

```
Unset
Salesmanago.updateContactMarketingConsents({
  email: OptInOption.DENIED,
  mobile: OptInOption.NO_ANSWER,
  monitoring: OptInOption.GRANTED
});
```

5.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 after the user's email address has been transferred to SALESmanago using the* `Salesmanago.updateContactProperties` *or* `Salesmanago.updateContactData` *method.*

Unset

```
Salesmanago.updateContactAdditionalConsents([
  { name: "some custom consent", status: OptInOption.GRANTED }
]);
```

5.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 after the user's email address has been transferred to SALESmanago using the* `Salesmanago.updateContactProperties` *or* `Salesmanago.updateContactData` *method.*

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

5.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 5.A–5.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.

6. 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.

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

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