Setting up the Mobile Push channel

Integrating SALESmanago with Your Mobile App: ANDROID

Contents

1. Prerequisites **2**
2. SDK installation and initialization **3**
3. Mobile Push and In-App setup **5**
3.A. Connecting SALESmanago to Firebase 5
3.B. Managing consents 7
3.C. Deep links 10
4. Contact data **11**
4.A. Updating all Contact properties at once 12
4.B. Updating basic Contact data 15
4.C. Updating Email Marketing and Mobile Marketing consent status 16
4.D. Updating custom consents 17
4.E. Adding and removing Contact tags 18
4.F. Data field specification 19
5. Adding events **22**

1. Prerequisites

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

- **Minimum supported OS Version:** The minimum Android version supported by the SALESmanago SDK is 5.0 Lollipop (API level 21).
- **google-services.json:** The google-services.json file (downloaded from your Firebase account) must be placed in the app-level root directory (see Section 3.A below).
- **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—for Android 13 (API level 33) and above;
    - 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* `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 SDK in your mobile app, add the dedicated dependency to the `build.gradle` file in your project, as shown below:

```
Unset
implementation("com.salesmanago:mobilepush:1.0.0")
```

Next, add a custom Maven location in the `repositories {}` section:

```
Unset
repositories {

    // ...


    maven {

        url =
uri("https://europe-central2-maven.pkg.dev/salesmanago-mvn/mobile")

    }

}
```

To initialize the SALESmanago SDK, call the `init()` method, passing the `applicationContext` object and the SALESmanago API key. This method must be called at the app's startup, usually in the `onCreate` method of the `Application` class.

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

Kotlin

```
Unset
Salesmanago.init(applicationContext, <SALESmanago API key>)
```

Java

```
Java
Salesmanago.INSTANCE.init(applicationContext, <SALESmanago API key>);
```

3. Mobile Push and In-App setup

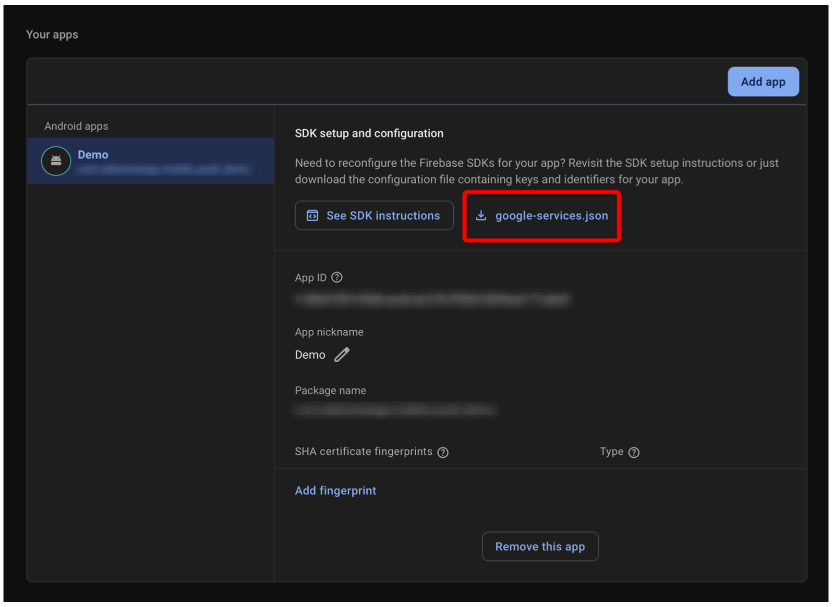
Push notifications are sent using Firebase Cloud Messaging. The SALESmanago SDK handles all Firebase management tasks for you.

3.A. 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.



Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.

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 recent downloaded config file in your app.

See the instructions in Firebase's help portal >>

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. 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

Starting from Android version 13, 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, for example, using the `ActivityCompat.requestPermissions` method.

When the system permission is granted or denied, this status must be explicitly transferred to SALESmanago using the `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 Android's notification settings for your app, once the user returns to the application, use the `updatePushNotificationSystemPermissions()` method to update the permission status.

Additionally, every time you call the `init` method, 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).

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.

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.*

Kotlin

Unset
```
object PushConsentManager {

    fun askForMarketingConsent(context: Context) {
        AlertDialog.Builder(context)
            .setTitle("Marketing Consent")
            .setMessage("Would you like to receive marketing
notifications?")
            .setPositiveButton("Yes") { _, _ ->
                Salesmanago.updateMobilePushOptIn(OptInOption.GRANTED)
            }
            .setNegativeButton("No") { _, _ ->
                Salesmanago.updateMobilePushOptIn(OptInOption.DENIED)
            }
            .setNeutralButton("Not now") { _, _ ->
                Salesmanago.updateMobilePushOptIn(OptInOption.NO_ANSWER)
            }
            .show()
    }
}
```

Java

```
Java
public class PushConsentManager {

    public static void askForMarketingConsent(Context context) {
        new AlertDialog.Builder(context)
            .setTitle("Marketing Consent")
            .setMessage("Would you like to receive marketing
notifications?")
            .setPositiveButton("Yes", (dialog, which) -> {
                Salesmanago.updateMobilePushOptIn(OptInOption.GRANTED);
            })
            .setNegativeButton("No", (dialog, which) -> {
                Salesmanago.updateMobilePushOptIn(OptInOption.DENIED);
            })
            .setNeutralButton("Not now", (dialog, which) -> {
                Salesmanago.updateMobilePushOptIn(OptInOption.NO_ANSWER);
            })
            .show();
    }
}
```

SALES manago

3.C. 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 declare them in your app.

For instance, if you create a notification with the following deep link: `salesmanago://main`, the destination screen must declare `host` and `scheme` in the `AndroidManifest.xml` file, as shown below:

XML

Unset
```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />

    <data
        android:host="main"
        android:scheme="salesmanago" />
</intent-filter>
```

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.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 Kotlin or set these parameters to null in Java. 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).

Kotlin

Unset
```
Salesmanago.updateContactProperties(
    contactData = ContactData(
        name = "John Doe",
        email = "john.doe@email.com",
        phone = "+44123456789",
        standardDetails = mapOf(
            "ecoprogrammember" to "yes",
            "size" to "XL"
        )
    ),
    marketingConsents = MarketingConsents(
        email = OptInOption.GRANTED,
        mobile = OptInOption.DENIED
    ),
    additionalConsents = listOf(
        AdditionalConsent(
            "demo custom consent",
            OptInOption.DENIED
        )
    ),
    tagsToAdd = listOf("tag_to_add"),
    tagsToRemove = listOf("tag_to_remove")
)
```

Java

```java
Java
Salesmanago.INSTANCE.updateContactProperties(
    new ContactData(
        "John Doe",
        "john.doe@email.com",
        "+44123456789",
        new HashMap<String, String>() {
            put("ecoprogrammember", "yes");
            put("size", "XL");
        }
    ),
    new MarketingConsents(
        OptInOption.DENIED, //Email Marketing
        OptInOption.NO_ANSWER, //Mobile Marketing
        OptInOption.GRANTED //Website monitoring*
    ),
    Arrays.asList(
        new AdditionalConsent(
            "demo custom consent",
            OptInOption.DENIED
        )
    ),
    Arrays.asList("tag_to_add"),
    Arrays.asList("tag_to_remove")
);
```

*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 facilitate the development of your mobile app, SALESmanago provides you with additional methods for updating only specific types of Contact properties (Sections 4.B–4.E). This way, you can avoid writing lengthy code blocks with null parameters in Java.

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 >>**.

Kotlin

Unset
```
Salesmanago.updateContactData(
    name = "John Doe",
    email = "john.doe@email.com",
    phone = "+44123456789",
    standardDetails = mapOf(
        "ecoprogrammember" to "yes",
        "size" to "XL"
    )
)
```

Java

Java
```
Salesmanago.INSTANCE.updateContactData(
    "John Doe",
    "john.doe@email.com",
    "+44123456789",
    new HashMap<String, String>() {
        put("ecoprogrammember", "yes");
        put("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 after the user's email address has been transferred to SALESmanago using the* `updateContactProperties` *or* `updateContactData` *method.*

Kotlin

```
Unset
Salesmanago.updateContactMarketingConsents(
    email = OptInOption.DENIED,
    mobile = OptInOption.NO_ANSWER,
)
```

Java

```
Java
Salesmanago.INSTANCE.updateContactMarketingConsents(
    OptInOption.DENIED, //Email Marketing
    OptInOption.NO_ANSWER, //Mobile Marketing
    OptInOption.GRANTED //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 after the user's email address has been transferred to SALESmanago using the* `updateContactProperties` *or* `updateContactData` *method.*

Kotlin

Unset
```
Salesmanago.updateContactAdditionalConsents(
    listOf(
        AdditionalConsent(
            "demo custom consent",
            OptInOption.GRANTED
        )
    )
)
```

Java

Java
```java
Salesmanago.INSTANCE.updateContactAdditionalConsents(
    Arrays.asList(
        new AdditionalConsent(
            "demo custom consent",
            OptInOption.GRANTED
        )
    )
);
```

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

Kotlin

```
Unset
Salesmanago.addTags(listOf("tag_to_add"))
Salesmanago.removeTags(listOf("tag_to_remove"))
```

Java

```
Java
Salesmanago.INSTANCE.addTags(Arrays.asList("tag_to_add"));
Salesmanago.INSTANCE.removeTags(Arrays.asList("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 [Kotlin] / Parameter number [Java]	Field	Limits	Description
`contactData` / First parameter	`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` / 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.

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.

Kotlin

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

Java

```java
Java
Salesmanago.INSTANCE.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