Android – Using the SDK

PRODUCT MANUAL | 25/01/2022

1 Foreword

Copyright

The contents of this manual cover material copyrighted by Selligent. Selligent reserves all intellectual property rights on the manual, which should be treated as confidential information as defined under the agreed upon software licence/lease terms and conditions.

The use and distribution of this manual is strictly limited to authorised users of the Selligent Interactive Marketing Software (hereafter the “Software”) and can only be used for the purpose of using the Software under the agreed upon software licence/lease terms and conditions. Upon termination of the right to use the Software, this manual and any copies made must either be returned to Selligent or be destroyed, at the latest two weeks after the right to use the Software has ended.

With the exception of the first sentence of the previous paragraph, no part of this manual may be reprinted or reproduced or distributed or utilised in any form or by any electronic, mechanical or other means, not known or hereafter invented, included photocopying and recording, or in any information storage or retrieval or distribution system, without the prior permission in writing from Selligent.

Selligent will not be responsible or liable for any accidental or inevitable damage that may result from unauthorised access or modifications.

User is aware that this manual may contain errors or inaccuracies and that it may be revised without advance notice. This manual is updated frequently.

Selligent welcomes any recommendations or suggestions regarding the manual, as it helps to continuously improve the quality of our products and manuals.

2 Table of Contents

3 Intro

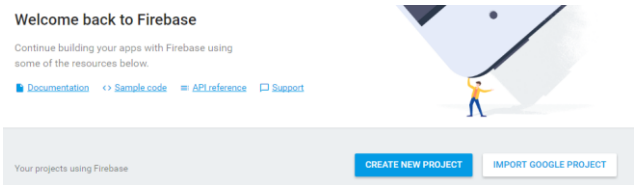
The purpose of this document is to detail how to install the SDK into your app and how to easily start using it.

- For more detailed implementation of the SDK please refer to the “Android - MobileSDK Reference.pdf” document.
- For an example of implementation check the “AndroidSDKTemplate” project.

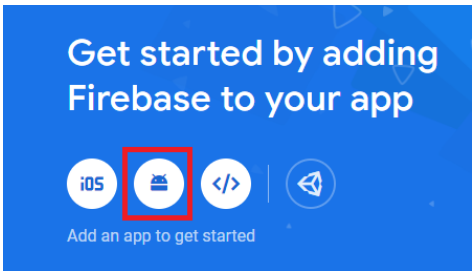
4 Creating an application

4.1 Firebase (Google)

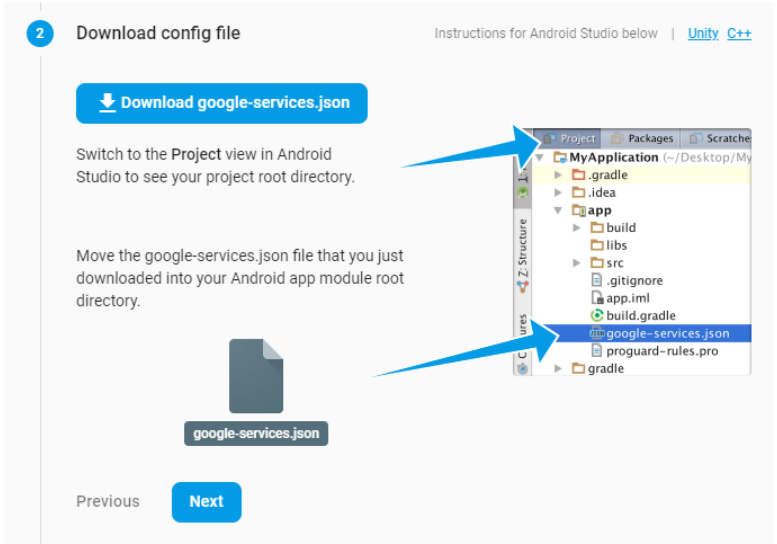
- If you already use push notifications (either yourself or through a third party library), simply re-use the Server Key and Sender ID you already have. If you don't already use it, we suggest you switch to integrating the json file given by the Firebase console.
- If you don't have any yet, go to https://console.firebase.google.com/ and sign in with a Google account.
- If you already have a project on Google Developer Console, click on "Import Google Project", otherwise click on "Create new project" and follow instructions.



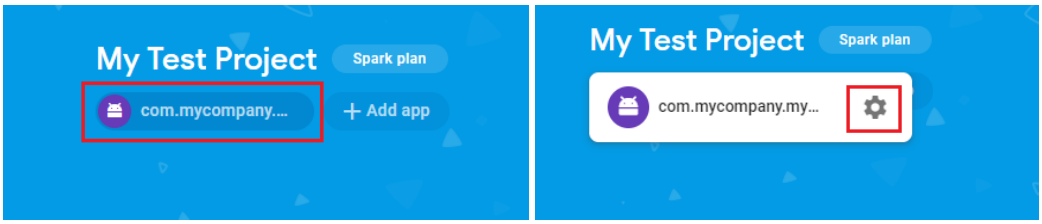
- Click on the Android icon and enter your package name.



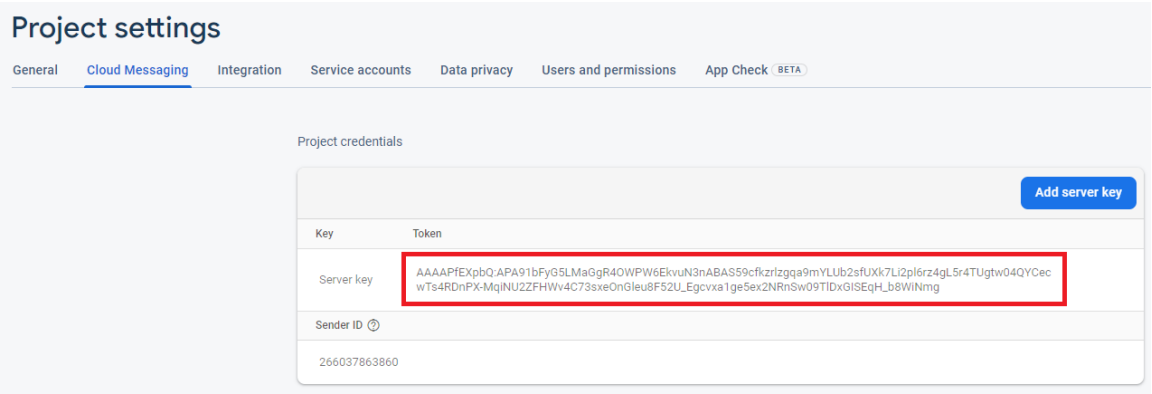
- Download the JSON file, put it in your app and update the gradle files as instructed on Firebase.



- Click on your app, then on the gear icon.



- Click on the tab “Cloud Messaging”, you will see the Server key and the Sender ID.



- Note the Server key, you will have to give it to the Selligent platform.

4.2 Huawei

Since 3.8, our SDK supports Huawei Mobile Services (HMS) and will use those when Google Play Services are not available. For your app to use HMS, you need to:

- Register on Huawei AppGallery Connect
- Create a project
- Enable Push Kit for that project
- Create an app in that project, using the package name of your Android app
- Enter the SHA-256 certificate fingerprint of your app
- Copy the json file in your app
- Note the Client ID and the Client secret, you will have to give those to the Selligent Platform.

For more information, consult the Huawei documentation.

NOTE: Sending push through Huawei Services is only supported by the Selligent Mobile Platform for customers using SDC. Ask your Selligent contact for more information about SDC.

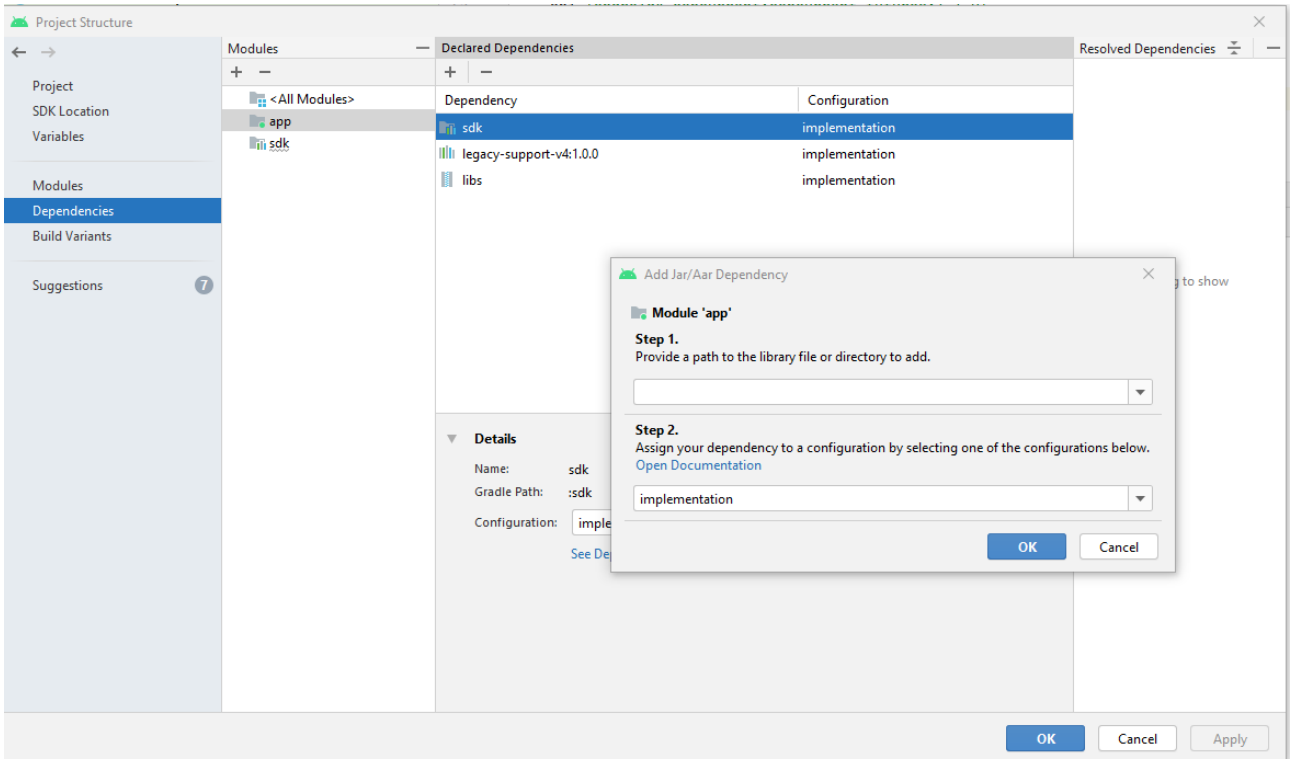
5 Including the SDK in your project

5.1 SDK library

NOTE: JCenter being made read-only since March 31st, you won't find new versions of our SDK over there anymore.

5.1.1 Import the Selligent library

Create a new module that will contain the aar file (you need to have it to do that). To do this, open the Project Structure dialog and add a new dependency to the app module and choose "JAR/AAR dependency".



And select the file.

Once it is done, synchronize and build the project.

5.1.2 minSdkVersion

Due to changes in Firebase and Google-Play-Services and to support Huawei Services, the minSdkVersion is now 19.

5.1.3 Gradle

The SDK was built using the Gradle Plugin 7.0.4

5.2 Other libraries

You need to add some external dependencies in your app gradle file.

- Firebase messaging and Firebase-core

If your version of Gradle is 5 or higher, you can simply add:

```
implementation platform('com.google.firebase:firebase-bom:29.0.0')
implementation 'com.google.firebase:firebase-messaging'
```

If you are using a lower version of Gradle, you need to specify the version of Firebase-messaging

```
com.google.firebase:firebase-messaging:23.0.0
```

- The version of Firebase must be at least 19 to be compatible with our SDK 3.x. Firebase 19 requires the use of the “AndroidX” libraries instead of the old “support” ones. The SDK was adapted to work with those new libraries, which means it will NOT work with the ”support” ones anymore.

  For the same reason, any google-play-services library must be at least 17.

  > **Technical Note:** If you haven’t adapted your app to use AndroidX yet, please use a lower version of our SDK.

- If you followed the instructions given to you by Firebase (cf. Creating a Google application) to update your gradle files, you should have the following:
  - In the build.gradle file at project level:

    ```
    dependencies {
        classpath 'com.google.gms:google-services:X.Y.Z'
        }
    ```

  - In the build.gradle file at app level, at the bottom:

    ```
    apply plugin: 'com.google.gms.google-services'
    ```

- If you plan on sending Map type push, you need a dependency to play-services-maps.

com.google.android.gms:play-services-maps

- GSON

com.google.code.gson:gson

- CardView

androidx.cardview:cardview

- WorkManager

These two dependencies are mandatory if you are going to send encrypted push to your app or rich push.

androidx.work:work-runtime:2.7.1

androidx.concurrent:concurrent-futures:1.1.0

These two replace FirebaseJobDispatcher which is deprecated and not used anymore by the Selligent SDK.

- PlotProject

If you want to use geolocation, you will need a dependency to the PlotProject library:

com.plotprojects:plot-android:3.10.0

For Gradle to find that dependency, you must add a reference to the Maven Plot repository. In the list of repositories in your top build.gradle file, add

```
allprojects {
    repositories {
        …
        maven {
            url 'https://maven-repo.plotprojects.com'
        }
    }
}
```

- Huawei

For the SDK to work with HMS, the following changes must be done to the build.gradle files:

At the root:

```
buildscript {
    repositories {
        …
        maven {
            url 'https://developer.huawei.com/repo/'
        }
    }

    dependencies {
        …
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
    }

}

allprojects {
    repositories {
        …
        maven {
            url 'https://developer.huawei.com/repo/'
        }
    }

}
```

In the app module:

```
dependencies {
    …
    implementation 'com.huawei.hms:base:6.2.0.300'

    implementation 'com.huawei.hms:push:6.1.0.300'
}

Apply plugin: 'com.huawei.agconnect'
```

NOTE: Sending push through Huawei Services is only supported by the Selligent Mobile Platform for customers using SDC. Ask your Selligent contact for more information about SDC.

6 How to use the SDK

6.1 Starting the SDK

6.1.1 Extending Application

The SDK needs to be started in a class extending Application.

If you do not already have one, create a new class, for example “MyApplication” that will extend Application:

```
public class MyApplication extends Application
{
    @Override
    public void onCreate()
    {
        [setup the SDK here]
        super.onCreate();
    }
}
```

On the OnCreate event, setup the SDK (cf. Start).

In the AndroidManifest.xml file, add the following:

```
<application
  android:name=".MyApplication"

  ...
```

6.1.2 Start

To start the SDK, in your class extending Application, use the following:

```
SMManager.getInstance().start(settings, this);
```

this is of course the instance of the class extending Application.

settings is an SMSettings object, proposing the following mandatory members:

WebServiceUrl must be the URL of the Selligent web service that will be called. It is given by Selligent.

GoogleApplicationId is deprecated, you can leave it to null as long as you use the JSON file given by Firebase.

ClientId is the public key allowing the connection to the web service.

PrivateKey is the private key allowing the connection to the web service.

Example:

```
SMSettings settings = new SMSettings();
settings.WebServiceUrl = "https://www.some.web.service.com";
settings.ClientId = "SomeClientId";
settings.PrivateKey = "SomePrivateKey";

SMManager.getInstance().start(settings, this);
```

6.1.2.1 Optional settings

There are optional settings on SMSettings:

public SMClearCache ClearCacheIntervalValue

You can set it to change the way the SDK manages the cache. It is recommended to leave it to its default value of Auto.

public SMInAppRefreshType InAppMessageRefreshType

Setting this value will enable the In-App messages. It tells how often the SDK must retrieve the In-App messages.

public SMInAppRefreshType InAppContentRefreshType

Setting this value will enable the In-App contents. It tells how often the SDK must retrieve the In-App contents.

public SMRemoteMessageDisplayType RemoteMessageDisplayType

Setting this value will enable/disable the automatic display of remote messages as they are received when the app is in foreground.

- Automatic: the message will be displayed right away
- Notification: a notification will be created and the message will be displayed after clicking on it.
- None: nothing will be done, the app will have to manage the display (using SMManager.getInstance().displayLastReceivedRemotePushNotification() and SMManager.getInstance().getLastRemotePushNotification()).

When the app is in background, a notification will always be displayed unless DoNotListenToThePush is true.

public boolean LoadCacheAsynchronously

Setting this value to true will make the SDK load the cache at start using a separate thread, making the loading asynchronously. This will improve the performance but has an impact on how you retrieve contents, especially the In-App contents. When set to true, it is recommended to use the method returning the In-App contents using a callback.

Its default value is false.

public boolean DoNotFetchTheToken

Setting this value to true will prevent the SDK from fetching the token. Instead, you will have to do it yourself and give it to the SDK using the following method:

```
SMManager.getInstance().setFirebaseToken(String token)
```

Its default value is false.

public boolean DoNotListenToThePush

Setting this value to true will prevent the SDK from listening to the push. Instead, you will have to do it yourself and either give it to the SDK using the following method

```
SMManager.getInstance().displayNotification(Context context, Intent intent)
```

or use the SDK to retrieve the payload from the intent and then manage the notification yourself.

Its default value is false.

public boolean ConfigureGeolocation

This will tell the SDK to initialize the geolocation capabilities. For it to work, you need to add a dependency to the PlotProject library (cf. Other libraries) and a setting file (cf. Geolocation).

Default value is false.

public boolean EnableNotifications

This will tell the SDK if the push notifications must be enabled or not once the token is retrieved. The default value is true (same behaviour as versions prior to 3.4.0). If you set the value to false, you will have to call at some point SMManager.getInstance.enableNotifications() to allow the push notifications to be sent to the device.

public boolean AddInAppMessageFromPushToInAppMessageList

This will tell the SDK if an In-App Message received through a push notification must be be added to the list of In-App Messages retrieved separately and will be available via SMManager.getInstance().getInAppMessages(SMInAppMessageReturn).

Default value is false.

public SMWebViewNavigationOverride WebViewNavigationOverride

This will allow the SDK to override the navigation in the WebViews displayed by the In-App messages. It will allow you to check the link clicked by the user and decide how the SDK will handle (or not) the navigation. Leave it to null if you don't want to override the navigation, no matter the link.

Example:

```
settings.WebViewNavigationOverride = new SMWebViewNavigationOverride()
{
   @Override
   public SMWebViewNavigationOption shouldHandleURL(Context context, String url)
   {
      if (url.contains("something://my/deep/link"))
      {
         return SMWebViewNavigationOption.StopNavigationAndExecuteDeeplink;
      }
      else if (url.contains("marvel.com"))
      {
         Toast.makeText(context, "Don't go there, you have to work!", Toast.LENGTH_SHORT).show();
         return SMWebViewNavigationOption.StopNavigation;
      }
      else
      {
         return SMWebViewNavigationOption.ResumeNavigation;
      }
   }
};
```

There are also some optional settings on SMManager:

SMManager.*DEBUG* = **true**;

Setting this to true will add the SDK logs to the logcat (it is better to do that before calling the start method to see everything logged when the SDK starts).

SMManager.*MAIN_ACTIVITY* = YourMainActivity.class;

Setting this will allow the SDK to know which activity is your main one so that it performs certain operations only when that one is active. For example, a dialog might need to be displayed to the user to update Google-Services or a security protocol on old devices. By specifying the MainActivity, the SDK will only show them on your main Activity and not on a splash or login screen. If it is not specified, the first activity being active will be used.

6.2 Device id

The device id is given by the Selligent Mobile Platform. There are two ways for you to retrieve it, should you need it:

- SMManager.getInstance().getDevideId()

  This method will return the device id stored by the SDK. Note that, as the device id is given by the Selligent Mobile Platform, it is possible that the value returned is either empty when the call is made.

- SMManager.getInstance().getObserverManager().observeDeviceId()

  This will allow you to observe the device id and receive it when it is received by the SDK.

6.3 Push notifications

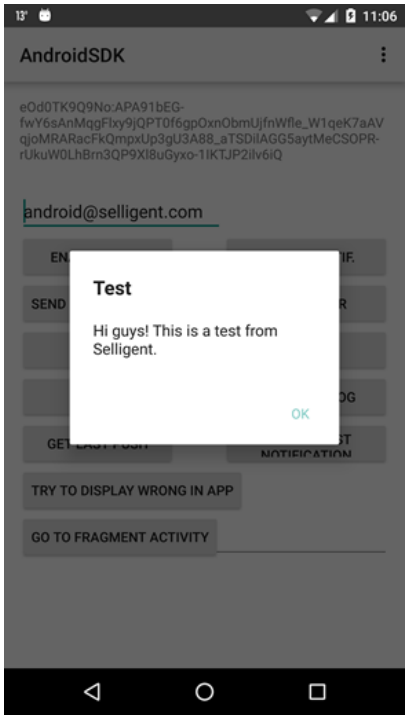
Push notifications are messages sent from the server to a device.

When a push is received, if the app is in background or inactive, we create a notification. When you click on it, the app opens and the message is displayed. If the app is in foreground, it will either directly show the message, create a notification, or do nothing, depending on the value of the setting RemoteMessageDisplayType (Automatic, Notification or None) given to the SMSettings object when calling the “start” method of SMManager.

Starting with version 3.7, you can also decide to manage everything yourself by using our method to retrieve the payload from the Intent and then do whatever you want with it (cf. Manual management of the push).

Also starting with version 3.7, the SDK will now check if the notifications were enabled/disabled in the settings of the OS for the current app and inform the Selligent Mobile Platform if needed.

Once the message is displayed, it is either in a dialog which, by default, looks like that:



or in a dedicated Activity, depending of the type of the push (Map, Image, HTML and URL open in a dedicated Activity).

6.3.1 Permissions for Push notification

If you plan on sending “Map” type push, you might want to add one of the following permissions for the user’s location to be displayed on the map:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

or

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

If your app targets Android 12 (targetSdkVersion 31) and you requests ACCESS_FINE_LOCATION, you must also request ACCESS_COARSE_LOCATION. In the other cases, only one is needed.

Coarse location is less precise than fine location. Note that if you do not add any, the map will still be displayed, just not the user’s location.

6.3.2 Listening to the push notifications and displaying the linked In-App message or executing the main action.

To check if a notification was received and to display it, you must add some code inside your activities (or, better, in any base Activity class you have).

First, you will need to add a member to your class with a type SMForegroundGcmBroadcastReceiver. This receiver listens to the push while the app is in foreground and also manages behaviour of the SDK when the connectivity changes. So it is important to instantiate it even if you decide to listen to the push yourself and give it to the SDK.

```
SMForegroundGcmBroadcastReceiver receiver;
```

Then, update the onStart, onStop and onNewIntent events like this:

```
@Override
protected void onStart()
{
   super.onStart();



   [...]

   if (receiver == null)
   {
      receiver = new SMForegroundGcmBroadcastReceiver(this);
   }
   registerReceiver(receiver, receiver.getIntentFilter());



   SMManager.getInstance().checkAndDisplayMessage(getIntent(), this);

}

@Override
protected void onStop()
{
   super.onStop();



   [...]
   unregisterReceiver(receiver);

}
```

```
@Override
protected void onNewIntent(Intent intent)
{
   super.onNewIntent(intent);

   SMManager.getInstance().checkAndDisplayMessage(intent, this);
}
```

The method checkAndDisplayMessage will check if information linked to a push is present in an intent and act accordingly. It can be to display an In-App message, execute a deep link, etc.

Since 3.6, checkAndDisplayMessage proposes an overload that allows you display the In-App message yourself. Simply implement the SMInAppMessageDisplay interface and make onBeforeDisplay return false to prevent the SDK from displaying the In-App. If it returns false, it will display it like usual.

Example:

```
SMManager.getInstance().checkAndDisplayMessage(getIntent(), this, new SMInAppMessageDisplay()
{
   @Override
   public boolean onBeforeDisplay(SMInAppMessage message)
   {
      Toast.makeText(BaseActivity.this, "In-App message '" + message.title + "' displayed by the app",
Toast.LENGTH_LONG).show();
      ShowDialog(message.getTitle(), message.getBody());
      return false;
   }
});
```

NOTE: Returning false is NOT the same as not calling checkAndDisplayMessage. The method also sends events to the Selligent Mobile platform and executes the action behind the push or its buttons (if any).

6.3.2.1 Extending SMBaseActivity

There is a class SMBaseActivity in the SDK that already does everything described in the previous point and displays the push notifications. You can make your activities extend it to avoid writing the code described in those points. SMBaseActivity extends the AndroidX version of AppCompatActivity.

```
public class MainActivity extends SMBaseActivity
{

}
```

If you extend SMBaseActivity, nothing else needs to be done.

6.3.3 Customization

If the app is in background when a push is received, an icon will appear in the status bar and a notification will be added to the Notification drawer. Clicking on it will call a specific Activity which will display the message. Both can be customized.

6.3.3.1 Setting a specific icon

To customize that icon, call these methods after starting the SDK in your Application class:

```
SMManager.getInstance().setNotificationSmallIcon(R.drawable.some_icon);
```

This sets the small icon that will be used for the notifications in the notification bar. If not set, the default icon of the SDK will be used (the head of the Android robot).

```
SMManager.getInstance().setNotificationLargeIcon(R.drawable.some_large_icon);
```

This sets the large icon that will be used for the notifications. If not set, no large icon will be specified to Android, so the small one will be used.

```
SMManager.getInstance().setNotificationIconColor(someIntColor);
```

This sets the color of the icon when it is displayed in the Notification center. It is a simple exposure of the setColor method of NotificationBuilder from the Android API (cf. Android doc).

6.3.3.2 Setting a specific Activity

By default, the Activity called to display the message of a push is NotificationActivity. If you want to keep it, to be able to go back to your application from it, it must be declared in the manifest as a child of an activity from your app (in the example, MainActivity).

```
<application
  ...>

  ...
  <activity
    android:name="com.selligent.sdk.NotificationActivity"
    android:parentActivityName=".MainActivity">
    <meta-data android:name="android.support.PARENT_ACTIVITY"
          android:value=".MainActivity"></meta-data>
  </activity>

</application>
```

You can also set any Activity of your app to be called instead (in which case, no need to add the previous code to your manifest).

In your Application class, after starting the SDK, do this:

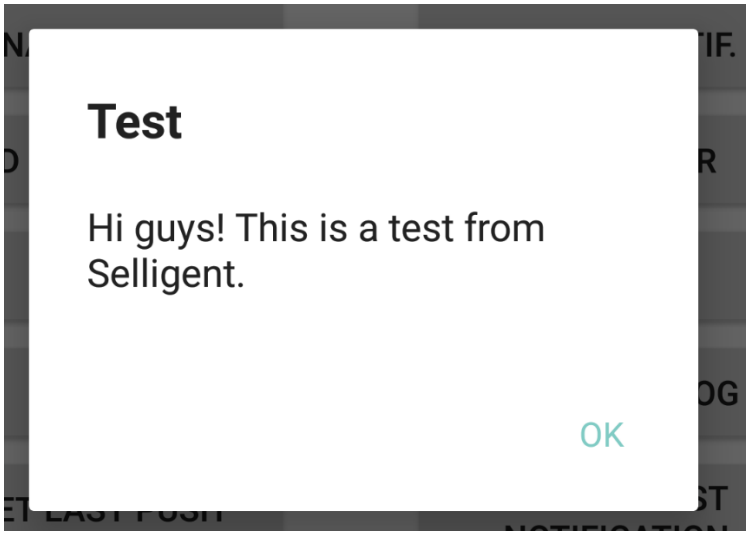
```
SMManager.NOTIFICATION_ACTIVITY = MyActivity.class;
```

If you used SMRemoteMessageDisplayType.Notification as value for RemoteMessageDisplayType (cf. Optional settings), you can also set this property in your base activity with the value returned by getClass(), that way the current activity will be the one called when clicking on the notification when the application is in foreground.

6.3.4 Design customization

6.3.4.1 Dialog

Some push messages are displayed as a dialog box which, by default, looks like this:



NOTE: the background and text colors will be the one defined in your theme.

This is a default layout made to have a “Material” look. It is entirely customizable. Its definition is in the file “styles.xml”. Here is its content:

```
<style name="Selligent.Dialog.Container">
    <item name="android:paddingLeft">0dp</item>
    <item name="android:paddingRight">0dp</item>
    <item name="android:paddingTop">0dp</item>
    <item name="android:paddingBottom">0dp</item>
</style>
<style name="Selligent.Dialog.Title">
    <item name="android:singleLine">true</item>
    <item name="android:ellipsize">end</item>
    <item name="android:layout_marginLeft">24dp</item>
    <item name="android:layout_marginRight">24dp</item>
    <item name="android:layout_marginTop">24dp</item>
```

```
    <item name="android:layout_marginBottom">0dp</item>
    <item name="android:textSize">20sp</item>
    <item name="android:textColor">?android:attr/textColorPrimary</item>
    <item name="android:typeface">sans</item>
    <item name="android:textStyle">bold</item>
    <item name="android:shadowRadius">0</item>
    <item name="android:gravity">start</item>
</style>
<style name="Selligent.Dialog.UpperDivider">
    <item name="android:layout_height">0dp</item>
    <item name="android:visibility">gone</item>
</style>
<style name="Selligent.Dialog.BodyScrollView">
    <item name="android:clipToPadding">false</item>
    <item name="android:layout_marginTop">20dp</item>
    <item name="android:layout_marginLeft">24dp</item>
    <item name="android:layout_marginRight">24dp</item>
    <item name="android:layout_marginBottom">24dp</item>
</style>
<style name="Selligent.Dialog.Body">
    <item name="android:textSize">16sp</item>
    <item name="android:typeface">sans</item>
    <item name="android:textColor">?android:attr/textColorPrimary</item>
    <item name="android:maxLines">10</item>
</style>
<style name="Selligent.Dialog.LowerDivider">
    <item name="android:layout_height">0dp</item>
</style>
<style name="Selligent.Dialog.ButtonScrollView">
</style>
<style name="Selligent.Dialog.ButtonContainer">
    <item name="android:gravity">end</item>
    <item name="android:paddingLeft">8dp</item>
    <item name="android:paddingRight">8dp</item>
    <item name="android:paddingTop">8dp</item>
    <item name="android:paddingBottom">8dp</item>
</style>
<style name="Selligent.Dialog.ButtonRow">
    <item name="android:layout_width">wrap_content</item>
</style>
<style name="Selligent.Dialog.Button">
    <item name="android:layout_height">36dp</item>
    <item name="android:minWidth">64dp</item>
    <item name="android:paddingLeft">8dp</item>
    <item name="android:paddingRight">8dp</item>
    <item name="android:radius">2dp</item>
    <item name="android:focusable">true</item>
    <item name="android:clickable">true</item>
    <item name="android:gravity">center_vertical|center_horizontal</item>
    <item name="android:textSize">14sp</item>
    <item name="android:typeface">sans</item>
```

```
    <item name="android:textAllCaps">true</item>
    <item name="android:textColor">#ff80cbc4</item>
    <item name="android:background">?android:attr/selectableItemBackground</item>
</style>
```

To customize it, in your own file “styles.xml”, override the styles you want to modify. Note that you have to copy the whole content of those styles, not only the items you want to change, otherwise the others will be lost.

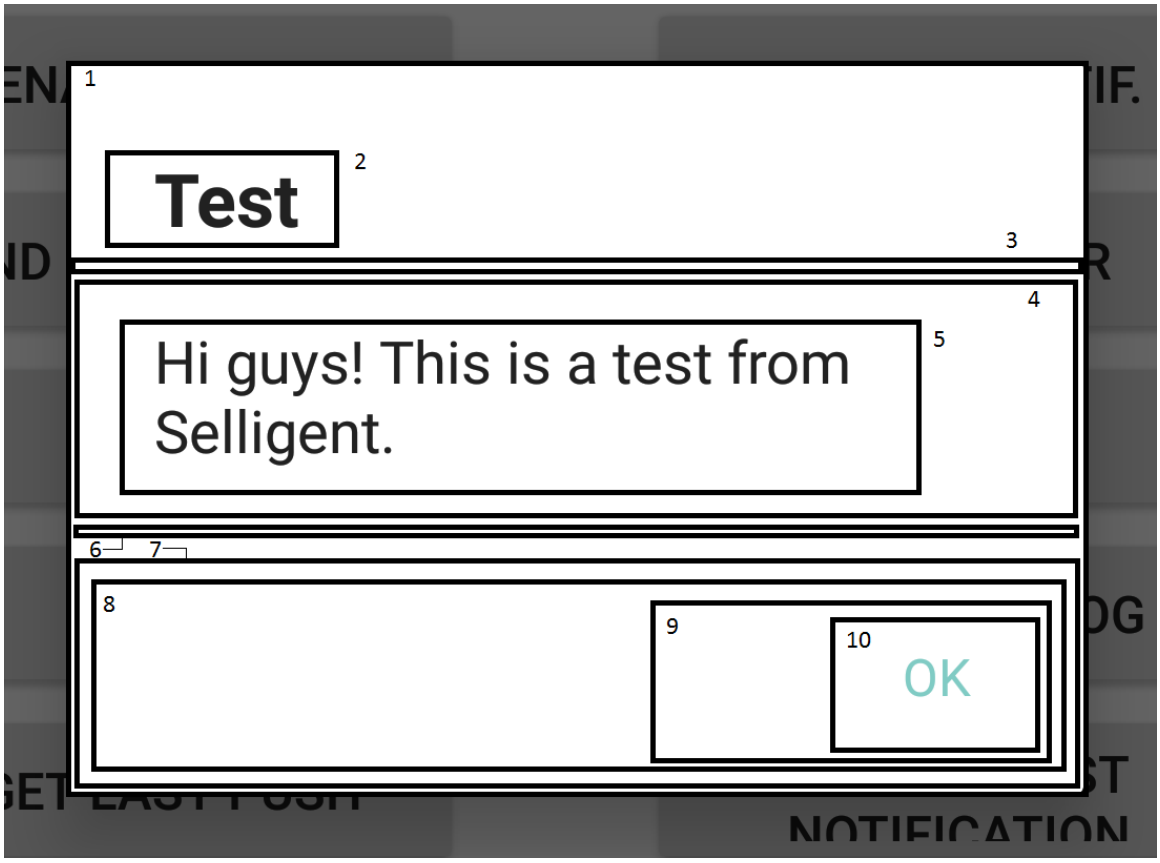
Example:

if you simply want to change the text color of a button to red, you still have to add in your file the whole style:

```
<style name="Selligent.Dialog.Button">
    <item name="android:layout_height">36dp</item>
    <item name="android:minWidth">64dp</item>
    <item name="android:paddingLeft">8dp</item>
    <item name="android:paddingRight">8dp</item>
    <item name="android:radius">2dp</item>
    <item name="android:focusable">true</item>
    <item name="android:clickable">true</item>
    <item name="android:gravity">center_vertical|center_horizontal</item>
    <item name="android:textSize">14sp</item>
    <item name="android:typeface">sans</item>
    <item name="android:textAllCaps">true</item>
    <item name="android:textColor">#ff0000</item>
    <item name="android:background">?android:attr/selectableItemBackground</item>
</style>
```

The different styles are applied like this:

```
1  <style name="Selligent.Dialog.Container">
2  <style name="Selligent.Dialog.Title">
3  <style name="Selligent.Dialog.UpperDivider">
4  <style name="Selligent.Dialog.BodyScrollView">
5  <style name="Selligent.Dialog.Body">
6  <style name="Selligent.Dialog.LowerDivider">
7  <style name="Selligent.Dialog.ButtonScrollView">
8  <style name="Selligent.Dialog.ButtonContainer">
9  <style name="Selligent.Dialog.ButtonRow">
10 <style name="Selligent.Dialog.Button">
```



6.3.4.2 Activities

Some other type of messages (like Map, HTML, etc.) are displayed in their own activity, not in a dialog. Those activities extend AppCompatActivity and, therefore, need an AppCompat theme. So, in order to avoid any crash when displaying them, we force our own AppCompat theme: Theme.SMTheme. It has for parent Theme.AppCompat.Light. You might want to override it to reflect your own layout.

To do so, simply define that theme in your app and use as parent the appropriate AppCompat theme.

Examples:

If you use Theme.Holo.Light, define Theme.SMTheme like this (in styles.xml):

```
<style name="Theme.SMTheme" parent="Theme.AppCompat.Light"></style>
```

If you use a customized Holo theme whose parent is Theme.Holo.Light, do this:

```
<style name="Theme.SMTheme" parent="Theme.AppCompat.Light">
   <item name="colorPrimaryDark">@color/yourPrimaryDarkColor</item>
   <item name="colorPrimary">@color/yourPrimaryColor</item>
   <item name="android:textColorPrimary">@color/yourTextColor</item>

</style>
```

(cf. https://developer.android.com/training/material/theme.html)

If you already use an AppCompat theme, then simply use it as parent:

```
<style name="Theme.SMTheme" parent="YourTheme"></style>
```

6.3.5 Retrieving the Firebase Cloud Messaging (FCM) token from the SDK

There are two ways to retrieve the FCM token: listening to a broadcast and calling a method.

NOTE: you will see "GCM" instead of "FCM" in the broadcast and method names, that is because "GCM" stands for "Google Cloud Messaging", which was the previous name of FCM, before Google moved the functionality to Firebase.

6.3.5.1 Broadcast -- deprecated

Technical Note: Due to LocalBroadcastManager being deprecated with AndroidX, this broadcast, although still sent, is now also deprecated. Use the observer instead.

You can listen to SMManager.BROADCAST_EVENT_RECEIVED_GCM_TOKEN (its value is "SMReceivedGCMToken"). This broadcast is sent after reception of the token from FCM and only if it is different from the one already stored.

It's a local broadcast and, therefore, must be listened to using a LocalBroadcastManager.

The value of the token can be retrieved from the intent received by using SMManager.BROADCAST_DATA_GCM_TOKEN (its value is "SMDataGCMToken").

Example: *considering you have a class EventReceiver:*

```
public class EventReceiver extends BroadcastReceiver
{

   public void onReceive(Context context, Intent intent)
   {
      String action = intent.getAction();

      switch (action)
      {
         case SMManager.BROADCAST_EVENT_RECEIVED_GCM_TOKEN:

            String gcmToken = intent.getStringExtra(SMManager.BROADCAST_DATA_GCM_TOKEN);

            //Do some stuff
```

```
            break;
        }
    }
}
```

6.3.5.2 Observer

Starting with version 3.0.0, the SDK proposes to use observers instead of listening to the (deprecated) broadcasts. To receive the token, you can do it like this:

```
final Observer<String> tokenObserver = new Observer<String>()
{
    @Override
    public void onChanged(String token)
    {
        //Do something with the token...
    }
};
SMManager.getInstance().getObserverManager().observeToken(this, tokenObserver);
```

Note: “this” can be used if that code is in an Activity, as the AndroidX activities implement LifecycleOwner.

By default, the onChanged event of the observer is triggered when a new token is received after the observer is created. If a new token was received before the observer is created, the event will not be triggered. If you want to change this behaviour, use the overload that requires a Boolean as third parameter and set it to true (default value is false). For more information on the two different behaviours, see Observers.

6.3.5.3 SMManager.getInstance().getGCMToken

This method will return the token stored by the SDK.

Note that, as the processing to fetch the token from FCM is asynchronous, it is possible that the value returned is either empty or not up-to-date when the call is made.

6.3.6 Enabling/disabling the notifications

By default, the notifications are enabled. They can be disabled at any time using the following method:

```
SMManager.getInstance().disableNotifications();
```

They are enabled again by doing:

```
SMManager.getInstance().enableNotifications();
```

6.3.7 Setup for special push

6.3.7.1 Map

If you expect to receive Map type notifications, you will need to specify a Google Map key in your manifest. This key needs to be generated with the google developer console.

To create it, please follow the steps described in this page:
https://developers.google.com/maps/documentation/android-api/signup#key-biz

At the end of this procedure you need to add this generated key under the APPLICATION xml tag in the AndroidManifest.xml like this:

```
<meta-data
   android:name="com.google.android.geo.API_KEY"
   android:value="xxxxxxxxxxxxxxx"/>
```

6.3.7.2 Event

When displayed, a notification, an In-App message can contain buttons. One type of button can send a specific value through the app, for you to execute some code when you receive it.

NOTE: Due to LocalBroadcastManager being deprecated with AndroidX, the broadcast propagating the value, although still sent, is now deprecated. So, the value is now also sent using LiveData and you can listen to it using an observer.

This broadcast is sent locally using LocalBroadcastManager. To listen to it, you need to add a BroadcastReceiver to your app, specify the action (the aforementioned value) and register it using LocalBroadcastManager.

The action needs to be the name of the event specified at the creation of the push.

Example: *if the value of the broadcast is "CustomEvent" and considering you have a class EventReceiver*

```
public class EventReceiver extends BroadcastReceiver
{

   public void onReceive(Context context, Intent intent)
   {
      String action = intent.getAction();

      switch (action)
      {
         case "CustomEvent":

            //Perform the actions requested by your app

               break;

      }

   }

 }
```

And in your activities/base activity:

```
EventReceiver localReceiver;

...

@Override
protected void onStart()
{
   super.onStart();



   [...]

   if (localReceiver == null)
   {
      localReceiver = new EventReceiver();
   }
   IntentFilter filter = new IntentFilter();
   filter.addAction("CustomEvent");
   LocalBroadcastManager.getInstance(this).registerReceiver(localReceiver, filter);
}
```

As mentioned above, the event can also be listened to using an observer. Here is how it works:

```
final Observer<String> customEventObserver = new Observer<String>()
{
   @Override
   public void onChanged(String event)
   {
      switch(event)
      {
         case "SomeEvent1":
            //Do something...
            break;

         case "SomeEvent2":
            //Do something else...
            break;
      }
   }
};
SMManager.getInstance().getObserverManager().observeEvent(this, customEventObserver);
```

NOTE: “this” can be used if that code is in an Activity, as the AndroidX activities implement LifecycleOwner.

6.3.8 Broadcasts -- deprecated

Technical Note: Due to LocalBroadcastManager being deprecated with AndroidX, all broadcasts, although still sent, are now also deprecated. Use the observers instead.

Some specific broadcasts are sent during the management of the push notifications (reception, display and interaction):

BROADCAST_EVENT_RECEIVED_REMOTE_NOTIFICATION : When a push is received, it contains its id and title.

This broadcast is only useful if RemoteMessageDisplayType is set to None, so you can decide when to display the push message. In all other cases, the SDK manages everything itself, so it is not needed.

BROADCAST_EVENT_BUTTON_CLICKED : When a button is clicked, it contains an SMNotificationButton object

BROADCAST_EVENT_WILL_DISPLAY_NOTIFICATION : When a message is about to be displayed

BROADCAST_EVENT_WILL_DISMISS_NOTIFICATION : When a message is about to be dismissed

BROADCAST_EVENT_RECEIVED_GCM_TOKEN : When the token is received, it contains the token (cf. Retrieving the Firebase Cloud Messaging (FCM) token)

For more info regarding the broadcasts, go to Broadcasts

6.3.9 Observers

By default, the onChanged event of the observer is triggered when there is a new value after the observer is created. If there is a new value before the observer is created, the event will not be triggered. If you want to change this behaviour, use the overload that requires a Boolean as third parameter and set it to true (default value is false). For more information on the two different behaviours, see Observers.

- Clicked button

  When a button is clicked, the corresponding SMNotificationButton object is sent through the app to allow it to react to it. Observe it using:

```
SMManager.getInstance().getObserverManager().observeClickedButton(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<SMNotificationButton> observer)
```

- Will display message

  This is triggered when a message is about to be displayed. Observe it using:

```
SMManager.getInstance().getObserverManager().observeDismissedMessage(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<Void> observer)
```

- Will dismiss message

  This is triggered when a message is about to be dismissed. Observe it using:

```
SMManager.getInstance().getObserverManager().observeDisplayedMessage(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<Void> observer)
```

- FCM token

  When the token is received, it is sent throughout the app (cf. Retrieving the Firebase Cloud Messaging (FCM) token). Observe it using:

```
SMManager.getInstance().getObserverManager().observeToken(@NonNull LifecycleOwner
lifecycleOwner, @NonNull Observer<String> observer)
```

- Custom event

  A custom event can be sent using the value associated to a button or a main action. Observe it using:

```
SMManager.getInstance().getObserverManager().observeEvent(@NonNull LifecycleOwner
lifecycleOwner, @NonNull Observer<String> observer)
```

6.3.10 Manual display of a push notification

If you set RemoteMessageDisplayType to None and listen to the broadcast BROADCAST_EVENT_RECEIVED_REMOTE_NOTIFICATION, you will want to use the following method to display the push:

```
SMManager.getInstance().displayLastReceivedRemotePushNotification(activity);
```

This method will display the last push received (the SDK only stores the last one), using a dialog or a dedicated Activity, depending on its type.

```
SMManager.getInstance().getLastRemotePushNotification();
```

This method will return a HashMap containing the id and title of the push (the keys are “id” and “title”).

6.3.11 Manual management of the push

If you do not want the SDK to manage the push notifications, follow these steps:

- Tell the sdk to not listen to the push by setting DoNotListenToThePush to true on SMSettings

```
SMSettings settings = new SMSettings();
settings.DoNotListenToThePush = true;
```

- Implement a mechanism to listen to the push, for example using FirebaseMessagingService
- Retrieve the payload of the push from the intent containing it using our SDK

```
SMManager.getInstance().retrieveNotificationMessage(message.toIntent(), new
    OnSMNotificationMessageRetrieved()
    {
      @Override
      public void onSuccess(@NonNull SMNotificationMessage smNotificationMessage)
      {
        //Do some stuff
       }
```

```
    @Override
    public void onError(Exception e)
    {
       //Do some stuff
    }
});
```

The SMNotificationMessage object contains all the information of the push (title, body, buttons, etc.), including the one from the linked in-app message if there is one.

If no push from Selligent is present in the intent, neither onSuccess nor onError will be called.

- Use our methods to send the events to the Selligent Mobile platform to inform that the push was received, seen or a button was clicked.

```
SMManager.getInstance().setNotificationMessageAsReceived(smNotificationMessage);
SMManager.getInstance().setNotificationMessageAsSeen(smNotificationMessage);
SMManager.getInstance().setButtonAsClicked(smNotificationMessage,
smNotificationMessage.getNotificationButtons()[0]);
```

Note that these methods will send the event to the platform every time they are called, so make sure to only call them when needed.

6.4 In-App messages

In-App messages are messages retrieved periodically by the SDK.

They are retrieved when the app becomes active (ie. at start, when going from background to foreground and when the orientation changes) ONLY if the last refresh is older than the value set for InAppMessageRefreshType.

6.4.1 Permissions

There is no mandatory permission required to use the In-App messages in general. However, like for push notifications, some In-App messages will require special permissions to be displayed properly.

If you plan on sending “Map” type messages, you might want to add one of the following permissions for the user location to be displayed on the map:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

or

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Only one of the two is needed. Coarse location is less precise than fine location. Note that if you do not add any, the map will still be displayed, just not the user location.

6.4.2 Enabling/disabling the In App-messages

In app messages are disabled by default unless you set InAppMessageRefreshType on SMSettings.

It is highly recommended to avoid setting the value to Minutely for production. It is there for testing purpose only.

If you want to disable them at some point (or if you want to give the user the ability to do it), use the following method:

```
SMManager.getInstance().disableInAppMessages();
```

They are enabled again by doing (this can also be used to change the refresh type):

```
SMManager.getInstance().enableInAppMessages(SMInAppRefreshType.Daily);
```

Those two methods can be called anywhere in your app.

6.4.3 Reception of the messages

The SDK retrieved the In-App messages automatically, according to the setting InAppMessageRefreshType set when starting the SDK.

NOTE: Due to LocalBroadcastManager being deprecated with AndroidX, the broadcast, although still sent, is now also deprecated. Use the observer instead.

When messages are received, a broadcast is sent: BROADCAST_EVENT_RECEIVED_IN_APP_MESSAGE.

Use SMManager.BROADCAST_DATA_IN_APP_MESSAGES to retrieve them from the intent.

(cf. Local broadcasts for more information on how to use it)

To observe the In-App messages, call the following method:

```
SMManager.getInstance().getObserverManager().observeInAppMessages(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<SMInAppMessage[]> observer)
```

Only the title and id of each message are sent. They can be used to display some inbox (a list of the title of the messages).

The list received contains all the In-App messages that have not been read by the user yet. Therefore, it is possible for you to receive some messages that you already got earlier.

By default, the onChanged event of the observer is triggered when new In-App messages are received after the observer is created. If they are erceived before the observer is created, the event will not be triggered. If you want to change this behaviour, use the overload that requires a Boolean as third parameter and set it to true (default value is false). For more information on the two different behaviours, see Observers.

Since version 3.4.0 there is a new method to retrieve the In-App messages:

```
SMManager.getInstance().getInAppMessages(final SMInAppMessageReturn callbackEvent)
```

This method will return all In-App messages currently stored by the SDK, where the observer will only give the ones retrieved in the latest fetch.

NOTE: the messages retrieved using this method are the exact content of the In-App message cache. If you started the SDK with the value None for SMSettings.ClearCacheIntervalValue, it means there is no cache for the In-App messages and, therefore, this will return an empty array. In that case, you must rely only on the observer to retrieve the In-App messages.

6.4.4 Display of an In-App message

Once you have the In-App messages, you can display one using the following method:

```
SMManager.getInstance().displayMessage(messageId, activity);
```

messageId is the id of the In-App message to display (received by listening to the broadcast as discussed in Reception of the messages) and *activity* the Activity that will display it.

It will be displayed the way the push notifications are.

If you want to bypass the SDK and display the In-App messages yourself, it is possible since version 3.4.0. Now the SMInAppMessage object contains all the information you need to do it, whether they are returned by the observer or the get method mentioned in the previous point.

Several exist to help you do that:

- SMManager.getInstance().setInAppMessageAsSeen(SMInAppMessage inAppMessage): when you display the content of an In-App message, call this method to let our platform know that the In-App message was seen by the user.
- SMManager.getInstance().setInAppMessageAsUnseen(SMInAppMessage inAppMessage): call this method to let our SDK know that the In-App message must be considered as unseen (the equivalent of a "Set as unread" in an inbox)
- SMManager.getInstance().executeButtonAction(Context context, SMNotificationButton button, SMInAppMessage message): This method will execute the action behind a button.
- SMManager.getInstance().deleteInAppMessage(String messageId): call this method to delete one In-App message. It will not be returned anymore by the SDK when calling getInAppMessages().
- SMManager.getInstance().deleteInAppMessages(String[] messageIds) : call this method to delete several In-App messages at once. They will not be returned anymore by the SDK when calling getInAppMessages().

6.4.5 Broadcasts -- deprecated

> **Technical Note:** Due to LocalBroadcastManager being deprecated with AndroidX, all broadcasts, although still sent, are now also deprecated. Use the observers instead.

Some specific broadcasts are sent during the management of the In-App messages (reception, display and interaction):

BROADCAST_EVENT_RECEIVED_IN_APP_MESSAGE : When In-App messages are received, it contains an array of SMInAppMessages.

BROADCAST_EVENT_BUTTON_CLICKED : When a button is clicked, it contains an SMNotificationButton object

BROADCAST_EVENT_WILL_DISPLAY_NOTIFICATION : When a message is about to be displayed

BROADCAST_EVENT_WILL_DISMISS_NOTIFICATION : When a message is about to be dismissed

For more info regarding the broadcasts, go to Broadcasts

6.4.6 Observers

By default, the onChanged event of the observer is triggered when there is a new value after the observer is created. If there is a new value before the observer is created, the event will not be triggered. If you want to change this behaviour, use the overload that requires a Boolean as third parameter and set it to true (default value is false). For more information on the two different behaviours, see Observers.

- Received In-App messages

  When the SDK fetches the In-App messages, it sends an array of the Ids and labels to the app. Observe it using:

  ```
  SMManager.getInstance().getObserverManager().observeInAppMessages(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<SMInAppMessage[]> observer)
  ```

- Clicked button

  When a button is clicked, the corresponding SMNotificationButton object is sent to the app to allow it to react to it. Observe it using:

  ```
  SMManager.getInstance().getObserverManager().observeClickedButton(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<SMNotificationButton> observer)
  ```

- Will display message

  This is triggered when a message is about to be displayed. Observe it using:

  ```
  SMManager.getInstance().getObserverManager().observeDismissedMessage(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<Void> observer)
  ```

- Will dismiss message

  This is triggered when a message is about to be dismissed. Observe it using:

  ```
  SMManager.getInstance().getObserverManager().observeDisplayedMessage(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<Void> observer)
  ```

6.5 Events

The following method can be used to send specific messages to the web service.

SMManager.getInstance().sendSMEvent(SMEvent event);

The kind of event message sent to the user will depend of the class of object given to the method. All the different classes extend SMEvent. They are described in the following points.

NB: Since 1.3, the data passed to the SMEvent is Hashtable<String, String> (In earlier versions it was Hashtable<String, Object>).

6.5.1 Registration/Unregistration

6.5.1.1 SMEventUserRegister

This object is used to send a “register” event to the server with the e-mail of the user, potential data and a callback.

Example:

```
Hashtable<String, String> hash = new Hashtable<>();
hash.put("Key1", "Registration value 1");
hash.put("Key2", "Registration value 2");
hash.put("Key3", "Registration value 3");
SMEvent event = new SMEventUserRegister("user@company.com", hash,
      new SMCallback()
      {
         @Override
         public void onSuccess(String result)
         {
            [Do something here]
         }

         @Override
         public void onError(int responseCode, Exception exception)
         {
            [Do something here]
         }
      });
)
SMManager.getInstance().sendSMEvent(event);
```

Since 3.2.0, an overload constructor exists without the email.

6.5.1.2 SMEventUserUnregister

This object is used to send an “unregister” event to the server with the e-mail of the user, potential data and a callback.

Example:

```
Hashtable<String, String> hash = new Hashtable<>();
hash.put("Key1", "Unregistration value 1");
hash.put("Key2", "Unregistration value 2");
hash.put("Key3", "Unregistration value 3");
SMEvent event = new SMEventUserUnregister("user@company.com", hash,
    new SMCallback()
    {
      @Override
      public void onSuccess(String result)
      {
        [Do something here]
      }

      @Override
      public void onError(int responseCode, Exception exception)
      {
        [Do something here]
      }
    });
SMManager.getInstance().sendSMEvent(event);
```

Since 3.2.0, an overload constructor exists without the email.

6.5.2 Login/Logout

6.5.2.1 SMEventUserLogin

This object is used to send a “login” event to the server with the e-mail of the user, potential data and a callback.

Example:

```
Hashtable<String, String> hash = new Hashtable<>();
hash.put("Key1", "Login value 1");
hash.put("Key2", "Login value 2");
hash.put("Key3", "Login value 3");
SMEvent event = new SMEventUserLogin("user@company.com", hash,
    new SMCallback()
    {
      @Override
      public void onSuccess(String result)
      {
        [Do something here]
      }

      @Override
      public void onError(int responseCode, Exception exception)
```

```
        {
            [Do something here]
        }
    });
SMManager.getInstance().sendSMEvent(event);
```

Since 3.2.0, an overload constructor exists without the email.

6.5.2.2 SMEventUserLogout

This object is used to send a “logout” event to the server with the e-mail of the user, potential data and a callback.

Example:

```
Hashtable<String, String> hash = new Hashtable<>();
hash.put("Key1", "Logout value 1");
hash.put("Key2", "Logout value 2");
hash.put("Key3", "Logout value 3");
SMEvent event = new SMEventUserLogout("user@company.com", hash,
    new SMCallback()
    {
        @Override
        public void onSuccess(String result)
        {
            [Do something here]
        }

        @Override
        public void onError(int responseCode, Exception exception)
        {
            [Do something here]
        }
    });
SMManager.getInstance().sendSMEvent(event);
```

Since 3.2.0, an overload constructor exists without the email.

6.5.3 Custom

6.5.3.1 SMEvent

This object is used to send a custom event to the server with some data and a callback.

Example:

```
Hashtable<String, String> hash = new Hashtable<>();
hash.put("Key1", "Custom value 1");
```

```
hash.put("Key2", "Custom value 2");
hash.put("Key3", "Custom value 3");
SMEvent event = new SMEvent(hash,
      new SMCallback()
      {
         @Override
         public void onSuccess(String result)
         {
            [Do something here]
         }

         @Override
         public void onError(int responseCode, Exception exception)
         {
            [Do something here]
         }
      });
SMManager.getInstance().sendSMEvent(event);
```

6.6 Geolocation

Geolocation is managed through a 3rd party library by PlotProjects, so you need to add a dependency to it (cf. Other libraries).

To configure that library, you must add a plotconfig.json file in the asset folder of your app. There, you can set a few properties but only one is mandatory: the Plot public token. For more information, check PlotProjects documentation.

Example:

```
{
  "publicToken": "YOUR_PUBLIC_TOKEN",
  "debug": true
}
```

To tell the SDK to use the Geolocation, set the property ConfigureGeolocation to true on the SMSettings object (cf. Optional settings).

If you set "enableOnFirstRun" to false in the plotconfig.json file, geolocation will be initialized but NOT enabled. So the permission will not be asked to the user and you will have to enable it manually.

To do so, you can call the method SMManager.getInstance().enableGeolocation(). Note that this will enable the geolocation but not ask the permission, you must do it yourself in that case.

This method, along with SMManager.getInstance().disableGeolocation() and SMManager.getInstance().isGeolocationEnabled() can also be used to provide the user with an opt-in/opt-out functionality.

NOTE:

- Don't initialize or call any method of the PlotProjects API in your app, everything is managed by our SDK.
- Our SDK sets the icon used by plot for the geolocation notifications, reusing the one that you already gave us, so no need to specify it a second time in the config file.
- Default value for "enableOnFirstRun" is true, so if you specify only the token in the config file, you don't have to call any method.

6.7 Broadcasts -- deprecated

A certain number of broadcasts are sent from the SDK at different moments. You can listen to them to be able to execute some code related to those events.

Refer to the template project for examples of what to do with the data sent with those broadcasts.

6.7.1 *Generic broadcasts*

Technical Note: Due to limitations to what can be done in background starting with Android O, this broadcast is now deprecated with SDK 1.6.0. It is still sent but you won't be able to listen to it if your app targets android O (targetSdk=26) and runs on an Android O device.

This does not affect the local broadcasts which continue to work normally.

These broadcasts are sent using Context.sendBroadcast(Intent). In order to listen to them, you have to register a BroadcastReceiver, either in your AndroidManifest.xml file or dynamically in your activity. Each of these require a category filter with the package name of your app.

BROADCAST_EVENT_RECEIVED_REMOTE_NOTIFICATION = "SMReceivedRemoteNotification"

Example:

(considering your package name is com.mycompany.myapp):

```
<receiver android:name=".EventReceiver">
   <intent-filter>
      <action android:name="SMReceivedRemoteNotification"/>

      <category android:name="com.mycompany.myapp"/>
   </intent-filter>
</receiver>
```

And in your class EventReceiver:

```
public class EventReceiver extends BroadcastReceiver
{
   public void onReceive(Context context, Intent intent)
   {
      String action = intent.getAction();

      switch (action)
      {
         case SMManager.BROADCAST_EVENT_RECEIVED_REMOTE_NOTIFICATION:
            String id = intent.getStringExtra("id");
             String title = intent.getStringExtra("title");
             break;
      }
   }
}
```

If you want to register your receiver dynamically:

```
EventReceiver receiver;

...
@Override
protected void onStart()
{
   super.onStart();

   if (receiver == null)
   {
      receiver = new EventReceiver();
   }
   IntentFilter filter = new IntentFilter();
   filter.addAction(SMManager.BROADCAST_EVENT_RECEIVED_REMOTE_NOTIFICATION);
   registerReceiver(receiver, filter);
}
```

6.7.2 Local broadcasts

Technical Note: Due to LocalBroadcastManager being deprecated with AndroidX, all the local broadcasts, although still sent, are now also deprecated starting at SDK 3.0.0. Use the observers instead.

These broadcast are sent using LocalBroadcastManager, they are local to the app. In order to listen to them, you have to register a BroadcastReceiver with LocalBroadcastManager dynamically in your activity. As they are local to your app, they do not require a category filter.

BROADCAST_EVENT_RECEIVED_IN_APP_MESSAGE = "SMReceivedInAppMessage"

BROADCAST_EVENT_BUTTON_CLICKED = "SMEventButtonClicked"

BROADCAST_EVENT_WILL_DISPLAY_NOTIFICATION = "SMEventWillDisplayNotification"

BROADCAST_EVENT_WILL_DISMISS_NOTIFICATION = "SMEventWillDismissNotification"

BROADCAST_EVENT_RECEIVED_GCM_TOKEN = "SMReceivedGCMToken";

Example: *consider you have a class EventReceiver:*

```
public class EventReceiver extends BroadcastReceiver
{
   String action = intent.getAction();

   switch (action)
   {
      case SMManager.BROADCAST_EVENT_RECEIVED_IN_APP_MESSAGE:
         SMInAppMessage[] messages =
(SMInAppMessage[])intent.getSerializableExtra(SMManager.BROADCAST_DATA_IN_APP_MESSAGES);
         //Do some stuff
         break;


      case SMManager.BROADCAST_EVENT_BUTTON_CLICKED:
         SMNotificationButton button =
(SMNotificationButton)intent.getSerializableExtra(SMManager.BROADCAST_DATA_BUTTON);
         //Do some stuff
         break;



      case SMManager.BROADCAST_EVENT_WILL_DISPLAY_NOTIFICATION:
         //Do some stuff
         break;

      case SMManager.BROADCAST_EVENT_WILL_DISMISS_NOTIFICATION:
         //Do some stuff
         break;

      case SMManager.BROADCAST_EVENT_RECEIVED_GCM_TOKEN:
         String gcmToken = intent.getStringExtra(SMManager.BROADCAST_DATA_GCM_TOKEN);
         //Do some stuff
         break;
   }
 }
```

And in your activities:

```
EventReceiver localReceiver;
```

```
...
@Override
protected void onStart()
{
   super.onStart();

   if (localReceiver == null)
   {
      localReceiver = new EventReceiver();
   }
   IntentFilter filter = new IntentFilter();
   filter.addAction(SMManager.BROADCAST_EVENT_BUTTON_CLICKED);
   filter.addAction(SMManager.BROADCAST_EVENT_WILL_DISMISS_NOTIFICATION);
   filter.addAction(SMManager.BROADCAST_EVENT_WILL_DISPLAY_NOTIFICATION);
   filter.addAction(SMManager.BROADCAST_EVENT_RECEIVED_IN_APP_MESSAGE);
   LocalBroadcastManager.getInstance(this).registerReceiver(localReceiver, filter);
}
```

6.8 Observers

Starting with version 3.0.0, the observers are the new way to listen to SDK events.

Depending which event you want to observe, you will create a different Observer and pass it the corresponding method. On the onComplete event of the observer, you will be given the value you are listening to (token, In-App messages, etc. or nothing if it is just to notify you that something happened). The events are sent only once to each observer (if you have several observers for the same event, each will be triggered once).

By default, the onChanged event of the observer is triggered when there is a new value after the observer is created. If there is a new value before the observer is created, the event will not be triggered. If you want to change this behaviour, use the overload that requires a Boolean as third parameter and set it to true (default value is false).

Here are the two behaviours in more details:

- If you use one of the observe methods (cf. list below) with two parameters, or set the third one to false, the event will be triggered for all the observers already created when the new value is sent to them. Any observer (re)created after that will not see its onChanged event triggered.

  In this case, you know that, when an event is triggered, it means there is a new value but you won't get it if it was sent before the creation of the observer.

  Example:

  - You have an observer for In-App messages in an Activity. The In-App messages are retrieved before arriving in that activity. The Activity and the observer are created, nothing happens.

- You have an observer for In-App messages in an Activity. The In-App messages are retrieved after arriving in that activity. The onChanged event is triggered. You navigate to another Activity, come back to that one, the event is not triggered.

- If you use one of the observe methods (cf. list below) with three parameters and set the third one to true, the event will be triggered for all the observers. If any observer is (re)created after that, its onChanged event will still get triggered.

  In this case, the event is triggered as soon as there is a value, whether it is new or not. It is up to you to know if you already treated that value. Note: this is the normal behaviour of the Android LiveData functionality.

  Example:

  - You have an observer for In-App messages in an Activity. The In-App messages are retrieved before arriving in that activity. The Activity and the observer are created, the event is triggered.
  - You have an observer for In-App messages in an Activity. The In-App messages are retrieved after arriving in that activity. The onChanged event is triggered. You navigate to another Activity, come back to that one, the event is triggered again. It will be triggered every time the observer is recreated.

Here is the list complete list of all the events you can observe:

- Clicked button

  When a button is clicked, the corresponding SMNotificationButton object is sent to the app to allow it to react to it. Observe it using:

```
SMManager.getInstance().getObserverManager().observeClickedButton(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<SMNotificationButton> observer)
```

- Will display message

  This is triggered when a message is about to be displayed. Observe it using:

```
SMManager.getInstance().getObserverManager().observeDisplayedMessage(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<Void> observer)
```

- Will dismiss message

  This is triggered when a message is about to be dismissed. Observe it using:

```
SMManager.getInstance().getObserverManager().observeDismissedMessage(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<Void> observer)
```

- FCM token

  When the token is received, it is sent throughout the app (cf. Retrieving the Firebase Cloud Messaging (FCM) token). Observe it using:

```
SMManager.getInstance().getObserverManager().observeToken(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<String> observer)
```

- Received In-App messages

  When the SDK fetches the In-App messages, it sends an array of the Ids and labels to the app. Observe it using:

```
SMManager.getInstance().getObserverManager().observeInAppMessages(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<SMInAppMessage[]> observer)
```

- Custom event

  A custom event can be sent using the value associated to a button or a main action. Observe it using:

```
SMManager.getInstance().getObserverManager().observeEvent(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<String> observer)
```

- Device id

  When the device id is received, it is sent throughout the app. Observe it using:

```
SMManager.getInstance().getObserverManager().observeDeviceId(@NonNull LifecycleOwner lifecycleOwner, @NonNull Observer<String> observer)
```

Example:

```
final Observer<HashMap<String, Integer>> inAppMessageObserver = new Observer<SMInAppMessage[]>()
{
   @Override
   public void onChanged(SMInAppMessage[] inAppMessages)
   {
      Toast.makeText(MainActivity.this, "Received " + inAppMessages.length + " InApp messages (liveData)", Toast.LENGTH_SHORT).show();
   }
};
```

```
SMManager.getInstance().getObserverManager().observeInAppMessages(this,
inAppMessageObserver);
```

6.9 Translations

When asking for a permission, a text is displayed explaining why we need it. By default, it is in English but a translation for a few languages is provided: Dutch, Spanish, French and German. If you want to add another language (or change the message), add under “res” a folder named “value-[language code]” (example: to add Russian, you would name it “value-ru”), create a “strings.xml” file and, under a <resources> tag, add:

```
<string name="sm_permission_explanation_location">Write here the message you want</string>
<string name="sm_permission_explanation_write_external_storage">Write here the message you want</string>

<string name="sm_permissions_not_enough">Write here the message you want</string>
```

7 Proguard

If you are using Proguard to minify the code of your app, add the following lines to the file proguard-rules.pro:

```
-dontwarn com.selligent.sdk.**
-keep class com.selligent.sdk.* {
   public private *;
}

-keep class com.google.firebase.messaging.FirebaseMessaging { *; }
```

8 Changelog

Version 3.8.1

- Fixed a bug making the Huawei dependencies mandatory when they should be optional

Version 3.8.0

- Added support for Huawei Mobile Services (HMS). Now, when our SDK is in an app running on a device that doesn't have Google Play Services but has Huawei Services, it will use those to retrieve a token and listen to the push notifications. This means the SDK will work in an Android app running on a Huawei device without Google Play Services (Android or HarmonyOS). It will not, however, work in a native HarmonyOS app.

  NOTE: Sending push through Huawei Services is only supported by the Selligent Mobile Platform for customers using SDC. Ask your Selligent contact for more information about SDC.

Version 3.7.0

- Added support to Android 12 (API 31)
- Added a check when starting the SDK to verify if the notifications were enabled/disabled in the OS settings and send the information to the Selligent Mobile platform if needed.
- Enabled DOM storage for the Web views
- Exposed the push notification payload and added helper methods to manage the events (cf. Manual management of the push).

Version 3.6.0

- Added an overload of checkAndDisplayMessage to allow you to display the In-App message linked to a push yourself.
- Added AddInAppMessageFromPushToInAppMessageList to SMSettings to allow the In-App message linked to a push to be stored with the other In-App messages.
- Added WebViewNavigationOverride to allow you to intercept the links clicked in a WebView of an In-App Message and decide if the navigation will proceed or not.
- Added method and observer to retrieve the device id.
- Corrected a bug preventing onNewIntent to be called.
- Different code clean-up/refactoring.

Version 3.5.1

- Corrected bug where SMMessageType was not public

Version 3.5.0

- Small modification to display the image from a rich push as a thumbnail when the notification is collapsed.
- Added a method to delete the In-App messages and another to mark them as unread.
- Some bug corrections
- Clean-up of obsolete code.

Version 3.4.0

- Added properties on SMInAppMessage and methods on SMManager to give access to the full payload of the In-App messages, allowing the possibility to display the In-App messages without the SDK.
- Added a property on SMSettings to prevent the SDK from automatically enabling the notifications once the token is retrieved.

Version 3.3.0

- Added Android 11 support (API 30)
- Corrected a problem when an In-App message was displayed from an activity having its launchMode set to “singleInstance”

Version 3.2.0

- Added support for rich push (image inside a notification). Nothing to implement, everything is managed internally by the SDK
- Added the possibility to set the color of a notification icon (method setNotificationIconColor on SMManager)
- Added an overload to SMUserLogin, SMUserLogout, SMUserRegister, SMUserUnregister constructors that doesn’t require the email.
- Added overloads to the Observe methods of SMObserverManager to give the possibility to change the behaviour regarding when the onChanged events are triggered on the observers.

Version 3.1.0

- FirebaseJobDispatcher, which was deprecated by Google and will not be supported anymore starting April 2020, was replaced by WorkManager. The only impact is the change in dependencies in the Gradle file (cf. Other libraries)
- The Selligent SDK was tested with the latest version of the PlotProject library (3.10.0 at the time of writing) which can be safely used.

Version 3.0.0

- Adapted the SDK to be compatible with Firebase-Messaging 19 (and above), and Google-play-services 17 (and above) by replacing the "support" libraries by the corresponding AndroidX ones. This means your app must use AndroidX to use this version of the SDK. It is not compatible anymore with the support libraries and, therefore with the Firebase-Messaging 18 and below, and Google-Play-Services 16 and below.
- Deprecated all the broadcasts (they are still sent, though)
- Added the management of observers to observe events. They replace the deprecated broadcasts.
- Added Android 10 support (API 29).
- To migrate to this version: if it is not already done, use Android Studio to migrate to AndroidX (Refactor -> Migrate to AndroidX) then replace your BroadcastReceiver that listened to our broadcasts by different observers depending on what you want to listen to as describe in the documentation above.

Version 2.3.0

- Added Android 10 support (API 29).
- **This version of the SDK is NOT compatible with Firebase-Messaging 19 (and above) and Google-Play-Services 17 (and above)**
- This version was made compatible with Android 10 (API 29) but it still uses the support libraries which will not be updated to 29 as Google pushes to migrate to AndroidX. So, if you want to update your app to target API 29, we recommend migrating to AndroidX and use version 3.0.0 of our SDK.

Version 2.2.0

- Updated minSdk to 16 to ensure compatibility with Firebase-Messaging 18
- Added another service to retrieve the token extending FirebaseMessagingService
- Improved way to manage the token after retrieval
- Moved the retrieval of the user agent to the background when possible (still done in main thread for old Android versions)

- Fixed Activity leak
- This version of the SDK is NOT compatible with Firebase-Messaging 19 and Google-Play-Services 17

Version 2.1.2

- Fixed a ConcurrentModificationException appearing sometimes.
- Fixed a bug when event callbacks are called out of context.
- Fixed a bug where calling displayLastReceivedRemotePushNotification displayed an encrypted message.
- Reworked the Reload method.
- Updated an error log to facilitate the resolution of the problem.

Version 2.1.1

- Fixed a bug preventing the retry of events when opening the app if the cache is loaded synchronously.
- Fixed a bug sending twice the PushReceived event when the push is received while the app is closed and the cache is set to load asynchronously.
- Fixed a bug in SMBaseActivity regarding when some operations are executed.

Version 2.1.0

- Added the setting LoadCacheAsynchronously to read the cache in a separate thread. All writing is now always done in a separate thread.
- Added the setting DoNotFetchTheToken to prevent the SDK from fetching the Firebase token itself. If set to true, it then becomes the responsibility of the app to do it and give it to the SDK using SMManager.getInstance().setFirebaseToken(String token)
- Added the setting DoNotListenToThePush to prevent the SDK from listening to the push itself. If set to true, it then becomes the responsibility of the app to do it and give it to the SDK using SMManager.getInstance().displayNotification(Context context, Intent intent)

Version 2.0.2

- Bug fix for Android versions <= KitKat (4.4)

Version 2.0.1

- Bug fix when sending the token to the platform

Version 2.0.0

- Added decryption of push messages
- Added Android P (API 28) support
- Added new way to retrieve FCM token (done automatically by the SDK, no extra call to our API needed)
- Improved communication security
- Deprecated the registration to FCM through the registerDevice method. Use the JSON file instead.

Version 1.9

- Added management of buttons inside the notification.

Version 1.8

- Added management of push without In-App messages and with an action triggered when clicking on the notification (like a deep link).

Version 1.7.2

- Corrected a bug preventing the push to be correctly handled when received while on a webview opened by a previous notification.

Version 1.7.1

- Corrected a bug preventing the image to be correctly displayed in an InApp-Content fragment when the image was larger than the screen.

Version 1.7.0

- Added geolocation functionality
- Android 8.1 compatible

Version 1.6.1

- Bug correction

Version 1.6.0

- Adaptations for Android O.
- Removal of the service com.selligent.sdk.GcmIntentService. If you reference it in your AndroidManifest.xml file, remove that entry.
- The broadcast *BROADCAST_EVENT_RECEIVED_REMOTE_NOTIFICATION* is now deprecated due to limitations with Android O. It is still sent but will not be received by an app targeting Android O and running on an Android O device.

Version 1.5.0

- Image type In App contents can now be marked at creation as downloadable. In that case, after retrieving the In App contents, the SDK will (asynchronously) download the image for each content marked as such and store it on the device.
- Due to changes in the Android SDK, the inclusion of the third party library com.drewnoakes:metadata-extractor is now required when using "Response" type buttons within a push or In-App messages.
- "Dangerous" permissions (those requiring to be explicitly granted by the user under Android 6.0 and above) are not included in the AndroidManifest.xml by the SDK anymore, it will have to be done in the app manifest. This will allow to restrict the permissions to the functionality actually used by the app. See each functionality to know which permission they require.
- Added support for Android SDK 24 and 25.
- Added management of "Passbook" type buttons with push, In-App message. Clicking on such a button will open a passbook app if the device has one or the browser instead.

Version 1.4.3

- sendSMEvent, when used with a custom event, will only send it if the data passed is new. If all entries in the hashtable have the same values as the last time it was sent, then we won't do anything. If you want to log when a specific action happened and the values do not change, add a date in the data.

Version 1.4.2

- Selligent SDK now available from JCenter

Version 1.4

- SMSettings.Theme is now deprecated. This value is not used anymore as the layout of the dialog in now completely customizable (cf. Dialog).

- The design of a dialog does not try to adapt to the theme and the version of Android anymore, instead there is a default material layout that is entirely customizable (cf. Dialog).
- The SetInfo event is now sent only when some of its info changed, not systematically at each start of the SDK anymore.
- Added support for Android SDK 23 and the new way permissions are managed.
- Added In-App contents management.

9 Use cases

9.1 I have a simple app with only one Activity

This is the simplest case. Start the SDK in your Application and set NOTIFICATION_ACTIVITY to your only activity. In that Activity, call the method checkAndDisplayMessage on the onStart and onNewIntent events, register our SMForegroundBroadcastReceiver on the onStart and unregister on the onStop.

9.2 My app has several Activities, I want the In-App message linked to the push to be displayed or the main action of the notification (like a deep link) to be executed no matter the Activity displayed

Start the SDK in your Application and set NOTIFICATION_ACTIVITY to your main activity.

It is recommended to use a base Activity that the others will extend, otherwise you will have to do the following in each one of them:

Like for the simple case, call the method checkAndDisplayMessage on the onStart and onNewIntent events, register our SMForegroundBroadcastReceiver on the onStart and unregister on the onStop.

But this time, in the onStart, also set NOTIFICATION_ACTIVITY to the current one. That way, when a push is received while you are in the app, the current activity will be called instead of your main one.

If you do that in a base activity, then set it like this:

```
SMManager.NOTIFICATION_ACTIVITY = this.getClass();
```

9.3 My app has a specific activity as splash screen and when I receive a push while the app is closed, after clicking on the notification, I want the app to open, go through the splash screen and once in the main one, have the In-App message displayed or the main action (like a deep link) executed

Start the SDK in your Application and set NOTIFICATION_ACTIVITY to your splash activity.

Do everything else like described above except for your splash Activity. There, don't call any of the SDK method but, when starting your main activity, transfer the push information to it like this:

```
Intent intent = new Intent(SplashActivity.this, MainActivity.class);
Bundle extras = getIntent().getExtras();
if (extras != null)
{
   intent.putExtras(extras);
}
startActivity(intent);
```

Also, update the onNewIntent event like this:

```
@Override protected void onNewIntent(Intent intent)
{
   super.onNewIntent(intent);

   Bundle extras = intent.getExtras();

   if (extras != null)
   {
      this.getIntent().putExtras(extras);
   }
}
```

It will allow a push received while on the splashscreen to be transferred to the main activity.

NOTE: this is not necessary with the new splash screen introduced in Android 12, simply set NOTIFICATION_ACTIVITY to your main activity.

10 Troubleshooting/FAQ

- **Q: When I look at the devices in Campaign, why don't they have a token?**

A: If there is no token, that means the SDK did not send it. There are a few reasons why this could happen:

- If you don't use the google-services.json file (cf. Creating a Google application), then you have to call the method registerDevice(Context context) on SMManager in the onStart method of your base Activity. Also, check if the senderId is correctly passed to the SDK (SMSettings.GoogleApplicationId). Note: this is not recommended anymore as that way of retrieving the token is deprecated.
- If you use the google-services.json file, make sure that it is correctly placed in your app and that you updated your build.gradle files as described on the Firebase website when adding cloud messaging to your app.

- If you set DoNotFetchTheToken to true at start, the SDK won't retrieve the token itself, you must do it yourself and then give it to it. Once you have the token, you must call the method `SMManager.getInstance().setFirebaseToken(String token)`

- **Q: Why don't I receive the push on my device?**

A: You must check a few things. First, look at the push status in Campaign. Here are some errors you might encounter:

- **Device subscription expired**. This means the token used to contact the device is not valid anymore. It may be because the user hasn't used the app for a while and the token expired or the user uninstalled the app. You can't do anything here except wait for a new one to be received.
- **Mismatch sender id**. This means the token used to contact the device was created with a sender id that does not correspond to the server key stored on the Selligent platform. Check both values on Firebase and then change the sender id given to the SDK and/or send us the correct server key.
- **Authentication failed**. The server key that you gave us is not correct. Check it on Firebase and send us the new one.
- If the push optout is 1, then the push will not be sent to the device. Maybe it was not able to get a token or there was a call to the method disableNotifications().

  If the status in Campaign is ok, then there might be something wrong in your app.
- If you don't use the google-services.json file, check that you added the correct permissions in your manifest (cf. Permissions for Push notifications)
- Do you have another BroadcastReceiver listening to GCM/FCM push? If yes, then it might be trying to interpret our JSON payload and crash before our receiver has time to finish its work. You can recognize a Selligent push at its "sm" property at the root of the JSON.
- When starting the SDK, did you set the property RemoteMessageDisplayType to None? This will prevent it from creating a notification and displaying the message when receiving a push while the app is in foreground.
- Did you call the method disableNotifications()? This will set the push optout to 1 in Campaign, preventing it to send push to that device.
- If you set DoNotListenToThePush to true, the SDK won't listen to the push, you must do it yourself. Then, when you receive a push, you must call the method `SMManager.getInstance().displayNotification(Context context, Intent intent)`

- **Q: My app crashes when opening, with the following trace**

```
at com.google.android.gms.iid.zzd.zzdo(Unknown Source)

at com.google.android.gms.iid.zzd.<init>(Unknown Source)

at com.google.android.gms.iid.zzd.<init>(Unknown Source)
```

```
at com.google.android.gms.iid.InstanceID.zza(Unknown Source)

at com.google.android.gms.iid.InstanceID.getInstance(Unknown Source)

at com.selligent.sdk.SMRegistrationIntentService.getInstanceID(SMRegistrationIntentService.java:19).
```

A: This can happen if the targetSdk of your app is 23 or above and your version of Google Play Services is 8.+. This is a known bug in Google Play Services that was corrected in later versions. Try using 9.2 or above.

- **Q: I see "The SDK did not start correctly" in the logs, what should I do?**

A: Check your code starting the SDK. Did you correctly set the 3 following values: webServiceUrl, clientId and privateKey? If any is missing, that message will appear.

- **Q: I try to send an event but nothing happens, the callback methods are not called**

A: The SDK did not start correctly (cf. previous question). You can see in the logs a message telling you that by setting

SMManager.DEBUG = true;

before calling the start method. If you did but still don't see anything, then it means your code starting the SDK is not called. In this case, you need to check your AndroidManifest.xml file and make sure that your class is correctly referenced in it. If your class is called "MyCustomApplication" and you package is "com.mycompany.myapp", then you must have in the manifest:

<application android:name="com.mycompany.myapp.MyCustomApplication"

Note that the package name is not mandatory, you can simply specify the name of the class preceded by a ".".

- **Q: I receive the push and the In-App message is displayed when clicking on the notification, but the layout is weird and the buttons are missing**

A: Your app is probably obfuscated/minified using Proguard. You need to add some rules in the file Proguar-rules.pro (cf. Proguard)

- **Q: I'm sending a push with a deep link, the activity is configured in AndroidManifest.xml, the method checkAndDisplayMessage is called on the onStart and onNewIntent events in the activities but when I tap on the notification, although the app opens, the activity corresponding to that deep link is not displayed**

A: Check the url defined and the app and the one set in the push, if there is any difference (even the slightest, like a part in uppercase in one and lowercase in the other), the deep link will not be triggered.