Android – Using the SDK

selligent

Analysis

Selligent

1 Table of content

2 Intro

The purpose of this document is to detail how to incorporate the SDK to an app.

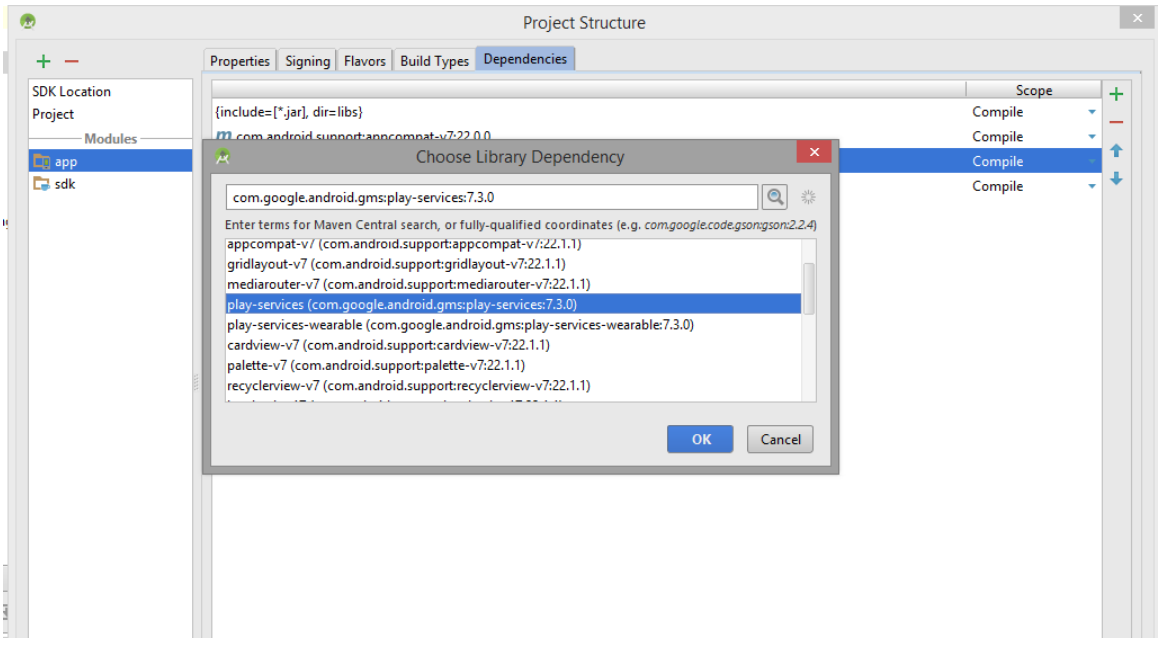
3 Creating a Google application

- Go to https://console.developers.google.com/project and sign in with a Google account.
- Click on “Create a project” and follow instructions.
- Note the project id, it will be used later
- Click on the project, under “Mobile APIs” click on “Google Cloud Messaging”
- Enable
- Click on “Credentials”, “Create credentials”, “API key” and create a new server key. Note that key, it will be used later.

4 Mandatory libraries

In Android Studio, a dependence to the Google Play Services and Google GSON must be added.

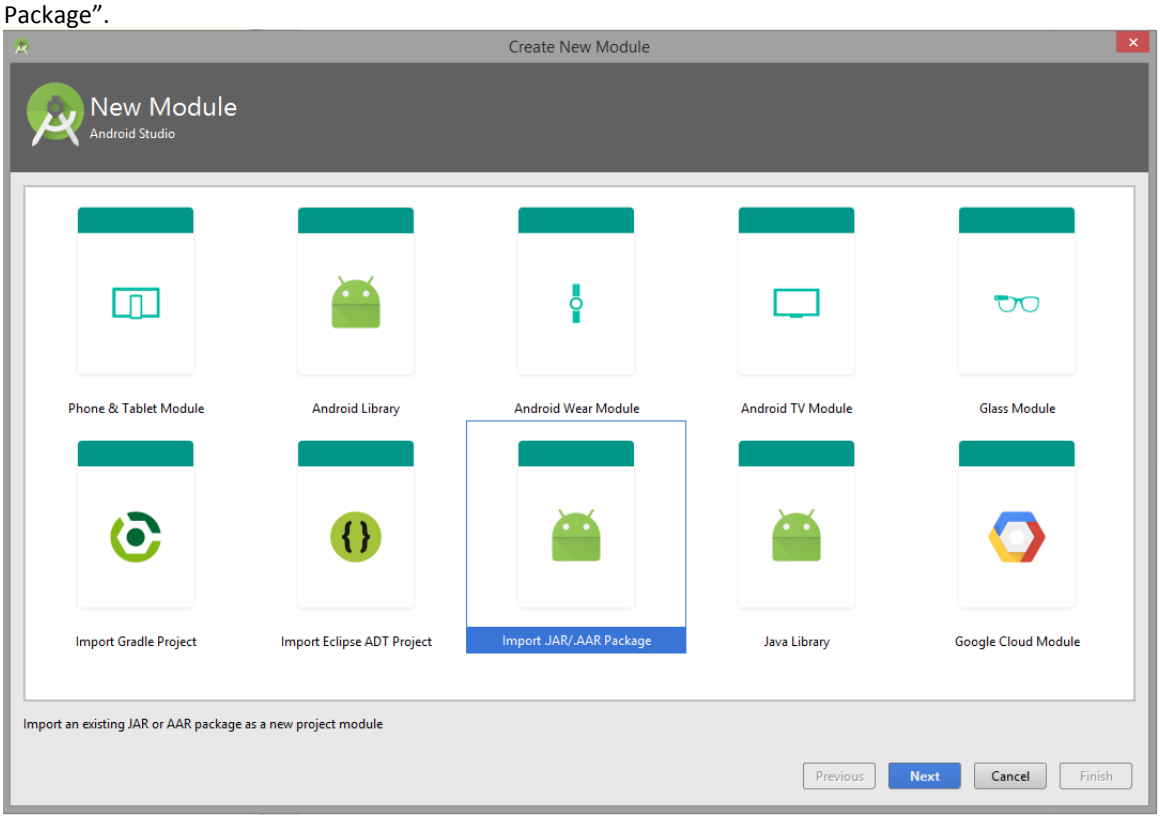
Right click on then the app module, “Open module settings”, “Dependencies”, click on “+”, “library dependency” and select “play-services”



Do the same for GSON

5 Including the SDK in your project

Now, the SDK can be added to the project. To do this, add a new module and choose “Import .JAR or .AAR Package”.



And select the file.
Once it is done, add a dependency to this new module in your app, then synchronize and build the project.

There is a bug in Android Studio versions prior to 1.3 that will make it look like the library is not recognized. It can be used, everything will compile but there will be no intellisense and everything from the library will be marked in red as if there was an error. To prevent that, follow these steps:

- In the build.gradle file of the app module (not the main one of the project), the line referencing the library must be replaced by

  ```
  compile(name: 'sellsdk-release', ext: 'aar')
  ```

  “sellsdk-release” is the name of the file

- Still in build.gradle, add:

  ```
  repositories
  {
      flatDir
      {
          dirs 'libs'
      }
  }
  ```

- Copy the library in the folder “libs” of the “app” module.
- Synchronise, everything should be fine now.

6 Starting the SDK

6.1 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. 6.2).
In the AndroidManifest.xml file, add the following:

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

6.2 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 web service that will be called.
GoogleApplicationId must be the id of the Google project created at point 3.
ClientId is the public key allowing the connection to the web service.
PrivateKey is the private key allowing the connection to the web service.

Ex.:

```
SMSettings settings = new SMSettings();
settings.WebServiceUrl = "https://www.some.web.service.com";
settings.GoogleApplicationId = "1111111111";
settings.ClientId = "SomeClientId";
settings.PrivateKey = "SomePrivateKey";
SMManager.getInstance().start(settings);
```

6.2.1 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.

`public SMInAppRefreshType InAppContentRefreshType`
Setting this value will enable the In App contents.

`public SMRemoteMessageDisplayType RemoteMessageDisplayType`
Setting this value will enable/disable the automatic display of remote messages as they are received.

- 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()).

There are also some optional settings on SMManager:

```
SMManager.DEBUG = true;
```
Setting this to true will add the SDK logs to the logcat.

7 Push notifications

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

7.1 Permissions for Push notification

In order to be able to receive push notifications, the following permissions must be granted in the AndroidManifest.xml file:

```
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-permission android:name="com.mycompany.myapp.permission.C2D_MESSAGE" />

<permission
    android:name="com.mycompany.myapp.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
```

com.mycompany.myapp must, of course, be replaced by your package.

7.2 Registering

To register your device to Google Cloud Messaging (and therefore allow it to receive push notifications), add the following call to the onStart event of your main activity or your base activity if you have one:

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

   […]

   SMManager.getInstance().registerDevice(this);
}
```
This registration is asynchronous and is executed only once after the application started (it will be executed again the next time you start the app after killing it).

7.3 Listening and displaying the push notifications

In order to check if a notification was received and to display it, you have to 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.

```
SMForegroundGcmBroadcastReceiver receiver;
```

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

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    […]

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

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

    […]

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

    SMManager.getInstance().registerDevice(this);
}

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

    […]
    unregisterReceiver(receiver);
}

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

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

7.3.1 Extending SMBaseActivity

There is a class SMBaseActivity in the SDK that already does everything described in 7.2 Registering and 7.3 Listening and displaying the push notifications. You can make your activities extend it to avoid writing the code described in those points. Be aware though that SMBaseActivity extends AppCompatActivity, so it might not work for your project.

```
public class MainActivity extends SMBaseActivity
{
}
```

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

7.4 Customization

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

7.4.1 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.

7.4.2 Setting a specific Activity

By default, the Activity called to display the message of a push is NotificationActivity. If you want to keep it, in order 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:

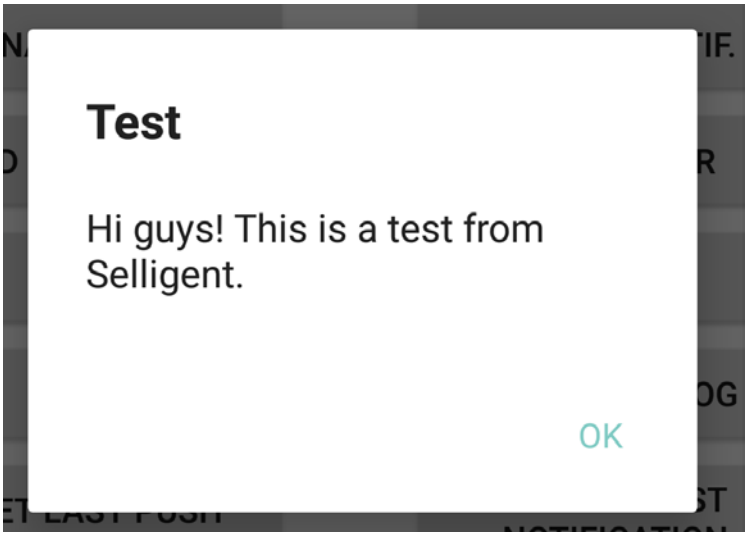
```
SMManager.NOTIFICATION_ACTIVITY = MyActivity.class;
```

If you set RemoteMessageDisplayType to SMRemoteMessageDisplayType.Notification (cf. Optional settings), you can also set this property in your base activity by giving it 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.

7.5 Design customization

7.5.1 Dialog

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



PS: 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>
```

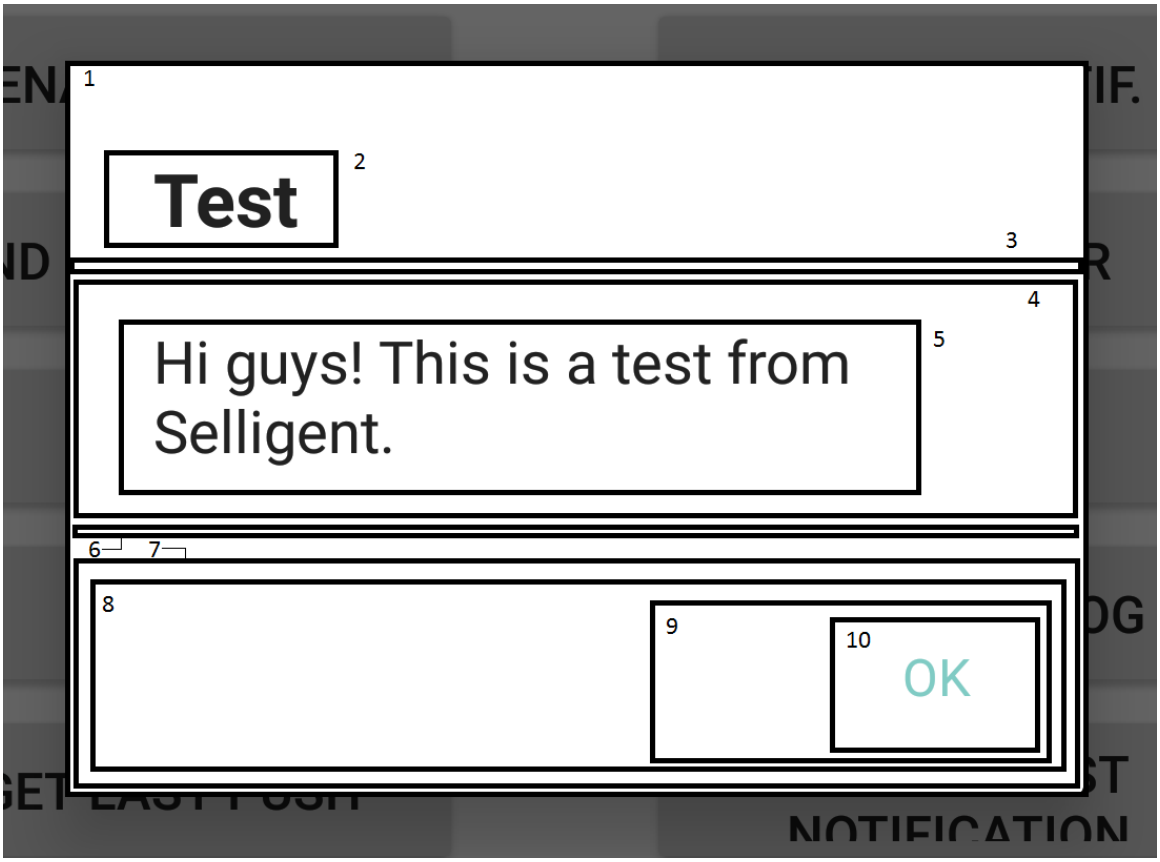
To customize it, in your own file “styles.xml”, override the styles you want to modify. Note that you have to copy the whole content of those styles, not only the items you want to change, otherwise the others will be lost.

For 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">
```



7.5.2 Activities

Some other type of messages, like Map, Image or HTML, are displayed in their own activity, not in a dialog. Those activities extend AppCompatActivity and, therefore, need an AppCompat theme. If you use one, then you don't have anything to do. However, if you don't, our own SMTheme will be forced onto those activities. In this case, you might want to override it to reflect your own layout (by default, it uses Theme.AppCompat.Light.DarkActionBar as parent).
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)

7.6 Retrieving the GCM token

There are two ways to retrieve the GCM token: listening to a broadcast and a call to a method.

7.6.1 Broadcast

You can listen to SMManager.BROADCAST_EVENT_RECEIVED_GCM_TOKEN (its value is "SMReceivedGCMToken"). This broadcast is sent after reception of the token from GCM 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").

For 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;
        }
    }
}
```

7.6.2 SMManager.getInstance().getGCMToken

This method will return the token stored by the SDK.

Note that, as the processing to register to GCM is asynchronous, it is possible that the value returned is either empty or not up-to-date if the registration is not finished yet when the call is made.

7.7 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();
```

7.8 Setup for special push

7.8.1 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/business/mobile/android/auth

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"/>
```

7.8.2 Event

When displayed, the message of a push can contain buttons. One type of button can send a broadcast containing a specific value, in order for you to execute some code when you receive it. This broadcast is sent locally using LocalBroadcastManager. In order 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.

For example, if the name of the event 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":
                //Do some stuff
                 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);
}
```

7.9 Broadcasts

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.

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. 7.5.1)

For more info regarding the broadcasts, go to Broadcasts

7.10 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);
SMManager.getInstance().getLastRemotePushNotification();
```

8 In App messages

8.1 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.

To disable them, 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);
```

8.2 Reception of the messages

The In App Messages are sent to you using the broadcast ***BROADCAST_EVENT_RECEIVED_IN_APP_MESSAGE***.
Use `SMManager.BROADCAST_DATA_IN_APP_MESSAGES` to retrieve them from the intent.

8.3 Display of a 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 and `activity` the Activity that will display it.
It will be displayed the way the push notifications are.

8.4 Broadcasts

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

9 In App contents

9.1 Libraries

In order for the project to build, you must have a dependency to the following library:

- com.android.support:cardview-v7

9.2 Enabling the In App contents

In app contents are disabled by default, unless you set InAppContentRefreshType on SMSettings.
It is highly recommended to avoid setting the value to Minutely for production. It is there for testing purpose only.
New content is retrieved when the app becomes active (ie. at start, when going from background to foreground, when the orientation changes) ONLY if the last refresh is older than the value set for InAppContentRefreshType.

9.3 Implementing the In App content using our Fragments

There are 3 types of In App contents: Image, URL and HTML. Each has its own Fragment, respectively: SMInAppContentImageFragment, SMInAppContentUrlFragment and SMInAppContentHtmlFragment.

Each one can either be displayed as a full screen dialog using the “show” method (they all extend DialogFragment) or be used as a standard Fragment (cf. official Android documentation on Fragments).

Note that you cannot reference them directly in a layout using a “fragment” tag because they need arguments. You have to instantiate them using the static method “newInstance” and add them programmatically to the layout.

Once instantiated, you can call the method hasContent() to know if there is any content available for the given category, so you can display something else instead of the fragment.

Example:

Considering the following layout:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    […]

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:id="@+id/urlFragmentContainer"/>

    </LinearLayout>
```

You will have this kind of code:

```
SMInAppContentUrlFragment urlFragment = SMInAppContentUrlFragment.newInstance("testUrl");

if (urlFragment.hasContent())
{
    FragmentManager fragmentManager = getFragmentManager();
    FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.replace(R.id.urlFragmentContainer, urlFragment, "URL");
    fragmentTransaction.commit();
}
else
{
     //do some other stuff
}
```

In App contents are sorted by categories (it is one of their property at creation). Therefore, all our Fragments require to be given that category at instantiation. Each one will only display the In App content corresponding to it.

SMInAppContentImageFragment and SMInAppContentUrlFragment both display only one content at a time. It is not the case for the HTML type, so SMInAppContentHtmlFragment needs a second argument which is the number of In App contents to display. Setting it to -1 means that all available contents will be used.

9.3.1 Customization

The layout of the In App contents is entirely customizable. We have a default one that is defined in the file “styles.xml”. Here is its content:

```
<style name="Selligent.InAppContents.Image">
    <item name="android:background">#FFFFFF</item>
</style>
<style name="Selligent.InAppContents.Html.Container">
    <item name="android:background">#EEEEEE</item>
    <item name="android:layout_margin">0dp</item>
</style>
<style name="Selligent.InAppContents.Html.Card">
    <item name="cardBackgroundColor">#FFFFFF</item>
    <item name="cardCornerRadius">2dp</item>
    <item name="cardElevation">2dp</item>
    <item name="cardMaxElevation">2dp</item>
    <item name="cardPreventCornerOverlap">true</item>
    <item name="cardUseCompatPadding">true</item>
    <item name="contentPadding">5dp</item>
    <item name="contentPaddingBottom">5dp</item>
    <item name="contentPaddingLeft">5dp</item>
    <item name="contentPaddingRight">5dp</item>
    <item name="contentPaddingTop">5dp</item>
</style>
<style name="Selligent.InAppContents.Html.CardContainer"/>
<style name="Selligent.InAppContents.Html.CardTitle">
    <item name="android:textSize">24sp</item>
    <item name="android:textColor">#000000</item>
    <item name="android:paddingLeft">16dp</item>
    <item name="android:paddingRight">16dp</item>
    <item name="android:paddingTop">24dp</item>
    <item name="android:paddingBottom">16dp</item>
</style>
<style name="Selligent.InAppContents.Html.UpperDivider">
    <item name="android:layout_height">0dp</item>
    <item name="android:visibility">gone</item>
</style>
<style name="Selligent.InAppContents.Html.CardBody">
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#555555</item>
    <item name="android:paddingLeft">16dp</item>
    <item name="android:paddingRight">16dp</item>
    <item name="android:paddingTop">0dp</item>
```

```
    <item name="android:paddingBottom">16dp</item>
</style>
<style name="Selligent.InAppContents.Html.LowerDivider">
    <item name="android:layout_height">0dp</item>
    <item name="android:visibility">gone</item>
</style>
<style name="Selligent.InAppContents.Html.CardLinkContainer">
    <item name="android:paddingLeft">8dp</item>
    <item name="android:paddingRight">8dp</item>
    <item name="android:paddingTop">8dp</item>
    <item name="android:paddingBottom">8dp</item>
    <item name="android:gravity">start</item>
</style>
<style name="Selligent.InAppContents.Html.CardLink">
    <item name="android:padding">8dp</item>
    <item name="android:layout_marginRight">8dp</item>
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#333333</item>
    <item name="android:textAllCaps">true</item>
    <item name="android:background">?android:attr/selectableItemBackground</item>
</style>
<style name="Selligent.InAppContents.CloseButton">
    <item name="android:layout_width">30dp</item>
    <item name="android:layout_height">30dp</item>
    <item name="android:layout_margin">5dp</item>
    <item name="android:padding">10dp</item>
    <item name="android:background">@drawable/sm_ic_close_image</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.

For example, if you simply want to change the background color of a Card to red, you still have to add in your file the whole style:

```
<style name="Selligent.InAppContents.Html.Card">
    <item name="cardBackgroundColor">#FF0000</item>
    <item name="cardCornerRadius">2dp</item>
    <item name="cardElevation">2dp</item>
    <item name="cardMaxElevation">2dp</item>
    <item name="cardPreventCornerOverlap">true</item>
    <item name="cardUseCompatPadding">true</item>
    <item name="contentPadding">5dp</item>
    <item name="contentPaddingBottom">5dp</item>
    <item name="contentPaddingLeft">5dp</item>
    <item name="contentPaddingRight">5dp</item>
    <item name="contentPaddingTop">5dp</item>
</style>
```

The visual elements corresponding to those styles are defined below.

9.3.1.1 Shared style

The button used to close the In App Content when displayed full screen is available for all contents.

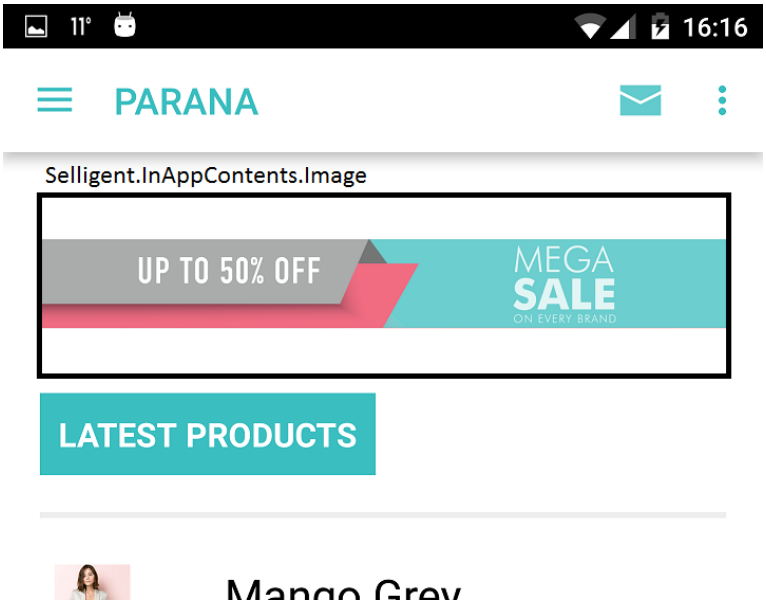
```
<style name="Selligent.InAppContents.CloseButton">
```

Selligent.InAppContents.CloseButton 

9.3.1.2 SMInAppContentImageFragment

<style name="Selligent.InAppContents.Image">

This style can be used, for example, to set a default background color when the image does not completely fill the space reserved for the fragment.

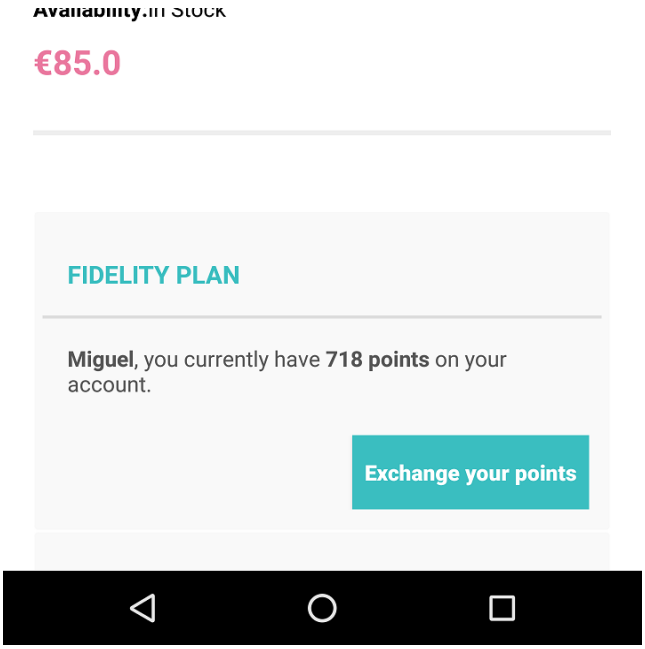


9.3.1.3 SMInAppContentUrlFragment

The url is displayed using the full size of the Fragment, so there is no customization available.

9.3.1.4 SMInAppContentHtmlFragment

Consider the following HTML type In App Content:



The following styles can be used to customize it:

1 <style name="Selligent.InAppContents.Html.Container">

2 <style name="Selligent.InAppContents.Html.Card"> (This one is used on the CardView element)

3 <style name="Selligent.InAppContents.Html.CardContainer"> (This one is used on the RelativeLayout element, first child of the CardView)

4 <style name="Selligent.InAppContents.Html.CardTitle">

5 <style name="Selligent.InAppContents.Html.UpperDivider">

6 <style name="Selligent.InAppContents.Html.CardBody">

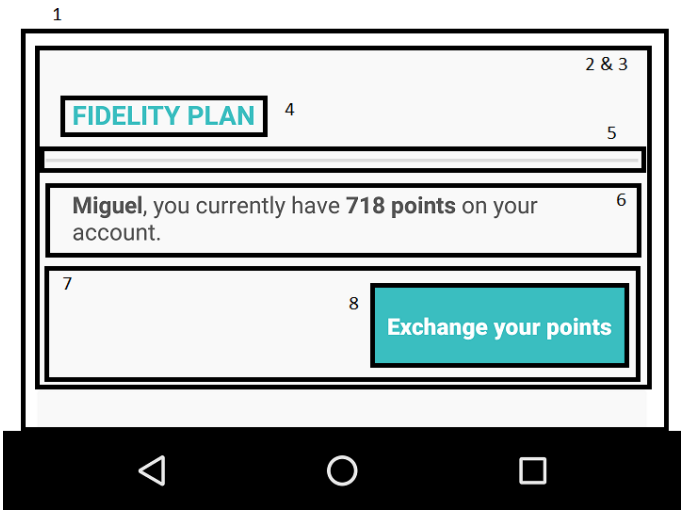
<style name="Selligent.InAppContents.Html.LowerDivider"> (It is not used in this screenshot. It is the equivalent of 5 but situated between the body and the links).

7 <style name="Selligent.InAppContents.Html.CardLinkContainer">

8 <style name="Selligent.InAppContents.Html.CardLink">

Availability: In Stock

€85.0



9.3.2 Refresh

If you ever want to refresh the content of one of our Fragments in case new content is available, you can call the “refresh” method. If the fragment is not displayed, it will just get the (possible) new content from the cache. If it is displayed, the layout will be refreshed too.

9.4 Implementing the In App content without using our Fragments

If you prefer to use your own controls to display the In App Contents, we give you a few methods to get and manage them. They are all available on SMManager.

- `public ArrayList<SMInAppContent> getInAppContents(String category, SMContentType type, int max)`
  This will return all the valid In App contents for a given category and a given type. If you want a certain amount of contents, just pass it to the method, otherwise pass -1 to get all the available ones.

- `public void setInAppContentAsSeen(SMInAppContent inAppContent)`
  This will mark the In App content as seen and send an “open” event to the platform.

- `public void executeLinkAction(Context context, SMLink link, SMInAppContent content)`
  This will execute the action behind the link (open browser, open phone app, etc.) and send a “click” event to the platform.

9.5 Broadcasts

BROADCAST_EVENT_RECEIVED_IN_APP_CONTENTS: When In App contents are received. It contains the number of content per category.
BROADCAST_EVENT_BUTTON_CLICKED : When a link is clicked, it contains an SMNotificationButton object.

10 Events

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

```
SMManager.getInstance().sendEvent(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>).

10.1 Registration/Unregistration

10.1.1 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.

Ex:

```
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().sendEvent(event);
```

10.1.2 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.

Ex:

```
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().sendEvent(event);
```

10.2 Login/Logout

10.2.1 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.

Ex:

```
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().sendEvent(event);
```

10.2.2 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.

Ex:

```
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().sendEvent(event);
```

10.3 Custom

10.3.1 SMEvent

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

Ex:

```
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().sendEvent(event);
```

11 Reload

In case you want to change the web service URL, there is a reload method to restart the SDK. It takes as parameter the same SMSetting object as the start method (all the values must be set in the object, even if they did not change).

12 Broadcasts

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.

12.1 Generic broadcasts

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);
}
```

12.2 Local broadcasts

These broadcast are sent using LocalBroadcastManager. 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_RECEIVED_IN_APP_CONTENTS = "SMReceivedInAppContent"
BROADCAST_EVENT_BUTTON_CLICKED = "SMEventButtonClicked"
BROADCAST_EVENT_WILL_DISPLAY_NOTIFICATION = "SMEventWillDisplayNotification"
BROADCAST_EVENT_WILL_DISMISS_NOTIFICATION = "SMEventWillDismissNotification"
BROADCAST_EVENT_RECEIVED_GCM_TOKEN = "SMReceivedGCMToken";
```

For 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_RECEIVED_IN_APP_CONTENTS:
           HashMap<String, Integer> categories = (HashMap<String, Integer>)
       intent.getSerializableExtra(SMManager.BROADCAST_DATA_IN_APP_CONTENTS);
           //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);
}
```

13 Changelog Customized

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.