# Android SDK Integration Guide - Mobile Engagement

## Android SDK Integration Guide - Mobile Engagement

This guide is a quick start to adding the Phunware Mobile Engagement SDK to an Android app to power Phunware's Marketing Automation module which allows you to create and send broadcast, geofence, and beacon campaigns to your mobile app users.

Android Studio is the recommended development environment for building an app with the Phunware Mobile Engagement SDK.

## Step 1: Add the Phunware Maven remote repository

Insert this block into `allprojects -> repositories`:

```
allprojects {
    repositories {
        maven {
            url
"https://nexus.phunware.com/content/groups/publ
ic/"
        }
    }
}
```

## Step 2: Add the Mobile Engagement SDK as a dependency in your app's b`uild.gradle` file

This will automatically import the required dependency for Phunware Core

```
apply plugin: 'com.android.application'

android {
 ...
}

dependencies {
 ...
    compile
'com.phunware.engagement:mobile-engagement:3.1.
2'
    ...
}
```

## Step 3: (Optional) Add beacon support if you are using beacons

If you would like to take advantage of the Mobile Engagement SDK's beacon support, simply add the `beacon-location-manager` dependency.

With properly configured beacons in your environments, no other code changes are required to take advantage of beacon-based messaging.

```
dependencies {
  ...
  compile
'com.phunware.engagement:mobile-engagement:3.1.
2'
  compile
'com.phunware.engagement:beacon-location-manage
r:3.1.2' {
  exclude group: 'com.phunware.engagement',
module: 'mobile-engagement'
  }
  ...
}
```

## Step 4: Retrieve App ID, Access Key and Signature Key from MaaS Portal

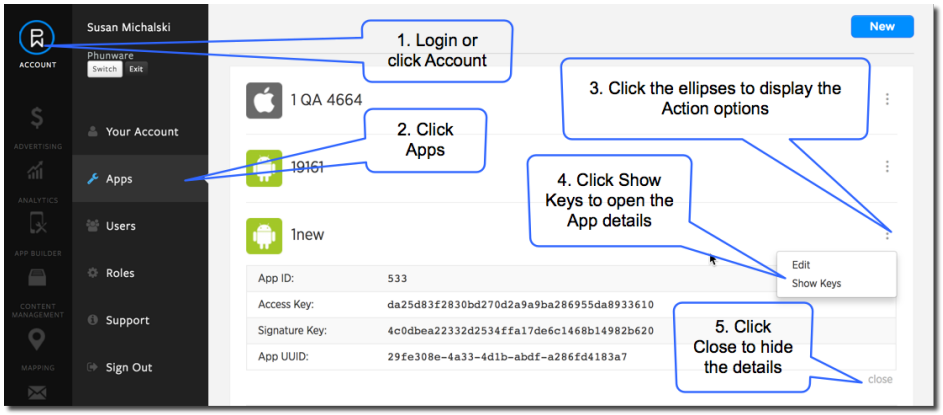
Navigate to Phunware's MaaS portal to find your App ID, Access Key and Signature key.

GCM setup (deprecated, see FCM): https://developers.google.com/cloud-messaging/android/client

FCM setup: https://firebase.google.com/docs/cloud-messaging/android/client

Part of the GCM setup is managed through the Mobile Engagement SDK including:

- Add the play-services-gcm to your application gradle
- changes required to the AndroidManifest




## Step 5: Add Phunware key resources to strings.xml for App Id, Access Key, Signature Key

Add the keys obtained in step 4 to strings.xml

**strings.xml**

```
<string name="app_id">APPID</string>
<string name="access_key">ACCESSKEY</string>
<string name="sig_key">SIGKEY</string>
```

## Step 6: Add Phunware keys for App Id, Access Key, and Signature Key to Manifest

Add the keys obtained in step 4 to Manifest.

```
<meta-data
android:name="com.phunware.APPLICATION_ID"
android:value="@string/app_id" />
<meta-data
android:name="com.phunware.ACCESS_KEY"
android:value="@string/access_key" />
<meta-data
android:name="com.phunware.SIGNATURE_KEY"
android:value="@string/signature_key" />
```

## Step 7: Add Location and Storage permissions to Manifest

This allows you to utilize location-based messages and beacon messaging.

> NOTE: Background location notifications currently cannot work with runtime permissions required for apps targeting Android SDK level 23 and higher, so your targetSdkVersion in your build.gradle file must be 22 or lower.

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

## Step 8: Configure the Mobile Engagement SDK with your environment.

You should only initialize the Mobile Engagement SDK once, after you initialize PwCoreSession.  Once it's complete, you can access the Location, Message and Attribute managers directly.

Once initialization is complete, users will be automatically notified with your custom broadcast messages.  If you have location and storage permissions they will also be able to receive messages for location events, like entering a retail store.

```
public class MyApplication extends Application
{

  @Override
  public void onCreate() {
    super.onCreate();
 //initialize PwCoreSession
 PwCoreSession.getInstance().registerKeys(this)
;

 //initialize Engagement
    new Engagement.Builder(this)
   .appId(/*your app ID from the MaaS portal,
as a long*/)
         .build();

 //start location manager to receive location
based events
 Engagement.locationManager().start();
  }

}
```

## Step 9: Designate an Activity to launch from notifications

Notifications can be customized by extending the `NotificationCustomizationService`. The intent which launches your activity from a notification will have extras with message and promo information.

```
<activity
    android:name=".MessageActivity" >
    <intent-filter>
        <action
android:name="android.intent.action.VIEW" />
        <category
android:name="android.intent.category.DEFAULT"
/>
        <data
android:mimeType="engagement/message" />
    </intent-filter>
</activity>
```

```
if (getIntent().getAction() ==
Intent.ACTION_VIEW) {

    Message intentMessage =
getIntent().getParcelableExtra(MessageManager.E
XTRA_MESSAGE);

Engagement.analytics().trackCampaignAppLaunched
(intentMessage.campaignId(),
            intentMessage.campaignType());

    boolean hasPromo = getIntent()

.getBooleanExtra(MessageManager.EXTRA_HAS_EXTRA
S, false);
    if (hasPromo) {
        long messageId =
intentMessage.campaignId();

Engagement.messageManager().getMessage(messageI
d, new Callback<Message>() {
      @Override
      public void onSuccess(Message data) {
          //do something with the message
      }

      @Override
      public void onFailed(Throwable e) {
          Log.e(TAG, "Failed to get message for
id: " + messageId, e);
      }
  });
   }
```

## Step 10: Show Messages

Using the `MessageManager` you can easily show a list of available messages.

> NOTE: Calls to the MessageManager are asynchronous, and may require loading data from the network.

```
public class MessageListFragment extends
Fragment {

  @Override
  public void onActivityCreated(@Nullable
Bundle savedInstanceState) {

super.onActivityCreated(savedInstanceState);

    Engagement.messageManager().getMessages(new
Callback<List<Message>>() {
      @Override
      public void onSuccess(List<Message> data)
{
        //do something with the Messages
      }

      @Override
      public void onFailed(Throwable e) {

      }
    });
  }


}
```

## Step 11: Customizing Notifications

This service allows app developers to customize notifications. It must be implemented initially even if the user would like to just use standard out of the box notifications

**AndroidManifest.xml**

```
 <manifest ...>
  <application>
    <service
android:name=".MyNotificationEditService"
android:exported="false">
      <intent-filter>
        <action
android:name="com.phunware.engagement.EDIT_NOTI
FICATION" />
      </intent-filter>
    </service>
  </application>
</manifest>
```

**MyNotificationEditService.java**

```
public class MyNotificationEditService extends
NotificationCustomizationService {

  @Override public void
editNotification(Notification.Compat.Builder
notificationBuilder) {
    notificationBuilder

.setSound(RingtoneManager.getDefaultUri(Rington
eManager.TYPE_NOTIFICATION))

.setSmallIcon(R.drawable.ic_motorcycle_black_24
dp);
  }

}
```

## Step 12: Enabling Push Notifications

New in Version 3.1.2 is the ability to use the SDK with or without push notifications as a feature. Enabling push notifications is as easy as enabling them through a simple call to the Engagement API

**MainActivty.xml**

```
Engagement.enablePushNotifications(this.getAppl
icationContext());
```