Push IO

Integration Guide – Android

Version 2.0
Last Updated 8/1/12

THIS DOCUMENT IS PRELIMINARY - INCOMPLETE AND SUBJECT TO CHANGE.

Software License Information
The use of any and all Push IO software regardless of release state is governed by the terms of a Software License Agreement and Terms of Use that are not included here but required by reference.

Privacy Information
The use of Push IO services is subject to various privacy policies and restrictions on the use of end user information. For complete information please refer to your Master Agreement, our website privacy policies, and/or other related documents.

Billing
For information on your account and billing, please contact sales@push.io

Contact Info
Push IO
1035 Pearl Street, Suite 400
Boulder, CO 80302 USA
303-335-0903
support@push.io
http://push.io

Table of Contents

Preface

Push IO is a leading provider of real-time push notification alerts and mobile data delivery. This document provides the necessary information to leverage Push IO for your mobile application.

This document corresponds to v1 of the Push IO Public API and the following components:

PushIOManager for Android version 2.0.0

Definitions

App ID
Each individual application has a unique ID used by the application to communicate with the Push IO service.

Sender Secret
This is a secret shared between Push IO and the developer using direct device messaging which authorizes them to send to an account.

Category
A category defines a specific content type that your app has which users might be interested in. For example, if your app is a sports app, a category may be a specific team.

Audience
An audience defines a group of users based on one or more categories. For example, you might make an audience out of all of the users who have registered to a category for a team in the Western Conference.

Test Device
A test device is a device which is contained in the Test Audience shown under the Set Up > Audiences section of your account at https://manage.push.io. This audience is also automatically populated with devices that have the Push IO Mobile Dashboard that you have used to sign in.

App Integration

Overview

Push IO provides a lightweight PushIOManager library for each supported platform.

The library provides simple methods for registering so Push IO can send push notifications to a device via the platform gateway. The library also provides interfaces for segmenting your users into groups called *Categories*. For instance, you may want to send targeted push notifications only to those users who have expressed an interest in Bird Watching.

Push IO is the only push notification provider which allows you to understand how push notification engagement leads your users to high-value actions like in-app purchases, premium content viewing, and more. The PushIOManager library provides a simple mechanism to capture this push conversion information, which is available to you via the Push IO dashboard at https://manage.push.io.

Integration Prerequisites

Before continuing, be sure you have what you need to integrate Push IO into your app.

1. Sign up for an account at https://manage.push.io
2. Download our Push IO Mobile Dashboard app to easily send your first push!
3. Setup your app and platform(s) at https://manage.push.io
4. Download the PushIOManager library for each platform from Set Up > [platform]
5. Download the pushio_config.json for each platform from Set Up > [platform]

All set? Now you're ready to integrate Push IO into your app and send your first push!

PushIOManager for Android

In order to add the PushIOManager framework to your application, follow these steps:

Step 1: Locate the two files you downloaded from Set Up > [platform] from your Push IO management dashboard.

Step 2: Add the PushIOManager.jar to your project's libs folder.

Step 3: Add the pushio_config.json to your project's assets directory.

Step 4: Now you need to set up your Android Manifest. The following user-permissions are required to use Google Cloud Messaging (replace com.example with your application namespace).

```
    <uses-permission
android:name="com.example.permission.C2D_MESSAGE" />
    <uses-permission
android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.INTERNET"/>

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

Step 5: In the Android Manifest under the “application” tag you need to set up the PushIOBroadcastReceiver. (Once again replace com.example for your application namespace.)

> *Even though Google has moved from C2DM to CCM the permissions still use the c2dm name for backward compatibility with older devices.*

```
        <receiver
android:name="com.pushio.manager.PushIOBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND" >
            <intent-filter>
                <action
android:name="com.google.android.c2dm.intent.RECEIVE" />
                <action
android:name="com.google.android.c2dm.intent.REGISTRATION" />
                <category android:name="com.example" />
            </intent-filter>
        </receiver>

        <activity
android:name="com.pushio.manager.PushIOActivityLauncher"/>
        <service
android:name="com.pushio.manager.PushIOGCMIntentService" />
```

Step 6: Set-up the ensureRegistration call when the application is first starting. This will ensure that registration is maintained between upgrades of the application.

```
PushIOManager pushIOManager =
   new PushIOManager(getApplicationContext(), this, null);
 pushIOManager.ensureRegistration();
```

Step 7: Register with Push IO

You can register a device with Push IO one of two ways. First, if your application gives users a way to specify a preference or favorite, you may want to register for that category you can you push to them relevant content. Note that the PushIOManager has two callbacks that can be optionally implemented that allow you to deal with success and errors. Additionally success and error is logged under the “pushio” tag.

This kind of registration would be tied to a notification UI, allowing the user to select categories.

```
// Register for US and World Headlines
PushIOManager pushIOManager =
   new PushIOManager(getApplicationContext(), this, null);
List<String> categories = new ArrayList<String>();
categories.add("US");
categories.add("World");
pushIOManager.registerCategories( categories, false );


// Unregister for US Headlines
PushIOManager pushIOManager =
   new PushIOManager(getApplicationContext(), this, null);
List<String> categories = new ArrayList<String>();
categories.add("US");
pushIOManager.unregisterCategories( categories, false );
```

If you just want to be able to broadcast to all your users at once, you can simple call ensureRegistration (on application startup). To unregister a device simply call unregisterDevice.

```
// Unregister device
PushIOManager pushIOManager =
   new PushIOManager(getApplicationContext(), this, null);
pushIOManager.unregisterDevice();
```

Step 8: Handle Push
In Android there are two ways to handle push notifications, you can use the PushIOManager built-in notifications, or you can implement them on your own.

For built-in notifications simply leave the last parameter null when creating the PushIOManager. When creating a new notification the PushIOManager will grab icon by name from you application drawables and the sound from your application resources. To handle notification presses you need to register a new intent filter for the action “com.example.NOTIFICATIONPRESSED”, with the “android.intent.category.DEFAULT” category specified in your Android Manifest. This filter will call your activity on a notification press.

```
  <activity android:name="NewsActivity">
            <intent-filter>
                <action
     android:name="com.example.NOTIFICATIONPRESSED"/>
                <category
     android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
  </activity>
```

For custom notifications you need to create your own intent service and register the service in your android manifest. You then pass the action into the setNotificationAction call. From here you can do anything you would like with the notification information. The intent will contain (as extras) “alert”, “sound” and “badge”.

In AndroidManifest.xml:

```
        <service android:name="com.example.CustomNotificationService">
            <intent-filter>
                <action
     android:name="com.example.CUSTOMNOTIFICATION"/>
                <category
android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </service>
```

Call to PushIOManager:

```
        PushIOManager lManager = new
PushIOManager(getApplicationContext(), null, null);

lManager.setNotificationAction("com.example.CUSTOMNOTIFICATION");
```