

Push IO ®

Integration Guide

Version 2.0

Last Updated July 31, 2012

THIS DOCUMENT IS PRELIMINARY - INCOMPLETE AND SUBJECT TO CHANGE.

Copyright © 2009 - 2012 All Rights Reserved Push IO Inc.

Push IO® is a registered trademark in the United States.

All other trademarks are the property of their respective owners.

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	4
Definitions	5
App Integration	6
Overview	6
Integration Prerequisites	6
PushIOManager for iOS	7

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 iOS version 2.0.1

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, and then perhaps only users who are interested in robins.

Push IO is the only push notification provider which provides data that 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!

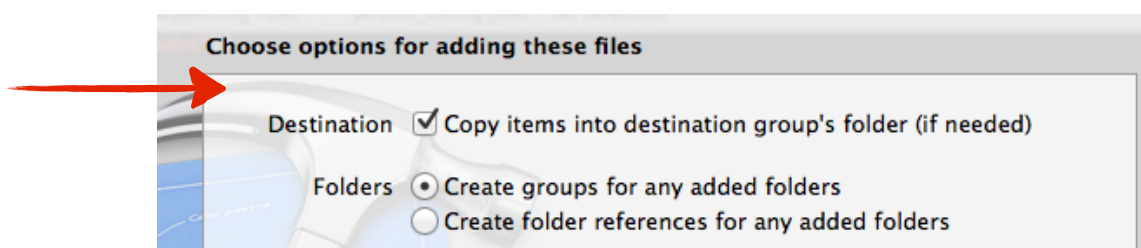
PushIOManager for iOS

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

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



Step 2: Drag the PushIOManager.framework bundle into your project. Be sure to click “Copy items into destination group’s folder”



Step 3: Drag pushio_config.json file to your project. A great location is in the same place your AppDelegate class is stored.

The pushio_config.json must be in your application bundle in order for the PushIOManager read your API key and properly connect to the backend service.

Step 4: Import the PushIOManager header file in your AppDelegate.h

```
#import <PushIOManager/PushIOManager.h>
```

Step 5: Declare the PushIOManagerDelegate in your app your AppDelegate.h

```
@interface AppDelegate : UIResponder <UIApplicationDelegate,
PushIOManagerDelegate>
```

Step 6: Integrate the PushIOManager into your application lifecycle in AppDelegate.m

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Initial setup
    [[PushIOManager sharedInstance] setDelegate:self];
    [[PushIOManager sharedInstance]
didFinishLaunchingWithOptions:launchOptions];

    // Requests a device token from Apple
    [[UIApplication sharedApplication]
registerForRemoteNotificationTypes:UIRemoteNotificationTypeAlert
 | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound];
}

- (void)applicationWillEnterForeground:(UIApplication *)application
{
    [[PushIOManager sharedInstance] applicationWillEnterForeground];
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    [[PushIOManager sharedInstance] applicationDidBecomeActive];
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    [[PushIOManager sharedInstance] applicationDidEnterBackground];
}
```


Step 7: Implement the Apple Push Notification Service callbacks

```

- (void)application:(UIApplication *)application
  didRegisterForRemoteNotificationsWithDeviceToken:(NSData
  *)deviceToken
{
    [[PushIOManager sharedInstance]
    didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}

- (void)application:(UIApplication *)application
  didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
{
    [[PushIOManager sharedInstance]
    didFailToRegisterForRemoteNotificationsWithError:error];
}

- (void)application:(UIApplication *)application
  didReceiveRemoteNotification:(NSDictionary *)userInfo
{
    [[PushIOManager sharedInstance]
    didReceiveRemoteNotification:userInfo];
}

```

Step 8: Implement PushIOManager delegate protocol

```

- (void)readyForRegistration
{
    // PushIOManager has a proper device token, so now you
    // are ready to register.
}

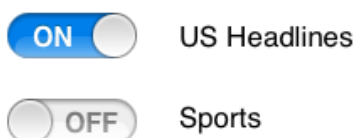
- (void)registrationSucceeded
{
    // Push IO registration was successful
}

- (void)registrationFailedWithError:(NSError *)error statusCode:
  (int)statusCode
{
    // Push IO registration failed
}

```

Step 8: Register with Push IO

You can register a device with Push IO in at least two ways. First, if your application gives users a way to specify a preference or favorite, you may want to register for that category so that you can push to them relevant content.



This kind of registration would be tied to a UI action, such as a UISwitch on/off action.

```
// Register for US Headlines
[[PushIOManager sharedInstance] registerCategory:@"US"];

// Unregister for US Headlines
[[PushIOManager sharedInstance] unregisterCategory:@"US"];
```

If you just want to be able to broadcast to all your users at once, you can register more generically.

```
// Register device, so broadcasts can be sent to the user.
[[PushIOManager sharedInstance] registerWithPushIO];

// Unregister device entirely from Push IO.
[[PushIOManager sharedInstance] unregisterWithPushIO];
```