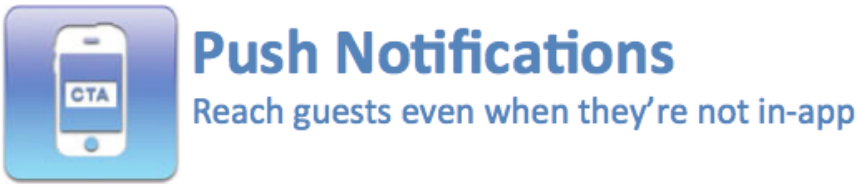
Push Notification Integration Instructions

Added by Caldara, Keey, last edited by Sodhi, Jasjot on Nov 15, 2013



Push Notifications enable your publishing team to reach guests even when the app is not running in order to drive engagement through incentives and awareness.

Content

Getting Started Checklist:

- Email **DI-MobileNetworkHelp@disney.com** for the Urban Airship App key and Secret. Please include:
  - App Name
  - App Icon (optional)
  - Category- Usually Game or Entertainment
  - Admin Person
  - **For iOS:** Push Certificate (You would normally get this from the Submissions team)
  - **For Android:** GCM Key and Package name (Bundle Id)

    ***Note**: As for Helium, Urban Airship is sunsetting this service so you only need to use GCM for both platforms (Google Play and Amazon).
- Integrate the library into your code using the instructions

Background

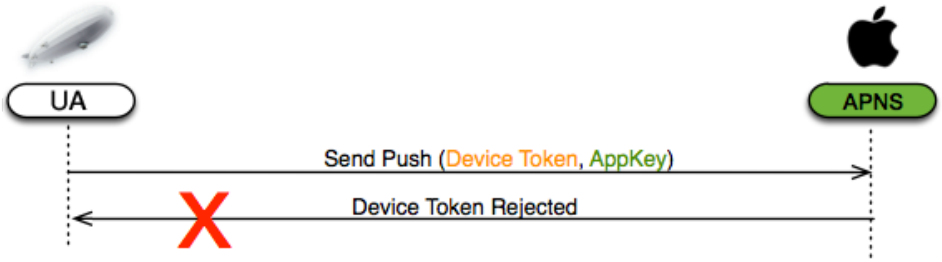
Direct integration with our push provider, Urban Airship, is the recommend means for supporting Push Notifications on iOS and Android. The Mobile Network team will help with this integration, configuring the needed pieces in Disney's Urban Airship account, assisting with issuing test notifications, etc..

Integrating Urban Airship in IOS

Production vs Development IOS Apps in Urban Airship

When you create or edit an application record on our server, you have to select where your app system is *"In development, connecting to test servers,"* or *"In production, connecting to real push servers."* Apple treats the two servers completely separately, so a device token for

development/sandbox will not work on production/distribution.



When building your app using a development provisioning profile, set your Urban Airship application to *In development*, and upload the development Push SSL certificate. To push to an app built with a distribution provisioning profile (either with a release build in Xcode, ad-hoc distribution, or the iTunes App Store), use an application that is *In production*, and upload the production Push SSL certificate.

Because Apple treats development and production/distribution as completely separate instances, we suggest making two applications in the Urban Airship web interface. That way you can continue to build and develop your application even after releasing it without interrupting your users.

Setting up Urban Airship on IOS

Please ensure that you have your Urban Airship App Key and Secret before proceeding any further. If you do not have these yet, you can get these by following the steps mentioned in the Checklist at the top of the page.

Below are the minimum steps required to configure your app(s) to talk to the Urban Airship service. Urban Airship will take things from there, communicating with APNS on your behalf.

Once the Urban Airship pieces are in place, they will maintain two-way communication with Apple, sending notifications and other content on your behalf, and receiving and reporting user activity and engagement data back to you.

Download & Install Our Library & Frameworks

Download and unzip the Urban Airship library found on the **Mobile Network Libraries** page into the same directory as your project.

> Please DO NOT download the SDK from Urban Airship directly

Required Libraries

The core library requires your application to link against the following Frameworks:

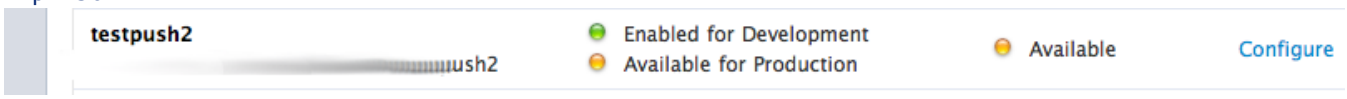
```
libUAirship-<version>.a
CFNetwork.framework
CoreGraphics.framework
Foundation.framework
MobileCoreServices.framework
Security.framework
SystemConfiguration.framework
UIKit.framework
libz.dylib
libsqlite3.dylib
CoreTelephony.framework
CoreLocation.framework
MessageUI.framework (only required for the sample UI)
AudioToolbox.framework (only required for the sample UI)
MapKit.framework (only required for the sample UI)
```

Create AirshipConfig.plist

The library uses a .plist configuration file named AirshipConfig.plist to manage your production and development application profiles.

1. Create two applications within your Urban Airship account: One for development and one for production. For example:
    a. Name_of_your_app_dev
    b. Name_of_your_app_prod
2. Create an AirshipConfig.plist file.
3. Set the following values to the ones in your applications:

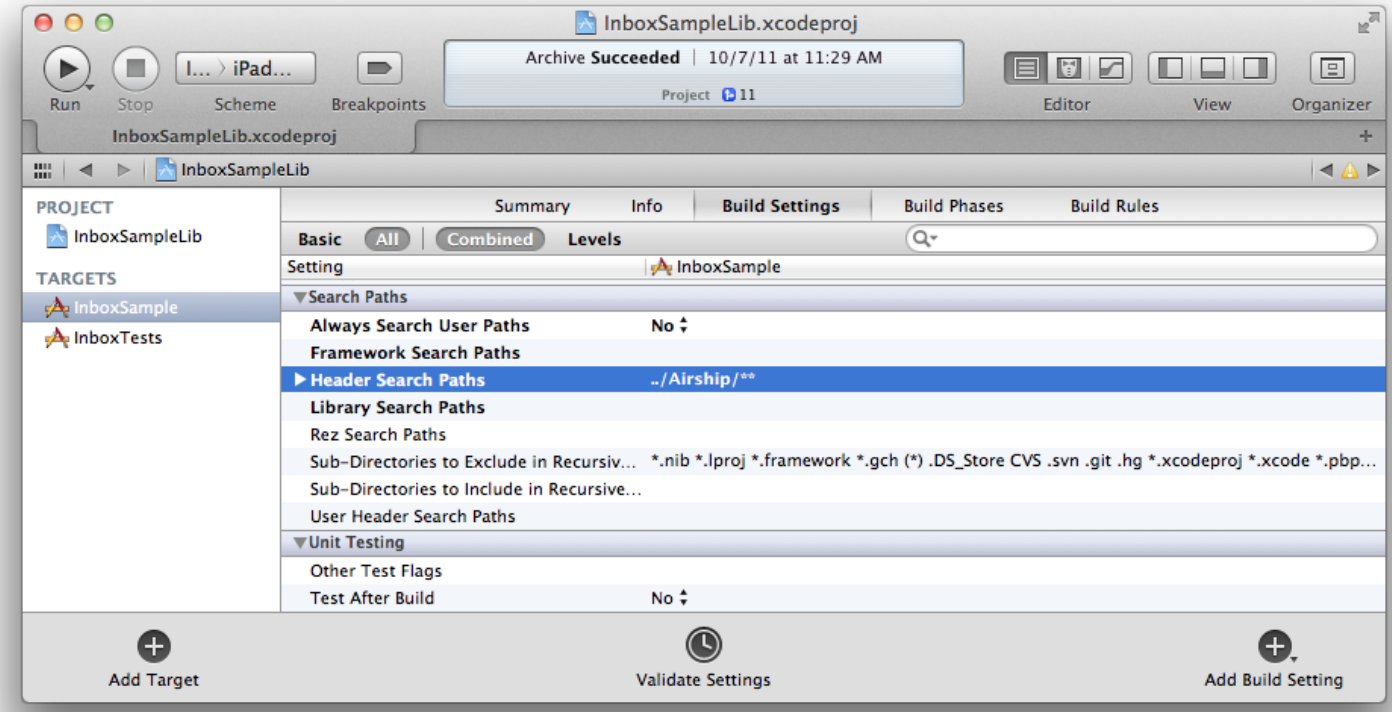
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>inProduction</key>
  <false/>
  <key>developmentAppKey</key>
  <string>Your Development App Key</string>
  <key>developmentAppSecret</key>
  <string>Your Development App Secret</string>
  <key>productionAppKey</key>
  <string>Your Production App Key</string>
  <key>productionAppSecret</key>
  <string>Your Production App Secret</string>
</dict>
</plist>
```



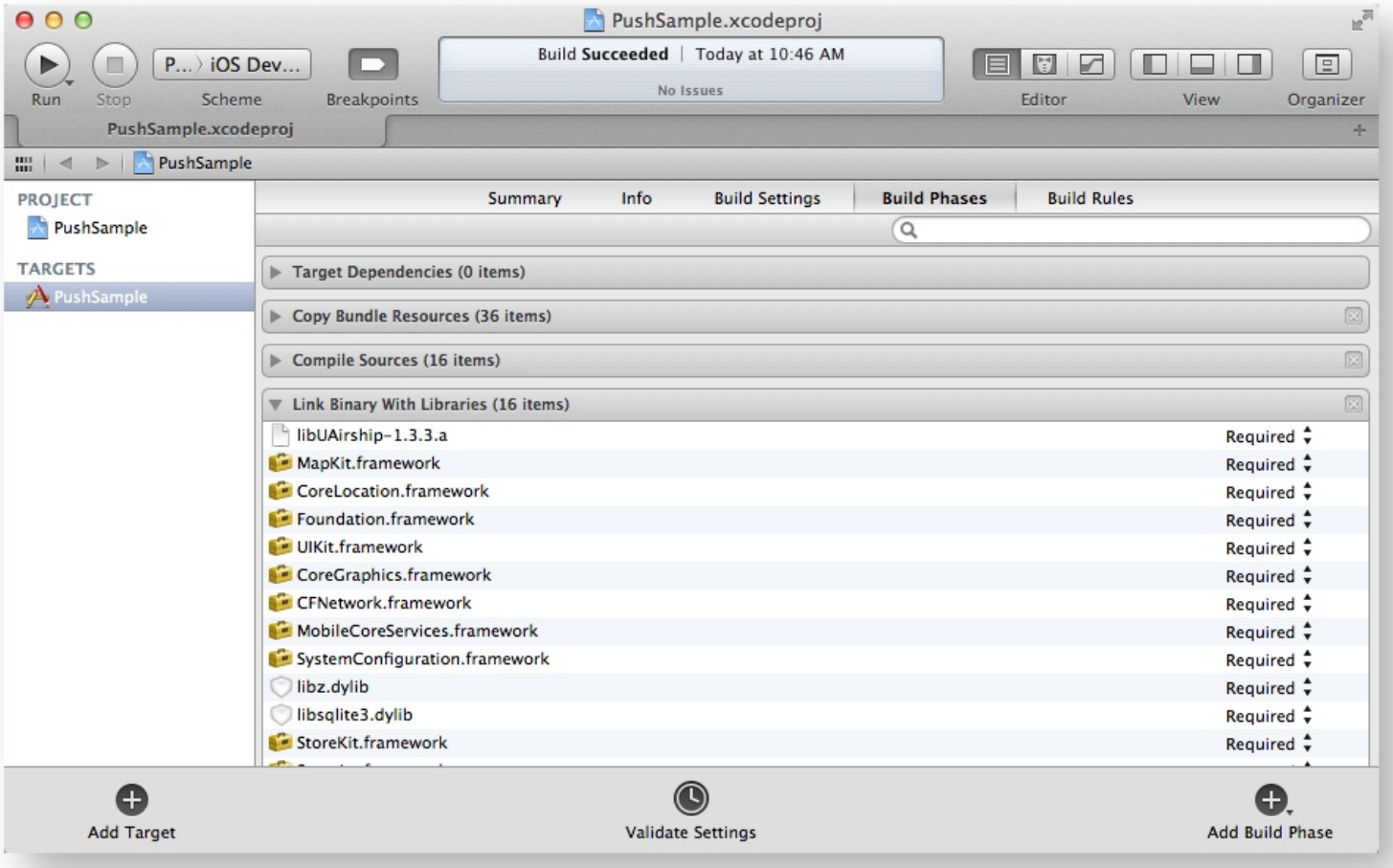
If you are using development builds and testing using the Apple sandbox set inProduction to false. For App Store and Ad-Hoc builds, set it to true. You may also allow the library to auto-detect the production mode by setting detectProvisioningMode to true.

Build Settings

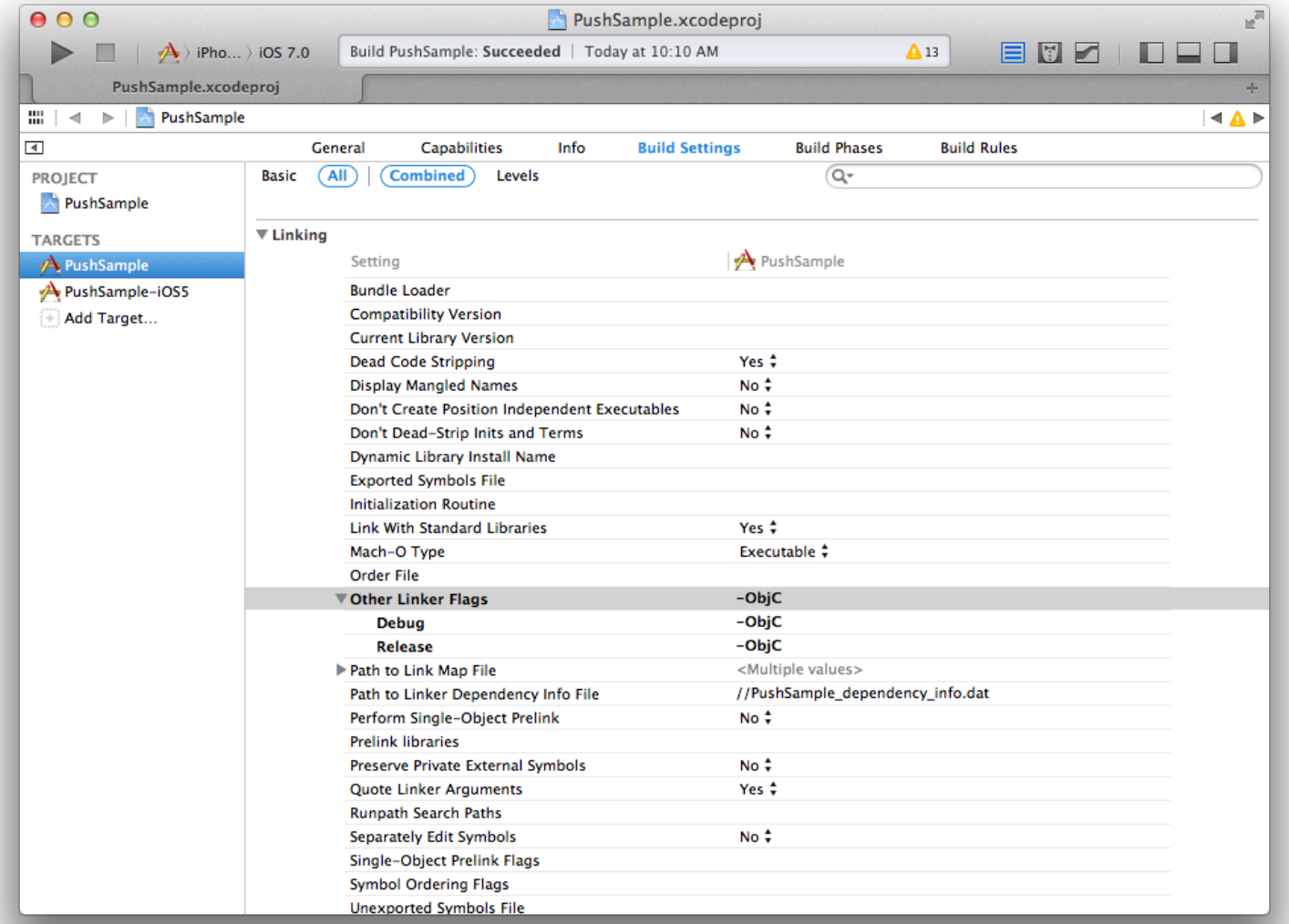
Header search path - Ensure that your project's header search path includes the Airship directory (recursive).



Link against the static library. Add the libUAirship.a file to the “Link Binary With Libraries” section in the Build Phases tab for your target.



Add “-ObjC” linker flag to “Other Linker Flags” to prevent “selector not recognized” runtime exceptions.



Objective C Implementation

```
#import "UAirship.h"
#import "UAConfig.h"
#import "UAPush.h"

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // Populate AirshipConfig.plist with your app's info from https://go.urbanairship.com
    // or set runtime properties here.
    UAConfig *config = [UAConfig defaultConfig];

    // You can also programmatically override the plist values:
    // config.developmentAppKey = @"YourKey";
    // etc.

    // Call takeOff (which creates the UAirship singleton)
    [UAirship takeOff:config];
}
```

Set the desired notification types on [UAPush shared]. Sounds, alerts and badges are the default types, but if you are using Newsstand or would like a custom set of types, you can set these immediately after [UAirship takeOff] is called. The library will register to receive these types of notifications as soon as push is enabled.

```
// Request a custom set of notification types
[UAPush shared].notificationTypes = (UIRemoteNotificationTypeBadge |
                                     UIRemoteNotificationTypeSound |
                                     UIRemoteNotificationTypeAlert |
                                     UIRemoteNotificationTypeNewsstandContentAvailability);
```

Push is enabled by default, but if you would like to defer the iOS push permission prompt, you can disable notifications by default. To do so, add the following after the call to [UAirship takeOff]:

```
[UAPush setDefaultPushEnabledValue:NO];
```

To enable push later on in your application:

```
// This will trigger the proper registration or de-registration code in the library.
[[UAPush shared] setPushEnabled:YES];
```

Register Your Device

1. Once you have this implemented, run & compile your code.
2. Add the application to your phone.
3. When you open the app on your phone, it will ask for permission to send notifications. If you aren't prompted to receive notifications, you should walk back through the steps on this page.

To make sure your device token is registered:

1. Log in to go.urbanairship.com
2. On the left-hand sidebar, navigate to Audience ==> Device Tokens
3. If your device token registered properly, it will appear in the Device Tokens listing.

Send a Test Notification

There are as many ways of using our HTTP-based API as there are programmers, but let's get started as quickly as we can. Here's an example using curl, a command-line tool that comes with OS X:

```
curl -X POST -u "<application key>:<master secret>" \
   -H "Content-Type: application/json" \
   --data '{"device_tokens": ["<token>"], "aps": {"alert": "Hello!"}}' \
   https://go.urbanairship.com/api/push/
```

This does a few things:

- It sets the HTTP Basic Authentication header with the application key as the user name, and the master secret for the password.
- It sets the Content-Type header to application/json, to tell our server you're sending JSON.
- It sets the data you're sending, including the device token as the recipient.

You should receive a 200 response code, with no data in the body. This means that we accepted your request, and are processing it and sending it on to Apple!

If it worked: Congratulations!

If it didn't work

In the iOS Provisioning Portal, click on *App IDs* and locate your application in the list. Next to it are two settings for push notifications with yellow or green status icons:

1. Log in to your Urban Airship account
2. Check your error console

Your Error Console provides help to debug any issues you may have as you get started. There should be reasons why the notification wasn't received.

If you used curl to send the notification, confirm that curl is working properly by using the echo endpoint:

```
curl https://go.urbanairship.com/api/echo/?msg=hello
```

If curl is set up properly, you should receive the following response:

```
{
   "msg": "hello"
}
```

If that fails, try a different testing tool, like HTTP Client.

If it succeeds, or if you get a different testing tool, try this to send a notification.

Check the error console in the *Xcode Organizer*; that can sometimes have useful information. Otherwise, go back through the process, paying close attention to:

The application ID, bundle ID, provisioning profile, and SSL certificate: All of these must point to the same identifier.

The device token: make sure you copied it correctly.

Integrating Urban Airship in Android Push

Production vs Development Android Apps in Urban Airship

Begin by creating both a development and production version of your app, and test GCM functionality using the development app. All new apps use GCM by default. Apps pre-existing prior to UA GCM support currently require assistance from support.

Urban Airship Android Setup

Before you move forward, please ensure you have your Urban Airship App key and secret by following the checklist at the top of the page.

Add the Library JAR to your Project

First, download the current client library jar from the Mobile Network Libraries page.

> Please DO NOT download the SDK from Urban Airship directly

This zip file contains the library JAR file, as well as Eclipse sample projects with the JAR file already included.

If you are using Eclipse to develop your project, add the JAR to the libs folder in your project directory.

Otherwise, if you're using a different IDE, just add a reference to the urbanairship-lib-<version>.jarfile in your classpath, and ensure that it is bundled in your final application.

Configuring your Client Application

Push requires that CoreReceiver, PushService, EventService, UrbanAirshipProvider and PushWorkerService* (see note below on PushWorkerService) be enabled in the manifest, as well as networking and storage-related permissions.

> * PushWorkerService is no longer required with UA Android lib ver. 3.0.0+.

Below is an example manifest, taken from the bundled "Push Sample" app. Please note that if you are testing with your own application, you cannot copy and paste this manifest verbatim and expect it to work. In particular, anywhere within this example that you see the package com.urbanairship.push.sample. This must be replaced with your app's fully qualified package name, including the intent filter categories for GCMPushReceiver, your custom intent receiver, and the C2D_MESSAGE permissions at the bottom of the example file. This is especially important when using GCM, as without proper permissions settings, your app will not receive registration responses from Google, and this will prevent your APID from being registered with Urban Airship.

Enabling Push Notifications

Enabling or disabling push notifications is a preference often best left up to the user, so by default, push is disabled in the client library. For testing purposes, enabling push at startup simply requires a few extra lines of code to your main application class.

The Urban Airship client library automatically starts push upon takeOFF, so for basic support, the amount of setup required is next to zero. Here is a simple example:

Do I have to integrate these libraries?

On iOS, you don't have to. The only thing that library is doing that we absolutely need done, is registering device tokens with Urban Airship (and unregistering them when they go "inactive," or when the user has disabled Push in your application). If you have a server-side component to your product that can easily receive device token information from your app, and you prefer to

send that token information from your server to Urban Airship's servers, that's fine. Just see their documentation for their iOS Push API.

On Android, although there's an analogous server API for registering APIDs, you still need to drop in Urban Airship's library, as it provides support for actually triggering and displaying the notifications on device.

Additional Resources

Where do I turn if Push isn't working when I'm expecting it to?

If you're interested in learning more about Apple's Push Notification Services in general, including local notifications (Urban Airship is providing us support for remote notifications, that come from a server), visit Apple's developer website and read Using Local and Push Notifications.

Apple has some tips for troubleshooting Push Notification problems, as a part of Tech Note TN2265: Troubleshooting Push Notifications.

Urban Airship has some tips too, for both iOS and Android. For iOS, see the "If it didn't work" subsection of 4. Send a Test Notification. For Android, see the "If it didn't work" subsection of Send Your First Push.

OK, it's tested to be working. Now what?

There are various ways you can send push notifications once your app is confirmed working, and is live to users. Here are the most common:

1. Use the Urban Airship dashboard to send pushes to specific users using a list of device tokens or APIDs, or send a broadcast to all of your users.
2. Use the Urban Airship server API to send pushes programmatically, be they broadcasts, or to a group of users (e.g. to only those users of Where's My Water? who have opted in to find out news about Swampy), or to an individual user (e.g. to a specific user that their friend has just challenged them to a game). Check out the docs for sending iOS pushes, and Android pushes, using UA's server API.
3. Configure the Urban Airship dashboard to send a push every time an RSS feed of your choosing updates, either to a group of users by device token, or as a broadcast. If you want a fairly automatic system for pushing, but find that using a CMS or blogging system to update an RSS feed is easier than coding to a server API, this might be a good fit for you. Find out more here.

The Mobile Network team can help you with any operation of the Urban Airship dashboards to which you may not have direct access.

How/Where can I request features and/or file bugs?

You can add any feature requests or file bugs by emailing DI-MobileNetworkHelp@disney.com

Like Be the first to like this