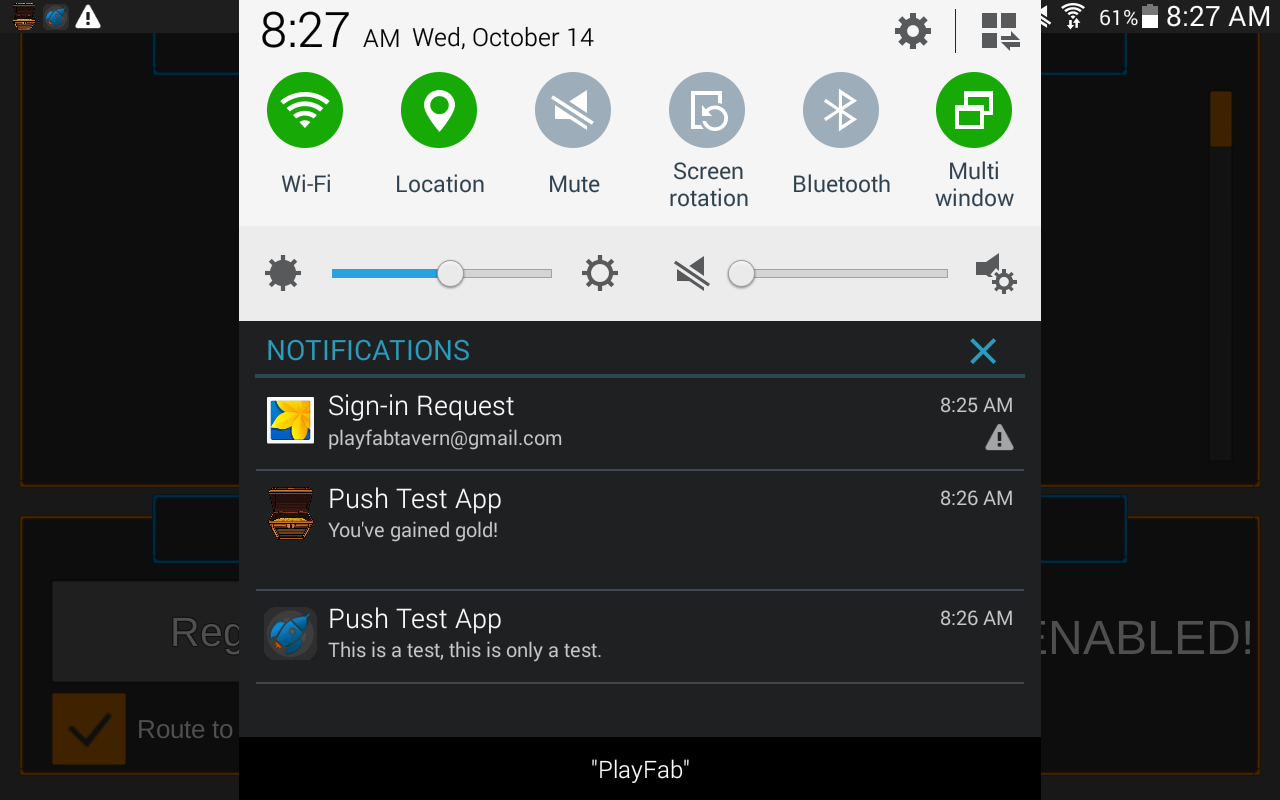
# Welcome to PlayFab’s Push Notification Crash Course

* [Download the Push Sample Project / Debugger](https://s3-us-west-1.amazonaws.com/external-large-file-access/PlayFabClientSample.zip)
* [Link to the Unity SDK & Plugin Source](https://github.com/PlayFab/UnitySDK)

# Introduction

Push notifications give developers a channel to send an immediate, customized, message to their player’s home screens. Using PlayFab's push feature, developers may send unlimited push notifications to any number of devices completely free of charge.

As a general guideline, always tell players up front how you will be using the push notification service. Explaining how your game rewards or communicates via push notification can be the difference in building community engagement versus driving players away.



*An screenshot from an Android device's notification area showing 2 push notifications.*

## The Push Notification System

Push notifications are possible due to a delicate linkage of three major systems:

1. The vendor-specific channel ( Google, Apple, etc )
2. The player’s OS / device (Android, iOS, etc)
3. Cross-platform message routing (PlayFab via [Amazon SNS](https://aws.amazon.com/sns/))

If any of these systems are unlinked, clients will stop getting notifications. It is fairly simple to accidentally change one of the three systems and thereby break the link. Furthermore, it is difficult to know at any individual point in the flow if all of the systems are configured properly.

## The PlayFab Push Routing System

Every PlayFab title can enable one Google (GCM) and one Apple Push (APNS, APNS\_SANDBOX) notification channel at a given time. This is configured using the PlayFab Admin API.

### Configure Android Notification Channel

1. Ensure you have the right Google project settings
   * You have a GooglePlay app that has been linked to a Google Development Console project. Save the **Project ID**, as it will be used in registering your Android clients. This ID should be a ~12 character sequence visible from both GooglePlay and Google Developer consoles.
   * In the Google Development Console:
     + Enable the “Google Cloud Messaging for Android” API
     + Add a **Server API Key** to your credentials.
     + Save the ~40 character string to use to link the Google channel to the PlayFab service.
2. Provide your API key to: [**SetupPushNotification**](https://api.playfab.com/Documentation/Admin/method/SetupPushNotification)
   * **Name** : “your\_game\_name”
   * **Platform** : “GCM”
   * **Credential** (Google server API key) : “AIza\*\*\*\*\*ppo”
   * **OverwriteOldARN** : true
3. You should get an HTTP 200 OK response with similar data

{

“code” : 200,

“status” : “OK”,

“data” :

{

“ARN” : “arn:\*\*\*\*\*\*\*/GCM/your\_game\_name”

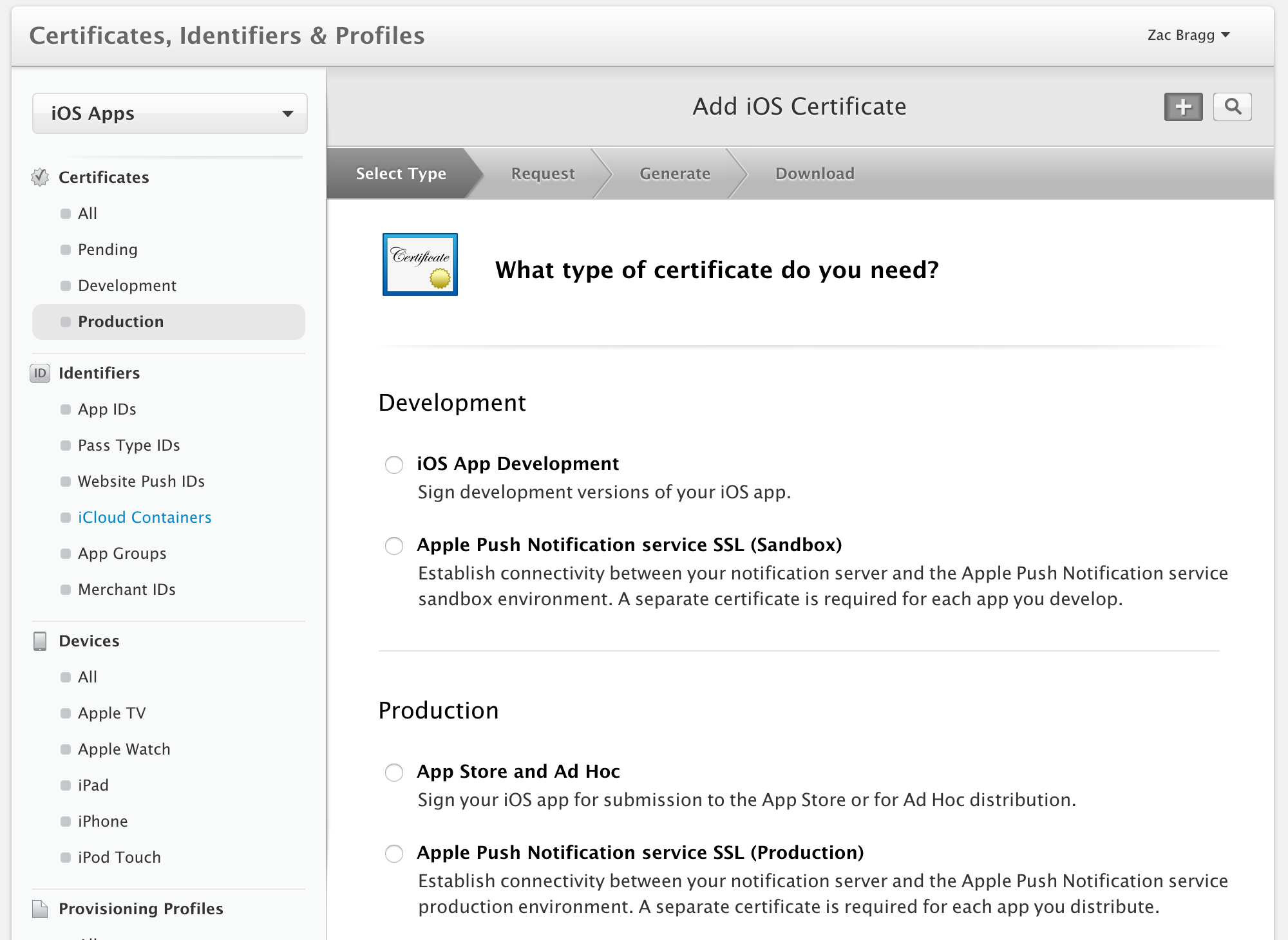
}

}

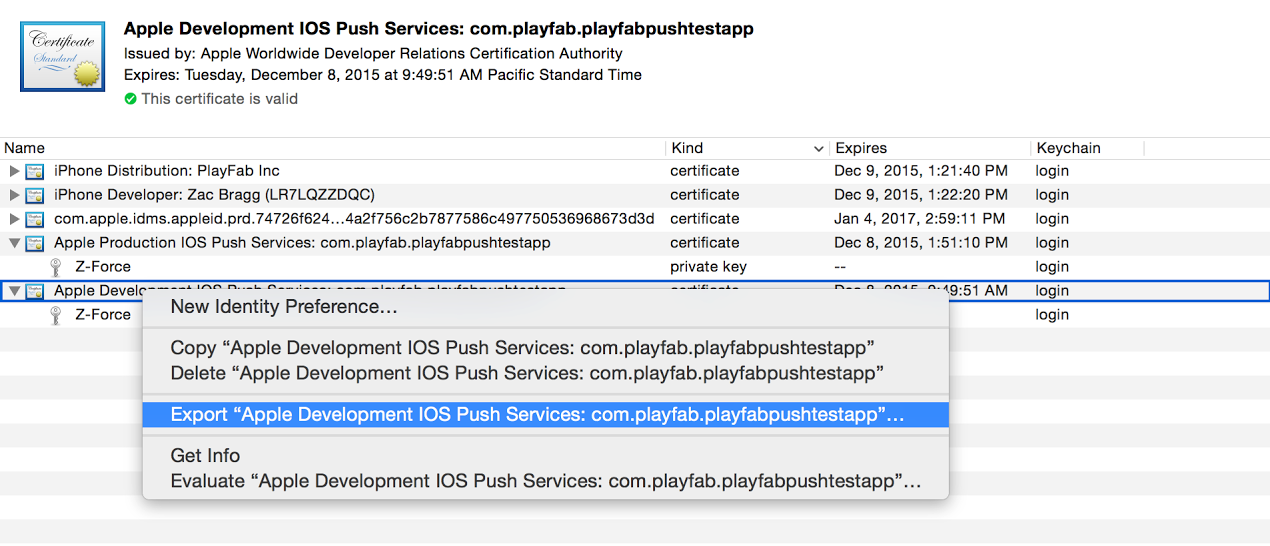
1. Congratulations! You have now configured your title’s Android messaging channel.

### Configure iOS Notification Channel

1. Ensure you have an active iOS development (APNS\_SANDBOX) or production (APNS) certification. If you do not, create one from the Apple Developer Portal under *Certificates, Identities and Profiles*:



1. Export the certificate of your choosing (Development or Production):
   * Begin by downloading a copy of your certificate from the Apple Developer Portal under *Certificates, Identities and Profiles*. This will create a \*\*\*.cer file.
   * Open and install the certificate into your Keychain Access
   * View your installed certificate under the Certificates subcategory
   * Export your Keychain Access Certificate to \*\*\*.p12 format:



* + Convert the .p12 file to a .pem file using the following console command:
    - **openssl pkcs12 -in apns-dev-cert.p12 -out apns-dev-cert.pem -nodes -clcerts**
  + Open the .pem file in a text viewer and observe your **RSA Private Key** & **Certificate.** These strings are used to link the Apple channel to the PlayFab service
  + Condense the multi-line keys into a single-line string:
    - First add a \n to each line ending between:
      * -----BEGIN CERTIFICATE----- & -----END CERTIFICATE-----
      * -----BEGIN RSA PRIVATE KEY----- & -----END RSA PRIVATE KEY-----
    - Delete the line breaks, leaving one contiguous string for both the certificate and the private key

1. Provide your RSA Private Key & Certificate to: [**SetupPushNotification**](https://api.playfab.com/Documentation/Admin/method/SetupPushNotification):
   * **Name** : “your\_game\_name”
   * **Platform** : “APNS” or “APNS\_SANDBOX”
   * **Key** (Certificate) : “-----BEGIN CERTIFICATE----- \n\*\*\* -----END CERTIFICATE-----”
   * **Credential** (RSA Private Key) : “-----BEGIN RSA PRIVATE KEY-----\n\*\*\* -----END RSA PRIVATE KEY-----”
   * **OverwriteOldARN** : true
2. You should get an HTTP 200 OK response with similar data:

{

“code” : 200,

“status” : “OK”,

“data” :

{

“ARN” : “arn:\*\*\*\*\*\*\*/APNS\_SANDBOX/your\_game\_name”

}

}

1. Congratulations! You have now configured your title’s iOS messaging channel.

## Client Configurations

Developers are free to implement their own push listener client for Android or iOS; however, we have created a sample Unity3d v5.2 project for testing purposes. This sample shows how to integrate our Android GCM plugin. The following code samples are excerpts from PlayFab's **push notification sample project**.

### Registering your Android Client for Push

Our Android plugin uses a callback pattern for communicating between Unity and the native GCM APIs. When your client starts, you will want to register for your desired callbacks and initialize the GCM plugin using your **Project ID**.

* Run on client **Start**():

if(!string.IsNullOrEmpty(this.GoogleProjectId))

{

PlayFabGoogleCloudMessaging.\_RegistrationReadyCallback += OnGCMReady;

PlayFabGoogleCloudMessaging.\_RegistrationCallback += OnGCMRegistration;

PlayFabAndroidPlugin.Init(this.GoogleProjectId);

}

* **OnGCMReady**(bool status) will be called after **PlayFabAndroidPlugin.Init**() completes. At this point we can now request a Push registration token:

PlayFabGoogleCloudMessaging.GetToken();

* **OnGCMRegistration**(string token, string error) will be called after **PlayFabGoogleCloudMessaging.GetToken()** completes. At this point, if no errors were thrown, we can now call the PlayFab API [**AndroidDevicePushNotificationRegistration**](https://api.playfab.com/Documentation/Client/method/AndroidDevicePushNotificationRegistration)**.**

if(!string.IsNullOrEmpty(token))

{

// success

Debug.Log("GCM Init Success");

AndroidDevicePushNotificationRegistrationRequest request = new Android...Request();

request.DeviceToken = token;

request.SendPushNotificationConfirmation = true;

request.ConfirmationMessege = "Thanks for playing!";

PlayFabClientAPI.AndroidDevicePushNotificationRegistration(request, (Android...Result result) =>

{

Debug.Log(“Success!”);

}, OnPlayFabError);

}

else

{

// error happened

Debug.Log("Push Token was null or empty: ");

}

* + If no errors occurred, congratulations! Your Android client has been successfully linked to your title’s Android channel. If you used the SendPushNotificationConfirmation = true flag, you should have immediately received a test push notification with whatever was in the ConfirmationMessege parameter.

### Registering your iOS Client for Push

For iOS, developers will need to rely on the default behavior of how iOS handles push notifications as PlayFab does not offer a native implementation on Unity at this time.

By default, notifications received while the game is in the background will be routed to the notification area. Alternatively, notifications received while the game is the active app will be received silently, and will not be visible in the notification area.

* Code from Sample, ran on client **Start**():

// must be called before trying to obtain the push token

// an asynchronous call with no callback into native iOS code that takes a moment or two before   
 // the token is available. (so spin and wait, or call this one early on)

// this will always return null if your app is not signed

UnityEngine.iOS.NotificationServices.RegisterForNotifications(UnityEngine.iOS.NotificationType.Alert | UnityEngine.iOS.NotificationType.Badge | UnityEngine.iOS.NotificationType.Sound, true);

* At this point, if the user has opted in for notifications, we can now call the PlayFab API [**RegisterForIOSPushNotification**](https://api.playfab.com/Documentation/Client/method/RegisterForIOSPushNotification)**:**

string hexToken = string.Empty;

byte[] token = UnityEngine.iOS.NotificationServices.deviceToken;

if(token != null)

{

RegisterForIOSPushNotificationRequest request = new RegisterForIOSPushNotificationRequest();

request.DeviceToken = System.BitConverter.ToString(token).Replace("-", "").ToLower();

hexToken = request.DeviceToken;

Debug.Log (hexToken);

PlayFabClientAPI.RegisterForIOSPushNotification(request, (RegisterForIOSPushNotificationResult result) =>

{

if(callback != null)

{

callback();

}

}, OnPlayFabError);

}

else

{

Debug.Log("Push Token was null!");

}

* If no errors occurred, congratulations! Your iOS client has been successfully linked to your title’s Apple notification channel. SendPushNotificationConfirmation = true flag has been omitted from this example as it will never be displayed without a plugin.

## Testing the Setup

For additional testing, you can send as many push notifications as you need via the PlayFab server API. You can use PlayFab's CloudScript service as a quick and secure server API environment. From there, push notifications can be sent as needed. Upload or append the following code to your title’s CloudScript:

//JavaScript

handlers.SendPushNote = function (args)   
{  
 var request = {};  
 request.Recipient = args.Id == undefined || args.Id == null ? currentPlayerId : args.Id;  
 request.Message = args.Msg == undefined || args.Msg == null ? "" : args.Msg;  
 request.Subject = args.Subject == undefined || args.Subject == null ? "" : args.Subject;  
 server.SendPushNotification(request);   
}

This CloudScript can be called from any client, as shown below:

void RequestTestPushNotification()

{

RunCloudScriptRequest request = new RunCloudScriptRequest();

request.ActionId = "SendPushNote";

request.Params = new { Msg = "This is a test, this is only a test.", Subject = "Testing Push" };

PlayFabClientAPI.RunCloudScript(request, null, OnPlayFabError);

}

Additionally, web API testing tools such as Postman & curl make testing push configurations a simple task.

1. \*a\* Link to Curl & Postman basics \*/a\*

## Push Notification Best Practices

This is an awesome power. When abused, however, it can drive away users and cause them to not only opt out of future notifications but to leave a game entirely. If used judiciously, though, push notifications can be one of your best tools for building and maintaining your game's community.

Please see our blog post for additional push notification strategies and best practices:

1. **https://playfab.com/blog/push-it-real-good-how-get-players-say-yes-push-notifications/**

Push notifications are a fire and forget message protocol, similar in vein to UDP. There is no guarantee that your players will receive, open, or engage with your message. With that caveat, it is a good practice to use messages as bonus content rather than as a critical part of your gameplay loop.

For iOS platforms, apps are given a one-time dialog prompt from the OS allowing the user to determine the status of push notifications. After the user has made their initial selection, this setting will persist until:

1. The app is updated or reinstalled
2. The user makes changes to the settings from their phone’s settings menu

For Android apps, notifications are enabled by default and can be turned on and off at will from the client. It is a good practice, however, to re-initialize your notification listeners with every session.

## Troubleshooting iOS

* [Verify that you have valid .pem files](http://docs.aws.amazon.com/sns/latest/dg/mobile-push-apns.html)
* Ensure that your certificate is used by XCode to sign your app
* Ensure that your signing certificate matches the PlayFab platform. When running [**SetupPushNotification**](https://api.playfab.com/Documentation/Admin/method/SetupPushNotification), use **OverwriteOldARN = true** to rebind the channel to a newplatform. Only one platform (APNS or APNS\_SANDBOX) can be active on a title at a given time.
  + APNS = Apple Production Certificate
  + APNS\_SANDBOX = Apple Development Certificate

## Troubleshooting Android

* Ensure that the client plugin is initializing with the proper values
* Ensure that the messages are being received on the client. The PlayFab plugin will output all push messages to the default adb logcat stream.
* Ensure that your plugin has been successfully integrated into your AndroidManifest.xml. This is already complete when using the PlayFab plugin.
* View the source project for our [Android plugin](https://github.com/PlayFab/UnitySDK/tree/master/UnityAndroidPluginSource)

## Sample Walkthrough and Advanced Behavior

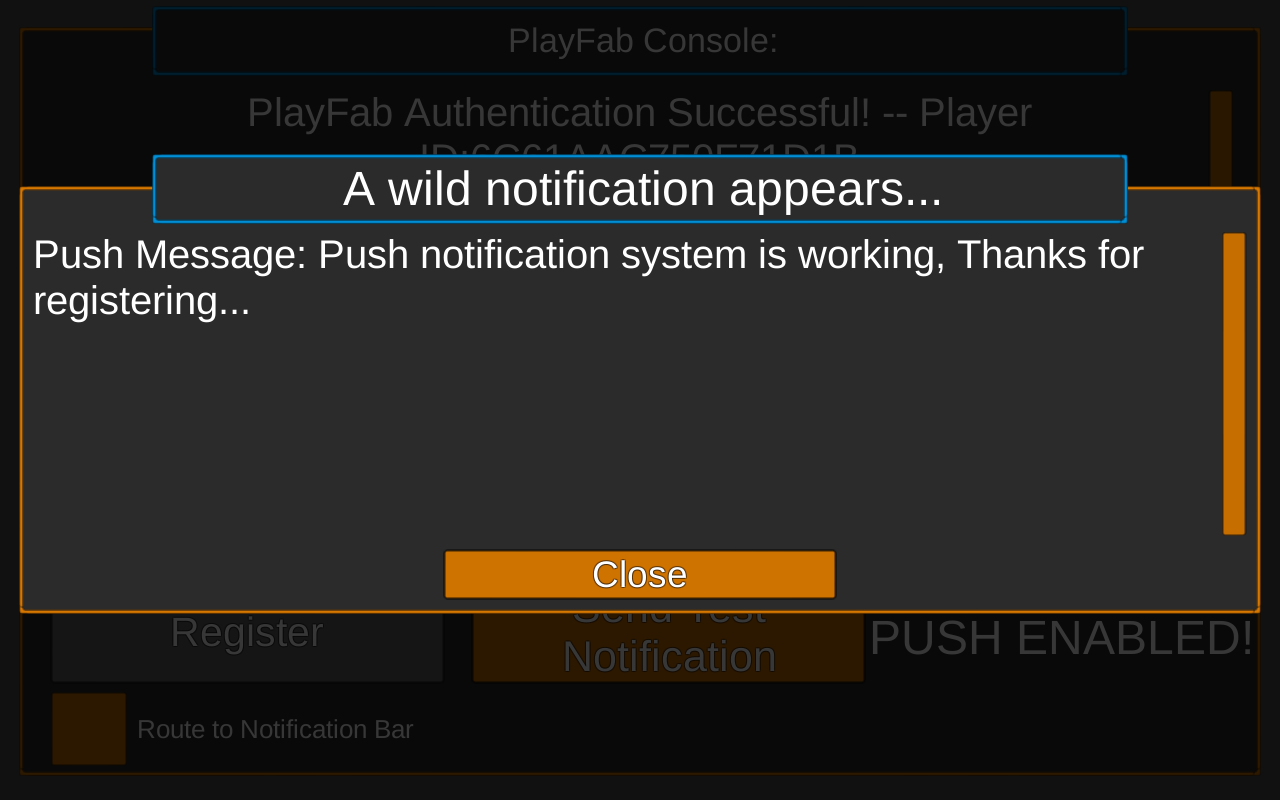
Now that you have a working push notification system, you are ready to start experimenting with some advanced push techniques. These techniques allow you to display custom icons, play custom sounds and parse custom JSON content. Using these advanced features, you can:

1. Send multiple classes of push notifications that look, sound and behave differently
2. Deep-link from a push notification into very specific game screens (like stores and battles)
3. Record push history for your players, allowing them to view their notification history at a their convenience
4. Analyze, track, and compare the effectiveness of your push campaigns

When running the sample project you will see (when built and run from an android device):

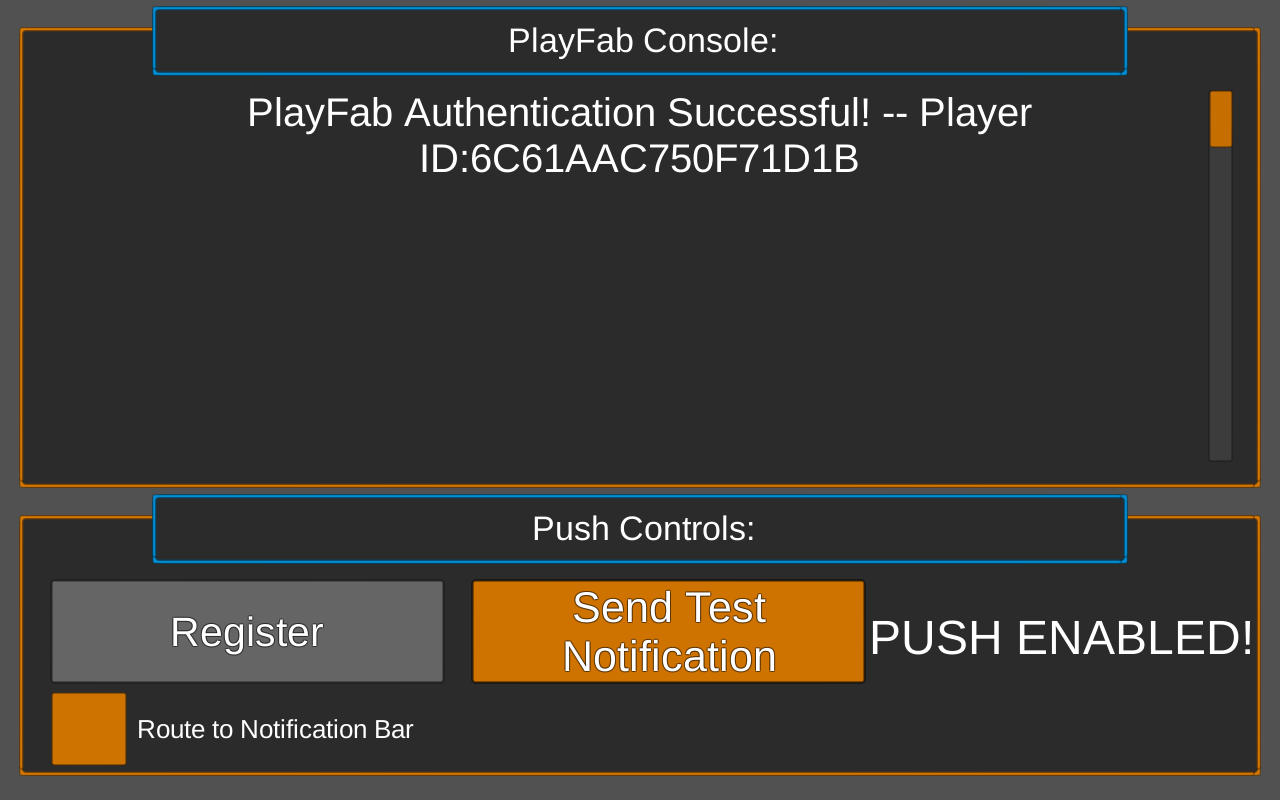
*On Start() the app authenticates automatically using* [*LoginWithAndroidDeviceID*](https://api.playfab.com/Documentation/Client/method/LoginWithAndroidDeviceID)*.*

*After authentication, the player can now register for push notifications.*



*Immediately after registering, when you receive the initial testing push as shown above*

*you will know that you have successfully configured your title to work with GCM.*

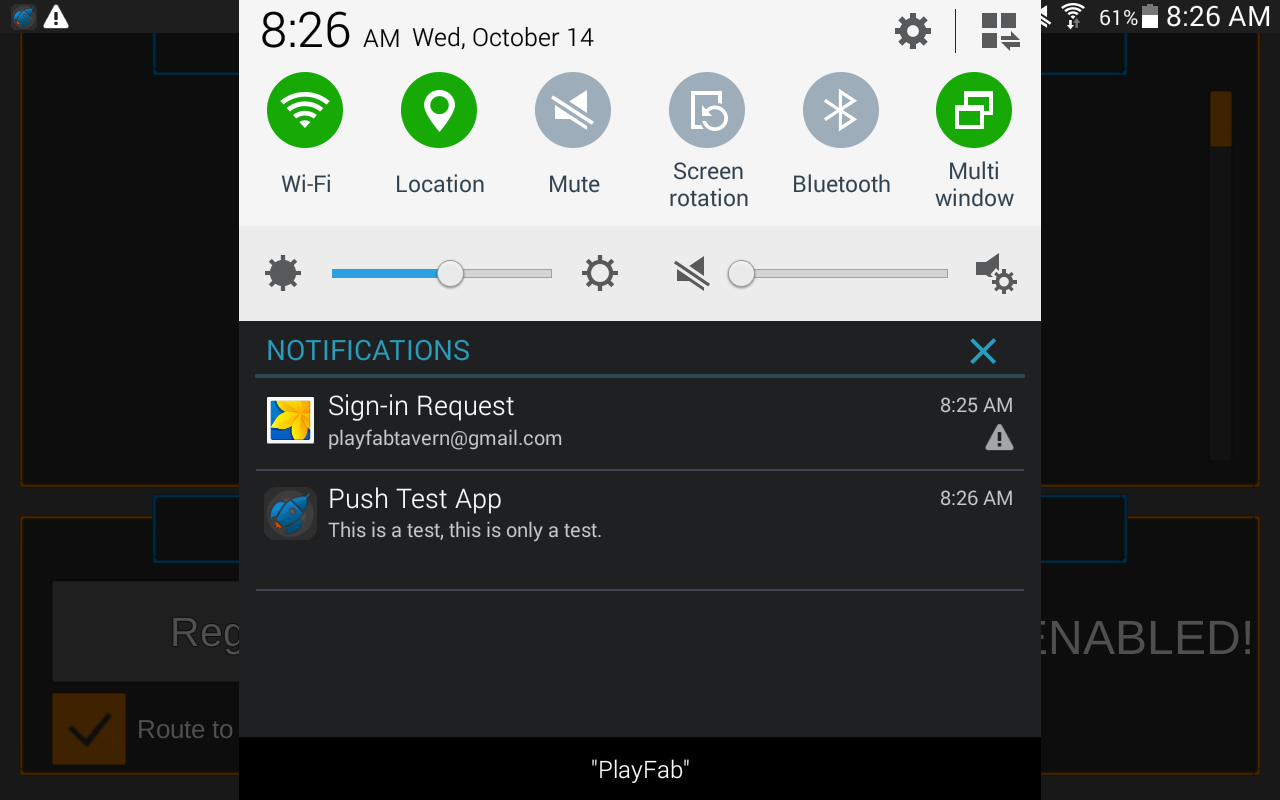


*For additional testing you can press the Send Test Notification button.*

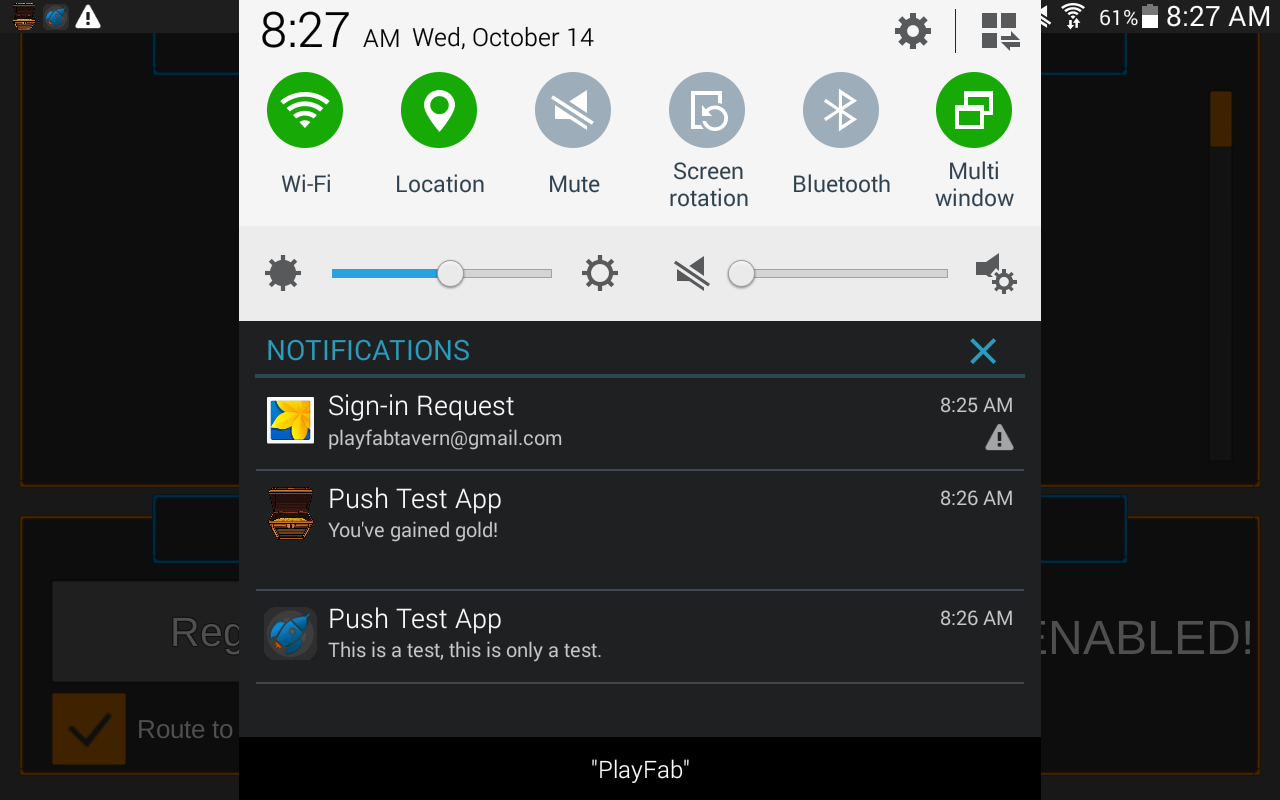
*This will trigger a simple push notification to be sent to the device via CloudScript.*



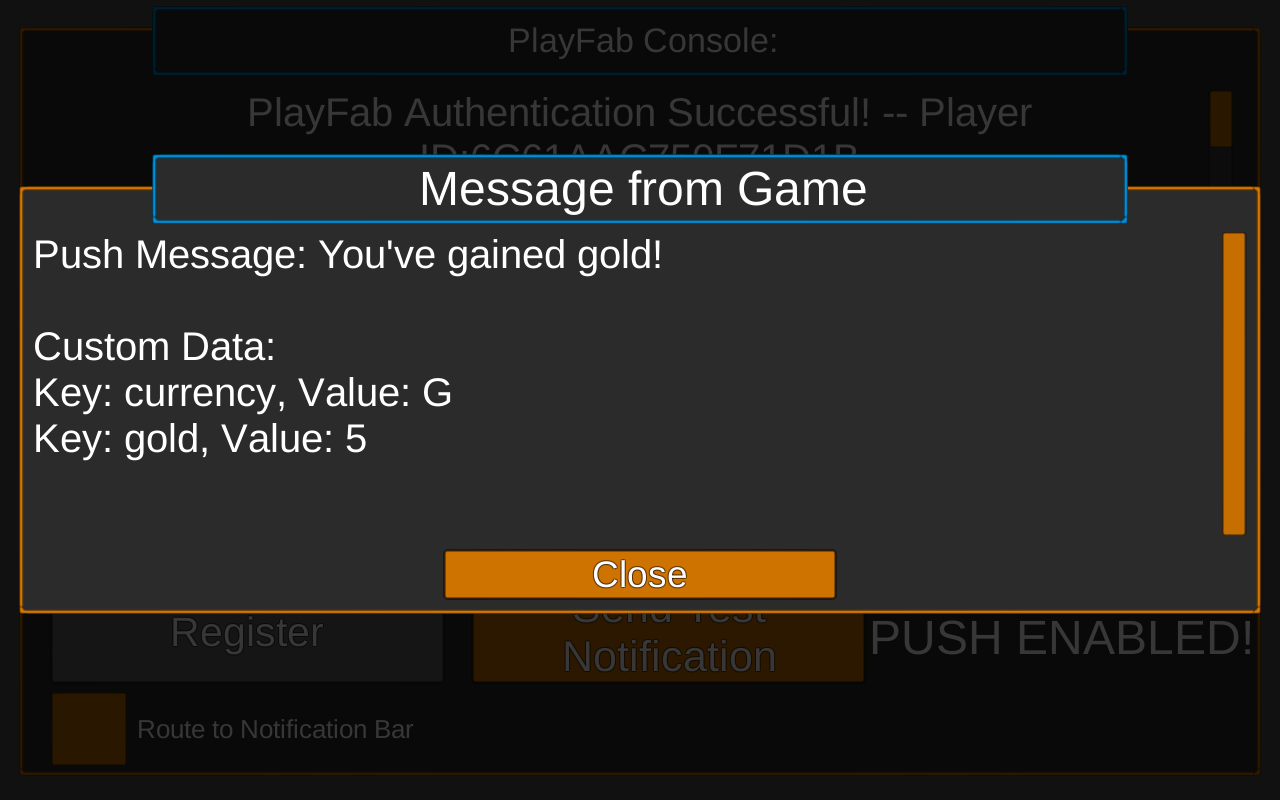
*This is the test push message. When testing with the "Route to Notification Bar" option unset, the game client will process any messages received and surface them similar to the example above.*



*When testing with the "Route to Notification Bar" option set, the game client will forward all messages to the OS's notification area.*



*Here is an example of an advanced push notification being routed to the notification area.*



*Here is an example of an advanced push notification being processed from within the game client. Take note of the custom data that is not visible when routed to the notification area.*

### Advanced Push for Android

By default, our plugin routes the push notification message to the notification area whether the game is the active app or not. To override this behavior, you will need to be subscribed to the push notification listener and update the plugin with your routing preference.

To subscribe to the event listener call:

1. PlayFabGoogleCloudMessaging.\_MessageCallback += OnPushReceived;

Update your routing preference call:

1. Enable routing to the notification area: PlayFabAndroidPlugin.UpdatePaused(true);
2. Enable routing to the game client: PlayFabAndroidPlugin.UpdatePaused(false);

### Custom Icons and Sounds

Developers can add custom icons and sounds to their push notifications by simply sending a serialized JSON object for the message parameter in [SendPushNotification](https://api.playfab.com/Documentation/Server/method/SendPushNotification). For these custom options to work, the icon and sound must be included in your resources directory (Android\Res):

1. Icons (png, jpg, etc) should be under Android\res\drawable
2. Sounds (wav, mp3, etc) should be under Android\res\raw

An example of a message using both custom icons and sounds would look like:

{

"Title": "Message from Game",

"Icon": "open\_chest",

"Sound": "raw/open",

"Message": "You've gained gold!",

"CustomData":{

"gold":"5",

"currency":"G"

}

}

JSON objects can be easily escaped using [this online tool](http://bernhardhaeussner.de/odd/json-escape/). After escaping our JSON, our API call would then look like the following:

{

"Recipient": "PLAYER\_ID\_HEX",

"Message": " {\n \"Title\": \"Message from Game\",\n \"Icon\": \"open\_chest\",\n \"Sound\": \"raw/open\",\n \"Message\": \"You've gained gold!\",\n \"CustomData\":{\n \"gold\":\"5\",\n \"currency\":\"G\"\n }\n }"

}

Upon message reception, the plugin will automatically strip and route the data accordingly.

### Custom Data

As illustrated above, PlayFab offers the ability to send arbitrary custom data along with the push notification. This custom data enables the developer to parse out Key-Value pairs and use the data accordingly for advanced interactions described at the beginning of this section. Custom data is never displayed when messages are routed to the notification area.

To access custom data, ensure that you are either:

A) Routing the notifications in game for direct parsing

-OR-

B) Retrieving the notification's custom data after your game resumes / returns from a paused state

Developers can access the custom data from the most recent push notification at any time by calling:

1. string cached = PlayFabGoogleCloudMessaging.GetPushCacheData();

This will return a JSON string that can easily be mapped to a Dictionary<string, string>

For more information on how to use advanced push notifications, see section 5 under the [readme](https://github.com/PlayFab/UnitySDK/tree/master/UnityAndroidPluginSource) appended to the plugin source.

### Advanced Push For iOS

Our advanced push plugin for iOS is currently in development. Advanced push notifications will not be possible over iOS until this plugin is complete.

# Additional Support

For help, example bugs, and related questions, drop us a line over at our [support forums](https://support.playfab.com/).

We are currently only support our services for the standard flow described in this document. If your team is looking for additional functionality with other common push services or plugins, please let us know! We love getting feedback from our developer community.

Documentation on the push payload via Amazon SNS:

* [*Amazon SNS Message & JSON Formats*](http://docs.aws.amazon.com/sns/latest/dg/json-formats.html)