



Salesforce.com: Summer '14

# Salesforce Mobile Push Notifications Implementation Guide



Last updated: May 6, 2014



# Table of Contents

<b>Mobile Push Notifications Overview.....</b>	<b>1</b>
Push Notification Registration and Flow.....	1
Check a User's Mobile Push Registrations.....	2
<b>Steps for Implementing Mobile Push Notifications.....</b>	<b>3</b>
Step 1. Developer Registration with Mobile OS Vendors.....	3
Step 2. Creating a Connected App.....	4
Create a Connected App for Android.....	4
Create a Connected App for Apple iOS.....	5
Step 3. Configure Your Mobile SDK App.....	6
Enabling Push Notifications in a Salesforce Mobile SDK Android App.....	7
Enable Push Notifications in a Salesforce Mobile SDK iOS App.....	7
Step 4. Using Apex Triggers to Send Push Notifications.....	8
<b>Reference.....</b>	<b>11</b>
PushNotification Class.....	11
PushNotification Constructors.....	12
PushNotification Methods.....	12
PushNotificationPayload Class.....	14
PushNotificationPayload Methods.....	14
Debug Log Events.....	17
<b>Index.....</b>	<b>18</b>



# Mobile Push Notifications Overview

Mobile push notifications allow Force.com mobile application developers to easily push notifications to their users' mobile devices when business events occur in the users' organizations.

Salesforce push notifications leverage the connected app framework and support customer apps built in-house.

Business events in the user's organization are trapped using Apex triggers that specify the notification message payload and recipient users. The push notifications feature hands off notifications to services operated by the device OS vendors, such as Apple or Google, for eventual delivery to user mobile devices.

## ***Push Notification Registration and Flow***

### ***Check a User's Mobile Push Registrations***

From a User page in your organization, an administrator can easily check which of the user's devices are currently registered for push notifications. Checking the registrations may help you troubleshoot push notification failures.

# Push Notification Registration and Flow

Several entities are involved when sending a push notification:

- The OS vendor that delivers the notification to devices
- The Salesforce organization that sends the notification
- The mobile devices that receive and display the notification

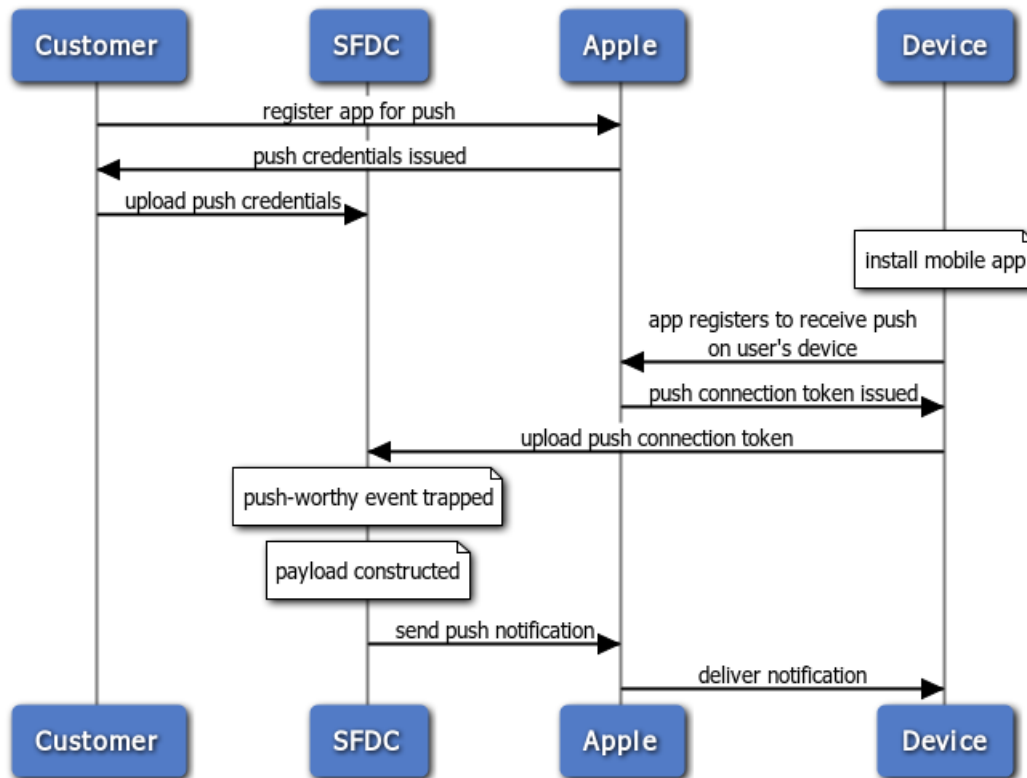
Proper registration must happen with these different entities so that the push notification can be delivered.

This is an outline of the registration process for developers.

1. Register with the mobile OS vendor, such as Apple or Google, for push service.
2. Create a connected app in Salesforce to upload the push credentials, such as the iOS .p12 certificate or the Android token.
3. Enable the mobile client app to handle push notifications using the Salesforce Mobile SDK.
4. Write Apex triggers to send push notifications when certain events occur on Salesforce records.

The following figure illustrates the complete push notification flow for customers who develop their own mobile apps and Apex triggers. The flow consists of the following sequence of events:

1. Developer registration with the OS vendor (Apple in this figure)
2. Connected app setup in Salesforce
3. Sending the push notification via the trigger from the Salesforce organization
4. Delivery of the push notification to mobile devices by the OS vendor



**Figure 1: Push Notification Flow for Customers**

Salesforce sends the message to the users specified in the `send` call of the `Apex Messaging.PushNotification` class.

## Check a User's Mobile Push Registrations

From a User page in your organization, an administrator can easily check which of the user's devices are currently registered for push notifications. Checking the registrations may help you troubleshoot push notification failures.

1. In Setup click **Manage Users** > **Users**.
2. Select a user's name from the Full Name column.
3. Under User Detail, click **View** next to Mobile Push Registrations.



**Note:** If a device you expected doesn't appear in the list, it doesn't necessarily mean that the device isn't properly configured. Mobile push registrations are volatile and depend on the device state as well as the mobile app state.

# Steps for Implementing Mobile Push Notifications

To implement mobile push notifications, you configure each of the participating technologies.

## **Step 1. Developer Registration with Mobile OS Vendors**

This step asks the OS vendor to be prepared to handle Salesforce push notifications sent to your app. Developer registration also provides some information you'll need to finish configuring your Salesforce connected app.

## **Step 2. Creating a Connected App**

Once you've registered your mobile client app with the mobile OS vendor, such as Apple or Google, for push capability, the next step is to create a connected app in Salesforce.

## **Step 3. Configure Your Mobile SDK App**

In your Mobile SDK app, implement push notification protocols required by Salesforce and the device OS provider.

## **Step 4. Using Apex Triggers to Send Push Notifications**

After registering with the mobile OS vendor for push notification service and creating a connected app, you can send push notifications to a mobile client app using Apex triggers.

## Step 1. Developer Registration with Mobile OS Vendors

This step asks the OS vendor to be prepared to handle Salesforce push notifications sent to your app. Developer registration also provides some information you'll need to finish configuring your Salesforce connected app.

To register your mobile client app, follow the process for your target mobile OS.

### **Android Registration**

When developing an Android app that supports push notifications, remember these key points:

- You must be a member of the Android Developer Program.
- You can test GCM push services only on an Android device with either the Android Market app or Google Play Services installed. Push notifications don't work on an Android emulator.

Salesforce sends push notifications to Android apps through the Google Cloud Messaging for Android (GCM) framework. See <http://developer.android.com/google/gcm/index.html> for an overview of this framework.

To begin, create a Google API project for your app. Your project must have the GCM for Android feature enabled. See <http://developer.android.com/google/gcm/gs.html> for instructions on setting up your project.

The setup process for your Google API project creates a key for your app. Once you've finished the project configuration, you'll need to add the GCM key to your connected app settings.



**Note:** Push notification registration occurs at the end of the OAuth login flow. Therefore, an app does not receive push notifications unless and until the user logs into a Salesforce org.

### **iOS Registration**

When developing an iOS app that supports push notifications, remember these key points:

- You must be a member of the iOS Developer Program.
- You can test Apple push services only on an iOS physical device. Push notifications don't work in the iOS simulator.
- There are no guarantees that all push notifications will reach the target device, even if the notification is accepted by Apple.

- Apple Push Notification Services setup requires the use of the OpenSSL command line utility provided in Mac OS X. Before you can complete registration on the Salesforce side, you need to register with Apple Push Notification Services (APNS). Registration with APNS requires the following items.

### Certificate Signing Request (CSR) File

Generate this request using the Keychain Access feature in Mac OS X. You'll also use OpenSSL to export the CSR private key to a file for later use.

### App ID from iOS Developer Program

In the iOS Developer Member Center, create an ID for your app, then use the CSR file to generate a certificate. Next, use OpenSSL to combine this certificate with the private key file to create an `appkey.p12` file. You'll need this file later to configure your connected app.

### iOS Provisioning Profile

From the iOS Developer Member Center, create a new provisioning profile using your iOS app ID and developer certificate. You then select the devices to include in the profile and download to create the provisioning profile. You can then add the profile to Xcode. Install the profile on your test device using Xcode's Organizer.

When you've completed the configuration, sign and build your app in Xcode. Check the build logs to verify that the app is using the correct provisioning profile. To view the content of your provisioning profile, run the following command at the Terminal window: `security cms -D -i <your profile>.mobileprovision`

For detailed instructions see:

- [Local and Push Notification Programming Guide at https://developer.apple.com/library/mac](https://developer.apple.com/library/mac)
- <http://www.raywenderlich.com/32960/>

## Step 2. Creating a Connected App

Once you've registered your mobile client app with the mobile OS vendor, such as Apple or Google, for push capability, the next step is to create a connected app in Salesforce.

The connected app enables you to provide the registration certificate or token of your mobile client app so that Salesforce can send push notifications. Also, the connected app provides the unique API name that identifies the mobile app to send notifications to.

### [Create a Connected App for Android](#)

### [Create a Connected App for Apple iOS](#)

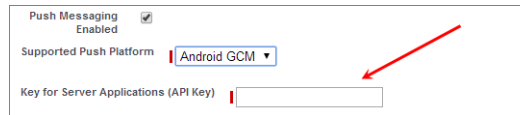
## Create a Connected App for Android

To create a connected app for Android:

1. From **Setup**, click **Create > Apps**.
2. In the Connected Apps section, click either **Edit** next to an existing connected app, or **New** to create a new connected app.
3. If you're creating a new connected app:
  - a. Enter a unique name for your connected app. The API name field is auto-filled.
  - b. Optionally, fill in other fields, such as a description for your connected app.
  - c. Enter a contact email address.
  - d. In the OAuth Settings section, select **Enable OAuth Settings**.



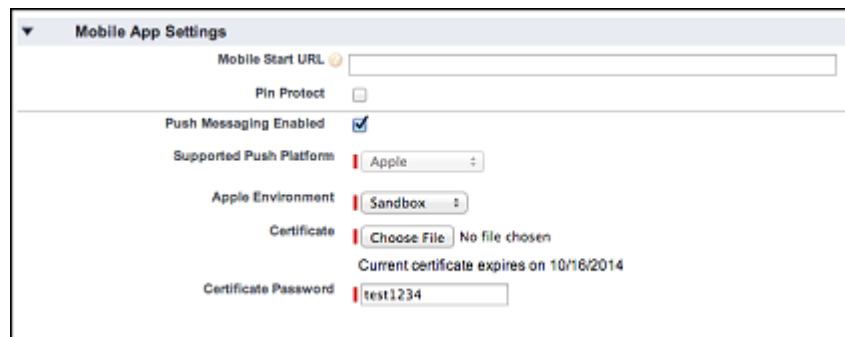
- e. Enter a callback URL.
  - f. Select OAuth scopes. At a minimum, select **Access and manage your data (api)** and **Perform requests on your behalf at any time (refresh\_token)**.
4. Under **Mobile App Settings**, select **Push Messaging Enabled**.
  5. For **Supported Push Platform**, select **Android GCM**.
  6. For **Key for Server Applications (API Key)**, enter the key you obtained during the developer registration with Google.



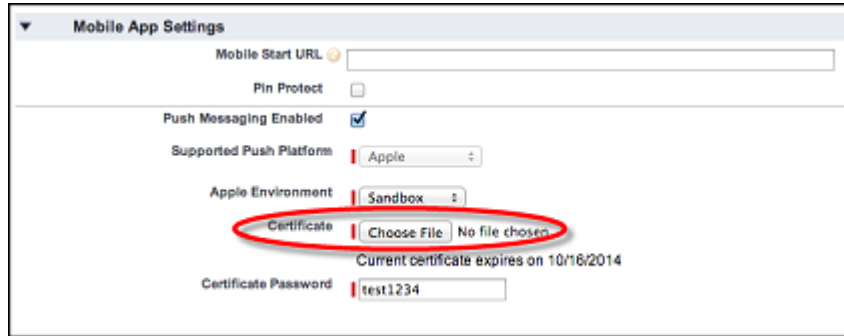
7. Click **Save**.  
After saving a new connected app, you'll get a consumer key. Mobile apps use this key as their connection token.

## Create a Connected App for Apple iOS

1. From **Setup**, click **Create > Apps**.
2. In the Connected Apps section, click either **Edit** next to an existing connected app, or **New** to create a new connected app.
3. If you're creating a new connected app:
  - a. Enter a unique name for your connected app. The API name field is auto-filled.
  - b. Optionally, fill in other fields, such as a description for your connected app.
  - c. Enter a contact email address.
  - d. In the OAuth Settings section, select **Enable OAuth Settings**.
  - e. Enter a callback URL.
  - f. Select OAuth scopes. At a minimum, select **Access and manage your data (api)** and **Perform requests on your behalf at any time (refresh\_token)**.
4. In the Mobile App Settings section, select **Mobile** and then **Push Messaging Enabled**.
5. For **Supported Push Platform**, select **Apple**.  
When you select the **Supported Push Platform** option, the dialog expands to show additional settings.



6. Add the `appkey.p12` file and its password to your connected app configuration under **Mobile App Settings > Certificate** and **Mobile App Settings > Certificate Password**.



## Step 3. Configure Your Mobile SDK App

In your Mobile SDK app, implement push notification protocols required by Salesforce and the device OS provider.

Follow the instructions for your target mobile platform (Android or iOS). Salesforce Mobile SDK does not currently support push notifications for hybrid apps.

[Enabling Push Notifications in a Salesforce Mobile SDK Android App](#)

[Enable Push Notifications in a Salesforce Mobile SDK iOS App](#)

## Enabling Push Notifications in a Salesforce Mobile SDK Android App

1. Add an entry for `androidPushNotificationClientId`.
  - a. In `res/values/bootconfig.xml` (for native apps):

```
<string name="androidPushNotificationClientId">1234567890</string>
```

- b. In `assets/www/bootconfig.json` (for hybrid apps):

```
"androidPushNotificationClientId": "1234567890"
```

This example value represents the project number of the Google project that is authorized to send push notifications to Android devices.

Behind the scenes, Mobile SDK automatically reads this value and uses it to register the device against the Salesforce connected app. This validation allows Salesforce to send notifications to the connected app. At logout, Mobile SDK also automatically unregisters the device for push notifications.

2. Create a class in your app that implements `PushNotificationInterface`. `PushNotificationInterface` is a Mobile SDK Android interface for handling push notifications. `PushNotificationInterface` has a single method, `onPushMessageReceived(Bundle message)`.

```
public interface PushNotificationInterface {
    public void onPushMessageReceived(Bundle message);
}
```

In this method you implement your custom functionality for displaying, or otherwise disposing of, push notifications.

3. In the `onCreate()` method of your `Application` subclass, call the `SalesforceSDKManager.setPushNotificationReceiver()` method, passing in your implementation of `PushNotificationInterface`. Call this method immediately after the `SalesforceSDKManager.initNative()` call.

```
@Override
public void onCreate() {
    super.onCreate();
    SalesforceSDKManager.initNative(getApplicationContext(),
        new KeyImpl(), MainActivity.class);
    SalesforceSDKManager.getInstance().
        setPushNotificationReceiver(myPushNotificationInterface);
}
```

## Enable Push Notifications in a Salesforce Mobile SDK iOS App

Salesforce Mobile SDK for iOS provides the `SFPushNotificationManager` class to handle push registration. To use it, import `<SalesforceSDKCore/SFPushNotificationManager>`. The `SFPushNotificationManager` class is available as a runtime singleton:

```
[SFPushNotificationManager sharedInstance]
```

This class implements four registration methods:

```
- (void)registerForRemoteNotifications;
- (void)didRegisterForRemoteNotificationsWithDeviceToken:
    (NSData*)deviceTokenData;
- (BOOL)registerForSalesforceNotifications; // for internal use
- (BOOL)unregisterSalesforceNotifications; // for internal use
```

Mobile SDK calls `registerForSalesforceNotifications` after login and `unregisterSalesforceNotifications` at logout. You call the other two methods from your `AppDelegate` class.

1. Register with Apple for push notifications by calling `registerForRemoteNotifications`. Place the call in the `application:didFinishLaunchingWithOptions:` method.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc]
        initWithFrame:[UIScreen mainScreen].bounds];
    [self initializeAppViewState];

    //
    // Register with APNS for push notifications. Note that,
    // to receive push notifications from Salesforce, you
    // also need to register for Salesforce notifications in the
    // application:
    // didRegisterForRemoteNotificationsWithDeviceToken:
    // method (as demonstrated below.)
    //
    [[SFPushNotificationManager sharedInstance]
        registerForRemoteNotifications];

    [[SFAuthenticationManager sharedInstance]
```

```

        loginWithCompletion:self.initialLoginSuccessBlock
                          failure:self.initialLoginFailureBlock];

    return YES;
}

```

If registration succeeds, Apple passes a device token to the `application:didRegisterForRemoteNotificationsWithDeviceToken:` method of your `AppDelegate` class.

2. Forward the device token from Apple to `SFPushNotificationManager` by calling `didRegisterForRemoteNotificationsWithDeviceToken:` on the `SFPushNotificationManager` shared instance.

```

- (void)application:(UIApplication*)application
  didRegisterForRemoteNotificationsWithDeviceToken:
    (NSData*)deviceToken
{
    //
    // Register your device token with
    // the push notification manager
    //
    [[SFPushNotificationManager sharedInstance]
     didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}

```

3. Register to receive Salesforce notifications through the connected app by calling `registerForSalesforceNotifications`. Make this call only if the access token for the current session is valid.

```

- (void)application:(UIApplication*)application
  didRegisterForRemoteNotificationsWithDeviceToken:
    (NSData*)deviceToken
{
    //
    // Register your device token with the
    // push notification manager
    [[SFPushNotificationManager sharedInstance]
     didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];

    if ([SFAccountManager sharedInstance].credentials.accessToken
        != nil){
        [[SFPushNotificationManager sharedInstance]
         registerForSalesforceNotifications];
    }
}

```

4. Add the following method to log an error if registration with Apple fails.

```

- (void)application:(UIApplication*)application
  didFailToRegisterForRemoteNotificationsWithError:
    (NSError*)error
{
    NSLog(@"Failed to get token, error: %@", error);
}

```

## Step 4. Using Apex Triggers to Send Push Notifications

After registering with the mobile OS vendor for push notification service and creating a connected app, you can send push notifications to a mobile client app using Apex triggers.

Push notification triggers use methods in the `Apex Messaging.PushNotification` and `Messaging.PushNotificationPayload` classes. The connected app in the Salesforce organization represents the mobile client app that will receive the notifications.



**Important:** To send push notifications to a connected app, Apex triggers must be added in the *same organization* in which the connected app is created.

## Sample Apex Trigger

This sample Apex trigger sends push notifications to the connected app named `Test_App`, which corresponds to a mobile app on iOS mobile clients. The trigger fires after cases have been updated and sends the push notification to two users: the case owner and the user who last modified the case.

```
trigger caseAlert on Case (after update) {
    for(Case cs : Trigger.New)
    {
        // Instantiating a notification
        Messaging.PushNotification msg =
            new Messaging.PushNotification();

        // Assembling the necessary payload parameters for Apple.
        // Apple params are:
        // (<alert text>,<alert sound>,<badge count>,
        // <free-form data>)
        // This example doesn't use badge count or free-form data.
        // The number of notifications that haven't been acted
        // upon by the intended recipient is best calculated
        // at the time of the push. This timing helps
        // ensure accuracy across multiple target devices.
        Map<String, Object> payload =
            Messaging.PushNotificationPayload.apple(
                'Case ' + cs.CaseNumber + ' status changed to: '
                + cs.Status, '', null, null);

        // Adding the assembled payload to the notification
        msg.setPayload(payload);

        // Getting recipient users
        String userId1 = cs.OwnerId;
        String userId2 = cs.LastModifiedById;

        // Adding recipient users to list
        Set<String> users = new Set<String>();
        users.add(userId1);
        users.add(userId2);

        // Sending the notification to the specified app and users.
        // Here we specify the API name of the connected app.
        msg.send('Test_App', users);
    }
}
```

## Sample Android Payload

The trigger sample uses `Messaging.PushNotificationPayload` to create a payload for an iOS notification. Unlike iOS, Android doesn't have special attributes or requirements for the payload; it just needs to be in JSON format. In Apex, you

create the Android payload as a `MAP<String,ANY>` object. The `Messaging.PushNotification` class handles conversion to JSON. Here is a sample Android payload.

```
Map<String, Object> androidPayload = new Map<String, Object>();  
androidPayload.put('number', 1);  
androidPayload.put('name', 'test');
```

# REFERENCE

## PushNotification Class

`PushNotification` is used to configure push notifications and send them from an Apex trigger.

### Namespace

Messaging

### Example

This sample Apex trigger sends push notifications to the connected app named `Test_App`, which corresponds to a mobile app on iOS mobile clients. The trigger fires after cases have been updated and sends the push notification to two users: the case owner and the user who last modified the case.

```
trigger caseAlert on Case (after update) {
    for(Case cs : Trigger.New)
    {
        // Instantiating a notification
        Messaging.PushNotification msg =
            new Messaging.PushNotification();

        // Assembling the necessary payload parameters for Apple.
        // Apple params are:
        // (<alert text>,<alert sound>,<badge count>,
        // <free-form data>)
        // This example doesn't use badge count or free-form data.
        // The number of notifications that haven't been acted
        // upon by the intended recipient is best calculated
        // at the time of the push. This timing helps
        // ensure accuracy across multiple target devices.
        Map<String, Object> payload =
            Messaging.PushNotificationPayload.apple(
                'Case ' + cs.CaseNumber + ' status changed to: '
                + cs.Status, '', null, null);

        // Adding the assembled payload to the notification
        msg.setPayload(payload);

        // Getting recipient users
        String userId1 = cs.OwnerId;
        String userId2 = cs.LastModifiedById;

        // Adding recipient users to list
        Set<String> users = new Set<String>();
        users.add(userId1);
        users.add(userId2);

        // Sending the notification to the specified app and users.
        // Here we specify the API name of the connected app.
        msg.send('Test_App', users);
    }
}
```

### [PushNotification Constructors](#)

### [PushNotification Methods](#)

## PushNotification Constructors

The following are the constructors for `PushNotification`.

### [PushNotification\(\)](#)

Creates a new instance of the `Messaging.PushNotification` class.

### [PushNotification\(Map<String, Object> payload\)](#)

Creates a new instance of the `Messaging.PushNotification` class using the specified payload parameters as key-value pairs. When you use this constructor, you don't need to call `setPayload` to set the payload.

## PushNotification()

Creates a new instance of the `Messaging.PushNotification` class.

### Signature

```
public PushNotification()
```

## PushNotification(Map<String, Object> payload)

Creates a new instance of the `Messaging.PushNotification` class using the specified payload parameters as key-value pairs. When you use this constructor, you don't need to call `setPayload` to set the payload.

### Signature

```
public PushNotification(Map<String, Object> payload)
```

### Parameters

#### *payload*

Type: `Map<String, Object>`

The payload, expressed as a map of key-value pairs.

## PushNotification Methods

The following are the methods for `PushNotification`. All are global methods.

### [send\(String application, Set<String> users\)](#)

Sends a push notification message to the specified users.

### [setPayload\(Map<String, Object> payload\)](#)

Sets the payload of the push notification message.



***setTtl(Integer ttl)***

Reserved for future use.

**send(String application, Set<String> users)**

Sends a push notification message to the specified users.

**Signature**

```
public void send(String application, Set<String> users)
```

**Parameters*****application***

Type: String

The connected app API name. This corresponds to the mobile client app the notification should be sent to.

***users***

Type: Set

A set of user IDs that correspond to the users the notification should be sent to.

**Example**

See the [Push Notification Example](#).

**setPayload(Map<String, Object> payload)**

Sets the payload of the push notification message.

**Signature**

```
public void setPayload(Map<String, Object> payload)
```

**Parameters*****payload***

Type: Map<String, Object>

The payload, expressed as a map of key-value pairs.

Payload parameters can be different for each mobile OS vendor. For more information on Apple's payload parameters, search for "Apple Push Notification Service" at <https://developer.apple.com/library/mac/documentation/>.

To create the payload for an Apple device, see the [PushNotificationPayload Class](#).

**Example**

See the [Push Notification Example](#).

## setTtl(Integer ttl)

Reserved for future use.

### Signature

```
public void setTtl(Integer ttl)
```

### Parameters

*ttl*

Type: Integer

Reserved for future use.

## PushNotificationPayload Class

Contains methods to create the notification message payload for an Apple device.

### Namespace

Messaging

### Usage

Apple has specific requirements for the notification payload, and this class has helper methods to create the payload. For more information on Apple's payload parameters, search for "Apple Push Notification Service" at <https://developer.apple.com/library/mac/documentation/>.

### Example

See the [Push Notification Example](#).

### *PushNotificationPayload Methods*

## PushNotificationPayload Methods

The following are the methods for `PushNotificationPayload`. All are global static methods.

### ***apple(String alert, String sound, Integer badgeCount, Map<String, Object> userData)***

Helper method that creates a valid Apple payload from the specified arguments.

### ***apple(String alertBody, String actionLockKey, String lockKey, String[] locArgs, String launchImage, String sound, Integer badgeCount, Map<String, Object> userData)***

Helper method that creates a valid Apple payload from the specified arguments.

## **apple(String alert, String sound, Integer badgeCount, Map<String, Object> userData)**

Helper method that creates a valid Apple payload from the specified arguments.

### Signature

```
public static Map<String, Object> apple(String alert, String sound, Integer badgeCount,
Map<String, Object> userData)
```

### Parameters

#### *alert*

Type: String

Notification message to be sent to the mobile client.

#### *sound*

Type: String

Name of a sound file to be played as an alert. This sound file should be in the mobile application bundle.

#### *badgeCount*

Type: Integer

Number to display as the badge of the application icon.

#### *userData*

Type: Map<String, Object>

Map of key-value pairs that contains any additional data used to provide context for the notification. For example, it can contain IDs of the records that caused the notification to be sent. The mobile client app can use these IDs to display these records.

### Return Value

Type: Map<String, Object>

Returns a formatted payload that includes all of the specified arguments.

### Usage

To generate a valid payload, you must provide a value for at least one of the following parameters: `alert`, `sound`, `badgeCount`.

### Example

See the [Push Notification Example](#).

## **apple(String alertBody, String actionLocKey, String locKey, String[] locArgs, String launchImage, String sound, Integer badgeCount, Map<String, Object> userData)**

Helper method that creates a valid Apple payload from the specified arguments.

### Signature

```
public static Map<String, Object> apple(String alertBody, String actionLocKey, String locKey,
String[] locArgs, String launchImage, String sound, Integer badgeCount, Map<String, Object>
userData)
```

## Parameters

### ***alertBody***

Type: String

Text of the alert message.

### ***actionLocKey***

Type: String

If a value is specified for the *actionLocKey* argument, an alert with two buttons is displayed. The value is a key to get a localized string in a `Localizable.strings` file to use for the right button's title.

### ***locKey***

Type: String

Key to an alert-message string in a `Localizable.strings` file for the current localization.

### ***locArgs***

Type: List<String>

Variable string values to appear in place of the format specifiers in *locKey*.

### ***launchImage***

Type: String

File name of an image file in the application bundle.

### ***sound***

Type: String

Name of a sound file to be played as an alert. This sound file should be in the mobile application bundle.

### ***badgeCount***

Type: Integer

Number to display as the badge of the application icon.

### ***userData***

Type: Map<String, Object>

Map of key-value pairs that contains any additional data used to provide context for the notification. For example, it can contain IDs of the records that caused the notification to be sent. The mobile client app can use these IDs to display these records.

## Return Value

Type: Map<String, Object>

Returns a formatted payload that includes all of the specified arguments.

## Usage

To generate a valid payload, you must provide a value for at least one of the following parameters: `alert`, `sound`, `badgeCount`.

## Debug Log Events

The mobile push notification service logs events that are useful to understand when you are monitoring an Apex debug log.

### Events

This table lists events that are logged, what fields or other information get logged with each event, as well as what combination of log level and category cause an event to be logged.

For more information on log levels and categories, as well as how to use the Developer Console or monitor a debug log, see the [Force.com Apex Code Developer's Guide](#).

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
PUSH_NOTIFICATION_INVALID_APP	App namespace, app name.  This event occurs when Apex code is trying to send a notification to an app that doesn't exist in the org, or is not push-enabled.	Apex Code	ERROR
PUSH_NOTIFICATION_INVALID_CERTIFICATE	App namespace, app name.  This event indicates that the certificate is invalid. For example, it's expired.	Apex Code	ERROR
PUSH_NOTIFICATION_INVALID_NOTIFICATION	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring), payload length.  This event occurs when a notification payload is too long.	Apex Code	ERROR
PUSH_NOTIFICATION_NO_DEVICES	App namespace, app name.  This event occurs when none of the users we're trying to send notifications to have devices registered.	Apex Code	DEBUG
PUSH_NOTIFICATION_NOT_ENABLED	This event occurs when push notifications are not enabled in your org.	Apex Code	INFO
PUSH_NOTIFICATION_SENT	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring)  This event records that a notification was accepted for sending. We don't guarantee delivery of the notification.	Apex Code	DEBUG

# Index

- A**
- Apex
  - debug logs [17](#)
- C**
- classes
  - PushNotification [11](#)
- Classes
  - PushNotificationPayload [14](#)
- D**
- Debug logs [17](#)
- M**
- Mobile Push Notification Service
  - check user device registrations [2](#)
  - configuring a Mobile SDK app [6](#)
  - Connected App Creation [4](#)
- Mobile Push Notification Service (*continued*)
  - Connected App Creation for Android [4](#)
  - Connected App Creation for Apple iOS [5](#)
  - developer registration with OS vendors [3](#)
  - enabling a Mobile SDK Android client app [6](#)
  - enabling a Mobile SDK iOS client app [7](#)
  - registration and Flow [1](#)
  - steps for implementing [3](#)
  - using Apex Triggers [8](#)
- mobile push notifications
  - PushNotification class [11](#)
- P**
- push notifications
  - overview [1](#)
  - PushNotification class [11](#)
- PushNotification
  - classes [11](#)
- PushNotificationPayload
  - classes [14](#)