https://www.airship.com/resources/explainer/push-notifications-explained/

What are push notifications?

A push notification is a message that pops up on a mobile device. App publishers can send them at any time; users don't have to be in the app or using their devices to receive them. They can do a lot of things; for example, they can show the latest sports scores, get a user to take an action, such as downloading a coupon, or let a user know about an event, such as a flash sale.

Push notifications look like SMS text messages and mobile alerts, but they only reach users who have installed your app. Each mobile platform has support for push notifications — iOS, Android, Fire OS, Windows and BlackBerry all have their own services.

Why are they used?

Push notifications provide convenience and value to app users. For example, users can receive:

- Sports scores and news right on their lock screen
- Utility messages like traffic, weather and ski snow reports
- Flight check in, change, and connection information

For app publishers, push notifications are a way to speak directly to a user. They don't get caught in spam filters, or forgotten in an inbox — click-through rates can be twice as high as email. They can also remind users to use an app, whether the app is open or not. They can also be used to drive actions, such as:

- Promoting products or offers to increase sales
- Improving customer experience
- Converting unknown app users to known customers
- Sending transactional receipts right away
- Driving users to other marketing channels, such as social networks

History

June 2009: Apple launches Apple Push Notification Service (APNs), the first push service.

May 2010: Google released its own service, Google Cloud to Device Messaging (C2DM).

May 2013: Google introduces "rich notifications". Rich notifications can contain images, as well as action buttons. Action buttons let users take immediate action from a notification. For example, the user can play a song, open the app, or see more information.

September 2014: Apple added interactive buttons. These buttons allow users to send a response right away to the app publisher. Shortly after, Apple extended push notifications to the Apple Watch.

How do push notifications work?

Some of the actors in sending a push notification include:

1. **Operating system push notification service** (OSPNS). Each mobile operating system (OS), including iOS, Android, Fire OS, Windows, and BlackBerry, has its own service.
2. **App publisher**. The app publisher enables their app with an OSPNS. Then, the publisher uploads the app to the app store.
3. **Client app**. This is an OS-specific app, installed on a user's device. It receives incoming notifications.

Adding to an app

1. The app publisher registers with the OS push notification service.
2. The OS service provides an application programming interface (API) to the app publisher. The API is a way for the app to communicate with the service.
3. The app publisher adds the SDK to the app. The SDK is a code library specific to the OS' push notification service.
4. The app publisher uploads the app to the app store.

User activation

1. The user visits an OS app store, downloads and then installs an app.
2. The user opens the app. Unique identifiers (IDs) for both the app, and the device, are registered with the OS push notification service.
3. The unique identifiers are passed back to the app from the OS push notification service. They are also sent to the app publisher.
4. The app publisher receives and stores these registration details, including the unique identifiers.

Sending

1. The app publisher composes a manual message through a message composer user interface. Or, the publisher sets up an automated message to be sent via the API.
2. The publisher defines the audience to whom the push notification will be sent.
3. The publisher determines whether the message should be sent immediately or scheduled.

Push notifications can be targeted to segments of your user base, and even personalized for specific app users. This is a major advantage when compared to SMS text messaging. However, they also require the management of user identification data. And they need some kind of interface for writing messages, targeting them and sending them.

Publishers can build this infrastructure themselves, or they can hire a vendor such as Urban Airship to provide it. Increasingly, app publishers pay for these services instead of building them, so that

they can focus on building a great app — building and maintaining a cross-platform push notification service takes significant resources and ongoing maintenance. Platform vendors also provide capabilities such as:

- Reporting
- Scheduling
- Mobile marketing automation
- User attribute collection and segmentation
- Data management
- Security
- Cross-platform support

Opting in

iOS apps require a user to grant permission for an app to send them push notifications, while Android and Fire OS do not. Convincing users to opt-in is important for the success of apps on iOS.

The majority of iOS apps show a standard iOS alert when the app is first opened. A better approach is to show the value of opting in — for example, with a customized welcome series upon first open — then let the user opt-in later. Median opt-in rates for iOS range from 58% for charity apps to 33% for games.

High performing apps across all industry verticals (those in the 90th percentile) have opt-in rates above 50%. Travel, business and charity app opt-in rates lead verticals with rates greater than 70%.

How do push notifications appear to users?

Mainly, users see a notification as a banner or pop-up alert as they are using their phone. This alert is shown no matter what the user is doing.

Most mobile operating systems also show push notifications together in a single view. On iOS, Apple has a Notification Center. The Notification Center is organized in chronological order, and users get to the Notification Center by swiping down from the top of the screen. Android devices show unread messages on the lock screen.

iOS lets users customize push notifications at an individual app level. Users can turn sounds on or off, and pick the style that iOS uses to show a notification. Users can also control the red “badge” showing the number of unread notifications on an app’s homescreen icon. Android uses a standard banner approach that users cannot change at an OS level.

Using location with push notifications

All mobile operating systems ask users for their permission to share location information. iOS presents an opt-in alert to users. Android offers location opt-in as part of the app’s permissions setup during installation.

Publishers can deliver more relevant messages by using location data combined with behavioral data. Examples include:

- A home improvement app sends offers for cooling units during a regional hot spell.
- A specialty boutique invites users within 50 miles of an invite-only VIP trunk sale.
- A national sporting goods chain invites local shoppers in for local pro athlete autographs.

Strategy

Push notifications are a direct path of communication with users. App publishers should treat the ability to communication with users via push notifications as a privilege, not a right. App publishers must provide value; if they don't, push notifications will be ignored or turned off. Some users will uninstall the app altogether.

Analytics and measurement are important tools for improving your app's performance. But it's important to write compelling push notifications that are valuable to users and that drive action.

Messaging strategies and tactics need to be measured and tested. Strategies such as maximizing opt-in rates, ensuring new users are properly onboarded and reducing app user churn rates are all key to an app's success. Other strategies include:

- Matching up each user's data across channels (web, mobile, store etc.) to better understand user behavior
- Making it easy for users to share content with their social networks
- Encouraging users to opt in by offering them incentives and examples of the value your notifications will provide
- Optimize the app experience to keep customers engaged

Want to learn more about how Push Notifications can help you connect with customers at each stage of the customer lifecycle? Contact us today and let's talk!

https://docs.repro.io/en/dev/sdk/push-notification/android.html#id6

Push Notification for Android

Settings for Push Notification

See Settings for FCM (Android) to setup.

Add Firebase SDK

Copy `google-services.json` which you have downloaded from Settings for FCM (Android) to the app directory, and add the lines below to the project level build.gradle file.

```
buildscript {
    ...
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
    }
}

allprojects {
    ...
    repositories {
        ...
        google()
    }
}
```

Add the following to your application's build.gradle.

```
dependencies {
    implementation 'com.google.firebase:firebase-core:16.0.4'
    implementation 'com.google.firebase:firebase-messaging:17.3.4'
}

apply plugin: 'com.google.gms.google-services'
```

Warning

Please specify version 17.1.0 or above for firebase-messaging. Repro SDK cannot retrieve Registration ID with versions below 17.1.0.

Note

'implementation' is not available for versions below Gradle 3.4. Please use 'compile' instead.

Edit `AndroidManifest.xml`

Register receiver

Replace "YOUR_PACKAGE_NAME" with your application's package name in the following snippet, and add it to your AndroidManifest.xml inside of `<application>` tag.

```
<receiver
    android:name="io.repro.android.ReproReceiver"
    android:exported="true"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="YOUR_PACKAGE_NAME" />
    </intent-filter>
</receiver>
```

Setup Notification Channels

Since Android O, Notification Channels were introduced to help users manage their notifications.

To set the notification channels in Repro SDK, please specify the following: ID, channel name, description for users, and with or without badge display.Replace the `value`, `resource` within the XML below with the desired value, and add the `<application>` tag into the AndroidManifest.xml.

Warning

- If the targetSDKVersion of your application is greater than or equal to 26, please note that channel id and name are required or the SDK won't show notificaitons sent from Repro while running on devices with Android O or later.

```
<meta-data
    android:name="io.repro.android.PushNotification.ChannelId"
    android:value="YOUR_CHANNEL_ID">
</meta-data>

<meta-data
    android:name="io.repro.android.PushNotification.ChannelName"
    android:resource="@string/YOUR_CHANNEL_NAME">
</meta-data>

<meta-data
    android:name="io.repro.android.PushNotification.ChannelDescription"
    android:resource="@string/YOUR_CHANNEL_DESCRIPTION">
</meta-data>

<meta-data
    android:name="io.repro.android.PushNotification.ShowBadge"
    android:value="true">
</meta-data>
```

If the channel of the specified id does not exist, it will be created automatically by the SDK based on the information provided in AndroidManifest.xml. If there is an existing channel with the specified id, the SDK will use it.

The SDK will update the channel name and description of the existing channel being used by the SDK.

The SDK will ignore the settings above when the targetSDKVersion of your application is 25 or below, or when the application is running on devices with versions earlier than Android O.

Note

Notification Channels were introduced as a standard feature in Android O. For the details, please refer to this page.

Customize Icon and Background Color

When customizing the icon and the background color displayed in the notification area for devices with Android 5.0 or above, add the XML construct below inside of the `<application>` tag in your AndroidManifest.xml. This setting will be ignored with devices under Android 5.0, and the notification area will use the application's icon as well as the system's default background color.

```
<meta-data
    android:name="io.repro.android.PushNotification.SmallIcon"
    android:resource="@drawable/YOUR_ICON_ID">
</meta-data>

<meta-data
    android:name="io.repro.android.PushNotification.AccentColor"
    android:resource="@color/YOUR_COLOR_ID">
</meta-data>
```

Send Registration ID to Repro

Right after calling `setup()`, call `enablePushNotification()`.

```
import io.repro.android.Repro;

public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();

        ...

        Repro.setup(this, "YOUR_APP_TOKEN");
        Repro.enablePushNotification();

        ...
    }
}
```

Call `Repro.setPushRegistrationID` in the method `onNewToken` inherited from `FirebaseMessagingService`.

```
import io.repro.android.Repro;

public class MyFirebaseMessagingService extends FirebaseMessagingService {
    ...
    @Override
    public void onNewToken(String token) {
```

```
            Repro.setPushRegistrationID(token);
        }
        ...
    }
```

Add the following snippet to your AndroidManifest.xml inside of `<application>` tag.

```
<service
    android:name=".MyFirebaseMessagingService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

Note

Adding multiple FirebaseMessagingService to AndroidManifest.xml will result in only one of the FirebaseMessagingServices actually working.

Therefore, if you have already implemented a class inherting FirebaseMessagingService, please call `Repro.setPushRegistrationID` in the `onNewToken` of the existing class, rather than creating a new class inherting FirebaseMessagingService.

After finishing the above implementation, see Create Message.

Option: customize the behavior when receiving messages

If you want to customize the behavior when receiving push notifications, please implement the onMessageReceived method of FirebaseMessagingService.

Implement onMessageReceived of FirebaseMessagingService

Override `onMessageReceived` and call the following three methods.

- Call `Repro.applicationShouldHandlePushNotification` and decide whether to use the received message in the application or not. If this method returns `false`, the message will be processed by the SDK.
- Call `Repro.isAlreadyReceivedPushNotification` and detect the message has already been received or not. The device may receive duplicated messages if the application is being uninstalled and re-installed repeatedly.
- Call `Repro.markPushNotificationReceived` and mark the message received.

Note

`ReproReceiver` is required to receive standard format messages. Do not delete `ReproReceiver`.

```
@Override
public void onMessageReceived(RemoteMessage message) {
    Map<String, String> data = message.getData();

    // check whether the application should handle this push notification.
    if (!Repro.applicationShouldHandlePushNotification(this, data)) {
```

```
        Log.d(TAG, "Ignore push notification: it will be handled by SDK: " +
data.toString());
        return;
    }

    // check whether this push notification is already received.
    if (Repro.isAlreadyReceivedPushNotification(this, data)) {
        Log.d(TAG, "Ignore push notification: it is already received: " +
data.toString());
        return;
    }

    // mark this push notification as "received".
    Log.d(TAG, "Mark push notification as received: " + data.toString());
    Repro.markPushNotificationReceived(this, data);

    // Implement procedures unique to the application here
    ...
}
```

Track notification as opened

Standard format message opening will be tracked automatically by the Repro SDK. For JSON formatted messages, you will need to call `Repro.trackNotificationOpened` to track message opens.

The below is a sample of creating notification that launches Activity from the JSON format message, and call `Repro.trackNotificationOpened` within the called Activity.

```
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "MessagingService";
    private static final String CHANNEL_ID = "custom-firebase-messaging-service-
channel";

    private final Random mRandom = new Random();

    @Override
    public void onNewToken(String token) {
        Repro.setPushRegistrationID(token);
    }

    @Override
    public void onMessageReceived(RemoteMessage message) {
        final int identifier = mRandom.nextInt();
        Map<String, String> data = message.getData();

        // check whether the application should handle this push notification.
        if (!Repro.applicationShouldHandlePushNotification(this, data)) {
            Log.d(TAG, "Ignore push notification: it will be handled by SDK: " +
data.toString());
            return;
        }

        // check whether this push notification is already received.
        if (Repro.isAlreadyReceivedPushNotification(this, data)) {
            Log.d(TAG, "Ignore push notification: it is already received: " +
data.toString());
            return;
        }
```

```
        // mark this push notification as "received".
        Log.d(TAG, "Mark push notification as received: " + data.toString());
        Repro.markPushNotificationReceived(this, data);

        String messageText;
        if (data.containsKey("message")) {
            messageText = data.get("message");
        } else {
            messageText = "default message";
        }

        final Notification.Builder builder = new Notification.Builder(this)

.setContentTitle(this.getResources().getString(R.string.app_name))
                .setContentText(messageText)
                .setSmallIcon(R.mipmap.ic_launcher_round)
                .setAutoCancel(true);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            createOrUpdateChannel();
            builder.setChannelId(CHANNEL_ID);
        }

        Intent resultIntent = new Intent(this, SomeActivity.class);

        String notificationIdKey =
data.get(io.repro.android.ReproReceiver.NOTIFICATION_ID_KEY);
        if (notificationIdKey != null) {

resultIntent.putExtra(io.repro.android.ReproReceiver.NOTIFICATION_ID_KEY,
notificationIdKey);
        }

        PendingIntent resultPendingIntent = PendingIntent.getActivity(this,
identifier, resultIntent, PendingIntent.FLAG_CANCEL_CURRENT);
        builder.setContentIntent(resultPendingIntent);

        final NotificationManager notificationManager =
(NotificationManager)this.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(identifier, builder.getNotification());
    }

    @TargetApi(Build.VERSION_CODES.O)
    private void createOrUpdateChannel() {
        NotificationChannel newChannel = new NotificationChannel(CHANNEL_ID,
"Custom Channel", NotificationManager.IMPORTANCE_DEFAULT);
        newChannel.setDescription("");
        newChannel.setShowBadge(true);

        // create or update the Notification channel
        final NotificationManager notificationManager =
(NotificationManager)this.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.createNotificationChannel(newChannel);
    }
}
```

Within the launched Activity, set the push notification ID from the Extras of the previous `resultIntent` to `Repro.trackNotificationOpened`. If the launch mode of the activity is `singleTop`, set the push notification the same way with `onNewIntent`.

```
public class SomeActivity extends Activity {

  @Override
  protected void onResume() {
      super.onResume();

      Intent intent = getIntent();
      Bundle extras = intent.getExtras();
      if (extras != null &&
extras.containsKey(io.repro.android.ReproReceiver.NOTIFICATION_ID_KEY)) {

Repro.trackNotificationOpened(extras.getString(io.repro.android.ReproReceiver.NOT
IFICATION_ID_KEY));
      }
  }

  @Override
  protected void onNewIntent(Intent intent) {
      super.onNewIntent(intent);

      Bundle extras = intent.getExtras();
      if (extras != null &&
extras.containsKey(io.repro.android.ReproReceiver.NOTIFICATION_ID_KEY)) {

Repro.trackNotificationOpened(extras.getString(io.repro.android.ReproReceiver.NOT
IFICATION_ID_KEY));
      }

      this.setIntent(intent);
  }
```

Note

- The push notification ID can be retrieved from the Bundle of the received message with the `io.repro.android.ReproReceiver.NOTIFICATION_ID_KEY` key.

- « Push Notification for iOS
- Push Notification (Unity) »