

---

# **QGDocs Documentation**

***Release 1.0***

**QuantumGraph Engineers**

**Aug 28, 2018**



<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Why QGraph? . . . . .	3
1.2	How do I get started? . . . . .	3
<b>2</b>	<b>Basics</b>	<b>5</b>
2.1	How does it work? . . . . .	5
2.2	User Profiles and Events . . . . .	5
2.2.1	User Profiles . . . . .	5
2.2.2	Events . . . . .	6
<b>3</b>	<b>Android SDK integration</b>	<b>7</b>
3.1	Installation in Android Studio . . . . .	7
3.1.1	A. If you use FCM . . . . .	7
3.1.1.1	Note . . . . .	8
3.1.2	B. If you use GCM . . . . .	8
3.2	Installation in Cordova . . . . .	9
3.3	Using Android SDK . . . . .	9
3.3.1	Import QG SDK in your activity . . . . .	9
3.3.2	Initialization of SDK . . . . .	9
3.3.3	Logging user profiles . . . . .	10
3.3.4	Logging events . . . . .	11
3.3.5	Retrieving stored notifications . . . . .	15
3.3.6	Configuring Batching . . . . .	15
3.3.7	InApp Notifications . . . . .	15
3.3.8	Event Attribution . . . . .	15
3.4	Notification checklist . . . . .	17
3.4.1	Launcher image . . . . .	17
3.4.2	Notification image . . . . .	17
3.4.3	Recommended sizes of images . . . . .	17
3.4.4	If you use your own Service to extend GCMListenerService . . . . .	17
3.4.5	Receiving key value pairs in activity . . . . .	18
<b>4</b>	<b>iOS SDK integration</b>	<b>19</b>
4.1	Installing iOS SDK Library . . . . .	19
4.1.1	Using Cocoapods . . . . .	19
4.1.2	Manual installation . . . . .	20
4.2	Generating .p12 file . . . . .	20

4.2.1	Generating SSL Certificate for APP ID . . . . .	20
4.2.2	Generating the Certificate Signing Request . . . . .	21
4.3	Using iOS SDK - Objective C . . . . .	22
4.3.1	Change required for APNS Token and User Tracking . . . . .	22
4.3.2	AppDelegate Changes . . . . .	23
4.3.3	Handling Push Notification . . . . .	24
4.3.4	Changes for iOS 10 . . . . .	25
4.3.5	AppDelegate Changes for iOS 10 . . . . .	25
4.3.6	Handling Push Notification in iOS 10 . . . . .	27
4.3.7	Handling Deeplink for QGraph Push . . . . .	29
4.3.8	Adding Extensions for iOS Push with Attachment and QGraph Carousel and Slider Push . . . . .	29
4.3.9	Notification Service Extension . . . . .	30
4.3.9.1	Adding Service extension . . . . .	30
4.3.9.2	Adding Content Extension . . . . .	31
4.3.10	Click Through and View Through Attribution . . . . .	31
4.3.11	Configuring Batching . . . . .	33
4.3.12	Matching mobile app users with mobile web users . . . . .	33
4.3.13	In app Notification . . . . .	33
4.3.14	Registering Your Actionable Notification Types . . . . .	34
4.3.15	Logging user profile information . . . . .	34
4.3.16	Logging events information . . . . .	35
4.4	Using iOS SDK - Swift (3.0) . . . . .	38
4.4.1	Change required for APNS Token and User Tracking . . . . .	38
4.4.2	Adding bridging headers . . . . .	38
4.4.3	App Delegate Changes . . . . .	38
4.4.4	Handling Push Notification . . . . .	40
4.4.5	Changes for iOS 10 . . . . .	41
4.4.6	AppDelegate Changes for Swift Apps for iOS 10 . . . . .	41
4.4.7	Handling Push Notification in iOS 10 . . . . .	43
4.4.8	Handling Deeplink for QGraph Push . . . . .	44
4.4.9	Adding Extensions for iOS Push with Attachment and QGraph Carousel and Slider Push . . . . .	46
4.4.10	Notification Service Extension . . . . .	46
4.4.11	Adding Service extension . . . . .	46
4.4.12	Adding Content Extension . . . . .	48
4.4.13	Click Through and View Through Attribution . . . . .	50
4.4.14	Configuring Batching . . . . .	50
4.4.15	Matching mobile app users with mobile web users . . . . .	50
4.4.16	In app Notification . . . . .	51
4.4.17	Registering Your Actionable Notification Types . . . . .	51
4.4.18	Logging user profile information . . . . .	51
4.4.19	Logging events information . . . . .	52
4.4.19.1	Registration Completed . . . . .	53
4.4.19.2	Category Viewed . . . . .	53
4.4.19.3	Product Viewed . . . . .	53
4.4.19.4	Product Added to Wishlist . . . . .	53
4.4.19.5	Product Purchased . . . . .	53
4.4.19.6	Checkout Initiated . . . . .	54
4.4.19.7	Product Rated . . . . .	54
4.4.19.8	Searched . . . . .	54
4.4.19.9	Reached Level . . . . .	54
4.4.19.10	Your custom events . . . . .	55
<b>5</b>	<b>Web SDK integration</b>	<b>57</b>
5.1	Background and Terminology . . . . .	57

5.2	Installing Web Pixel . . . . .	58
5.2.1	If your site is HTTPS . . . . .	60
5.2.2	If your site is HTTP . . . . .	61
5.3	Logging Data . . . . .	62
5.3.1	Logging profile information . . . . .	62
5.3.2	Logging event information . . . . .	63
<b>6</b>	<b>Advanced Integration Topics</b>	<b>65</b>
6.1	Passing Dates and Times to QGraph Servers . . . . .	65
6.1.1	Format for Date . . . . .	65
6.1.2	Format for Time . . . . .	65
6.1.3	Format for Datetime . . . . .	65
6.2	Passing data to QGraph from your servers . . . . .	66
<b>7</b>	<b>Using Web UI</b>	<b>69</b>
7.1	1. Recent Users . . . . .	69
7.2	2. Recent Activities . . . . .	70
7.3	3. Segments . . . . .	70
7.4	4. Campaigns . . . . .	70
<b>8</b>	<b>Using API</b>	<b>71</b>
8.1	Sending notifications . . . . .	71
8.1.1	Specifying key value pairs . . . . .	74
8.2	Getting user profiles . . . . .	74
8.2.1	Specifying start and end dates . . . . .	74
8.2.2	Specifying OS . . . . .	74
8.2.3	Specifying specific fields to retrieve . . . . .	75
8.3	Create a user uploaded segment . . . . .	75
<b>9</b>	<b>Downloads</b>	<b>77</b>
9.1	Android . . . . .	77
9.2	iOS . . . . .	77



Contents:





# CHAPTER 1

---

## Introduction

---

QGraph is the User Engagement Platform for your mobile app and websites.

If you are an app or website developer you know that keeping your users engaged with your app or website is very important. Push (App push or web push), email and SMS are most important channels for app engagement.

Using QGraph you can send relevant, timely and personalized push notifications, emails and SMS to your users.

### 1.1 Why QGraph?

QGraph is the easiest way to send notifications to your app users. When you use QGraph, you do not need to worry about intricacies and mechanism of sending notifications, which vary widely across different platforms. These intricacies are all abstracted out in our SDK.

We have made the process of sending notifications as simple as writing a “Hello World!” program. From registering on our site to sending a test notification should take less than 15 minutes.

We provide the capabilities for fine grained segmentation and message customization at per user level. We provide detailed statistics about the response to the notifications.

### 1.2 How do I get started?

Head over to <http://qgraph.io/> and make an account for yourself.

Follow the setup steps given there and send a test notification to one or more devices.

Once you are done, go to the section “Using QGraph SDK” to do a complete integration with your app.

You can use our web app at <http://app.qgraph.io/> to manage segments and campaigns.



### 2.1 How does it work?

Here is a thousand feet overview of how QGraph SDK sends notifications to your app users.

1. You register at our site integrate the SDK in your app. SDK provides you with some functions that you can call to send us data related to your app users.
2. You send us user data by calling the functions of the SDK. There are two types of data: user profile data, like the name of user, gender of user, city of user etc) and event (activity) data, like a user viewing a product, a user purchasing a product, etc.
3. You go to our web panel at <http://app.qgraph.io>. You create one more segments. A segment is a set of users. For instance you can create a segment of the users who reside in Bangalore and have not opened your app in last 7 days. You also create a campaign. A campaign is a segment together with a creative (i.e. the title, the message, the image etc of the notification). Once you have created a campaign, you send the notification.
4. After you send the notification, you can go to respective campaign and view statistics around how many of those notifications were opened, and what events happend as a result of notifications.

### 2.2 User Profiles and Events

Nextly you need to know about user profiles and events.

#### 2.2.1 User Profiles

User profile is information regarding attributes of the user: for instance his name, email, city, gender and so on. User profile may contain information that is specific to your app, for example, the maximum level attained in a gaming app, total lifetime purchase made by a user, or his interests. Each user profile item has a “key” and a “value”. For instance for the name of a person, the key is “name” and value might be “John Appleseed”.

## 2.2.2 Events

Events are the activities that a user performs, for instance viewing a product, purchasing a product, playing a game or liking an item. Each event has a name, say “product\_viewed”, or “product\_purchased”. Each event also has some parameters which consist of “keys” and “values”. For instance, for the event “product\_viewed”, the parameter keys would be “id”, “name”, “img\_url”, “deep\_link” etc with sample values 123, “Nikon Camera”, “<http://mysite/product/123.png>” and “myapp://myapp/product/123” respectively.

## 3.1 Installation in Android Studio

### 3.1.1 A. If you use FCM

1. Add dependencies to *app/build.gradle*:

```
compile "com.quantumgraph.sdk:QG:5.2.0"  
compile 'com.google.firebase:firebase-messaging:11.2.2'
```

2. If you have implemented `FirebaseMessagingService` in your project add the following code inside `onMessageReceived(RemoteMessage remoteMessage)` method:

```
String from = remoteMessage.getFrom();  
Map data = remoteMessage.getData();  
if (data.containsKey("message") && QG.isQGMessage(data.get("message").  
    toString())) {  
    Bundle qqData = new Bundle();  
    qqData.putString("message", data.get("message").toString());  
    Context context = getApplicationContext();  
    if (from == null || context == null) {  
        return;  
    }  
    Intent intent = new Intent(context, NotificationJobIntentService.class);  
    intent.setAction("QG");  
    intent.putExtras(qqData);  
    JobIntentService.enqueueWork(context, NotificationJobIntentService.class,   
    1000, intent);  
    return;  
}
```

3. If you have implemented `FirebaseInstanceIdService`, add the following code inside `onTokenRefresh()`:

```
QG.logFcmId(getApplicationContext());
```

4. Additional settings:

- If you would like to reach out to uninstalled users by email, add following line in *app/src/main/AndroidManifest.xml* outside the `<application>` tag:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

- If you would like us to track the city of the user, add the following line in *app/src/main/AndroidManifest.xml* outside the `<application>` tag:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- If you would like us to track device id the user, add the following line in *app/src/main/AndroidManifest.xml* outside the `<application>` tag:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

### 3.1.1.1 Note

If while building your project, you get a `ClassNotFoundException`, check the following

1. Check that you are using Support Library version 26 or above

```
compile 'com.android.support:appcompat-v7:26.0.1'
```

2. Check that you have included maven properly in *project/build.gradle*

```
allprojects {
    repositories {
        jcenter()
        maven {
            url "https://maven.google.com"
        }
    }
}
```

### 3.1.2 B. If you use GCM

We prefer that you integrate using FCM. However, if you are already using GCM (and have GCM tokens), follow the following steps

1. Add dependencies to *app/build.gradle*:

```
compile "com.quantumgraph.sdk:QG:2.3.5"
compile "com.google.android.gms:play-services-gcm:11.2.2"
```

2. Additional settings:

- If you would like to reach out to uninstalled users by email, add following line in *app/src/main/AndroidManifest.xml* outside the `<application>` tag:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

- If you would like us to track the city of the user, add the following line in *app/src/main/AndroidManifest.xml* outside the `<application>` tag:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- If you would like us to track device id the user, add the following line in *app/src/main/AndroidManifest.xml* outside the `<application>` tag:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

3. If you use your own service that extends `GCMListenerService`, following code snippet must be added in your service:

```
@Override
public void onMessageReceived(String from, Bundle data) {
    if (data.containsKey("message") && QG.isQGMessage(data.getString("message")))
    ↪ {
        Context context = getApplicationContext();
        Intent intent = new Intent(context, NotificationJobIntentService.class);
        intent.setAction("QG");
        intent.putExtras(data);
        JobIntentService.enqueueWork(context, NotificationJobIntentService.class,
    ↪ 1000, intent);
        return;
    }
}
```

## 3.2 Installation in Cordova

QGraph supports apps built with Cordova. Please look at our github plugin for cordova [here](#).

## 3.3 Using Android SDK

Follow these steps to use our android SDK

### 3.3.1 Import QG SDK in your activity

In all the activity classes, starting with the class for the main activity, import QG SDK:

```
import com.quantumgraph.sdk.QG;
```

### 3.3.2 Initialization of SDK

1. Define a variable called `qg` in your activity:

```
private QG qg;
```

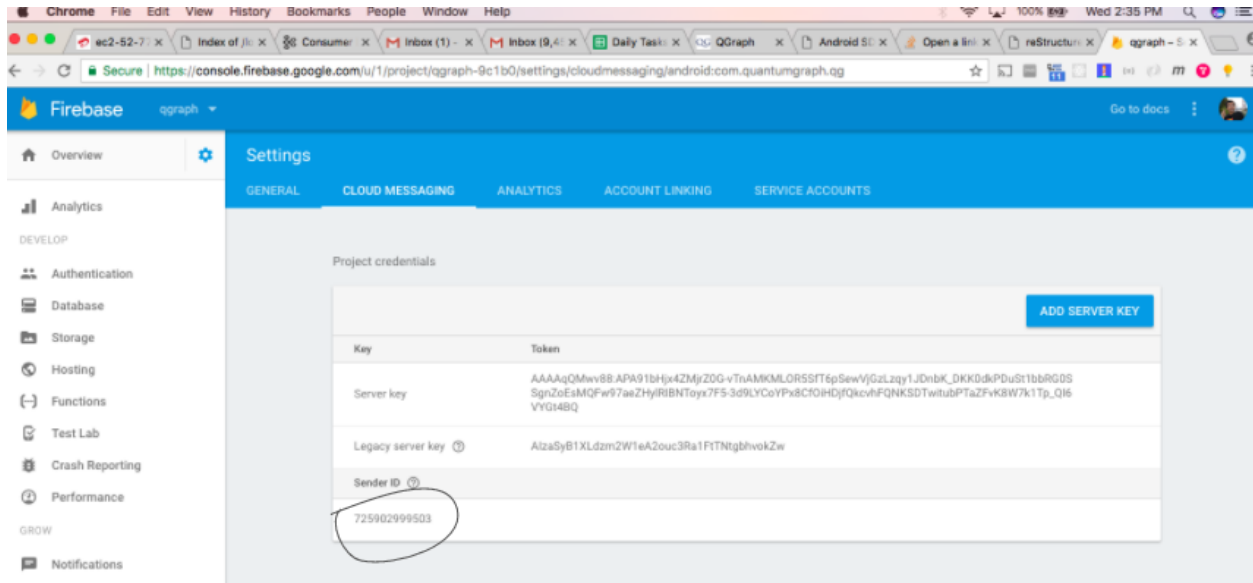
2. Add a line in `onCreate()` of your activity. If you do not use Firebase in your app, add the following:

```
QG.initializeSdk(getApplication(), <your app id>);
```

If you use Firebase in your app, you need to know your sender id. In that case, add the following:

```
QG.initializeSdk(getApplication(), <your app id>, <your sender id>);
```

App id for your app is available from the settings page of our webapp. To get your sender id, go to your project settings in <https://console.firebase.google.com>. (You need to access “Cloud Messaging” tab in Firebase console).



3. In the `onStart()` function of your activity, add the following:

```
qq = QG.getInstance(getApplicationContext());
qq.onStart();
```

4. In case you want to enable GA integration via our SDK, add the following in your main activity's `onCreate()`:

```
qq.enableGATrackingWithGAID("<your GA id>");
```

Replace `<your GA id>` above **with** the GA id **for** your project.

### 3.3.3 Logging user profiles

User profiles are information about your users, like their name, city, date of birth or any other information that you may wish to track. You log user profiles by using one or more of the following functions:

```
qq.setUserId(String userId);
```

`userId` is the id of the user. It might be email, or username, or facebook id, or any other form of id that you may wish to keep.

Other functions that you may use are:



```

qg.setName(String name);
qg.setFirstName(String firstName);
qg.setLastName(String lastName);
qg.setCity(String city);
qg.setEmail(String email);
qg.setDayOfBirth(int day);
qg.setMonthOfBirth(int month);
qg.setYearOfBirth(int year);
qg.setPhoneNumber(String phoneNo);

```

Other than these functions, you can log your own custom user parameters. You do it using:

```
qg.setCustomUserParameter(String key, E value);
```

For instance, you may wish to have the user's current rating like this:

```
qg.setCustomUserParameter("current_rating", 123);
```

As implied by the function definition, the value can be of any data type.

Once user profile is set, you can use this to create personalized messages (For example: “Hi John, exciting deals are available in Mountain View”), or to create user segments (For example you can create a segment of users who were born after 1990 and live in Mountain View)

Apart from above user profile parameters, you can log the UTM source through which the user installed the app, using the following functions:

```

qg.setUtmSource(String utmSource);
qg.setUtmMedium(String utmMedium);
qg.setUtmTerm(String utmTerm);
qg.setUtmContent(String utmContent);
qg.setUtmCampaign(String utmCampaign);

```

### 3.3.4 Logging events

Events are the activities that a user performs in your app, for example, viewing the products, playing a game or listening to a music. Each event has a name (for instance, the event of viewing a product can be called `product_viewed`), and can have some parameters. For instance, for event `product_viewed`, the parameters can be `id` (the id of the product viewed), `name` (name of the product viewed), `image_url` (image url of the product viewed), `deep_link` (a deep link which takes one to the product page in the app), and so on.

Once you log event information to use, you can segment users on the basis of the events (For example, you can create a segment consisting of users have not launched for past 7 days, or you can create a segment consisting of users who, in last 7 days, have purchased a product whose value is more than \$1000)

You can also define your events, and your own parameters for any event. However, if you do that, you will need to sync up with us to be able to segment the users on the basis of these events or customize your creatives based on these events.

You can optionally log a “value to sum” with an event. This value will be summed up when doing campaign attribution. For instance, if you pass this value in your checkout completed event, you will be able to view stats such as a particular campaign has been responsible to drive Rs 84,000 worth of sales. You can also optionally provide a currency code for the value to sum. Currency needs to be a 3 digit code A currency, as described [in this page](#).

Thus, there are four variants of the function `logEvent()` which logs the event

- `logEvent(String eventName)`

- `logEvent(String eventName, JSONObject parameters)`
- `logEvent(String eventName, JSONObject parameters, double valueToSum)`
- `logEvent(String eventName, JSONObject parameters, double valueToSum, String valueToSumCurrency)`

Here is how you set up some of the popular events.

### Registration Completed

This event does not have any parameters:

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject registrationDetails = new JSONObject();
try {
    qq.logEvent("registration_completed", registrationDetails);
} catch (JSONException e) {
}
```

### Category Viewed

This event has one parameter:

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject categoryDetails = new JSONObject();
try {
    categoryDetails.put("category", "apparels");
} catch (JSONException e) {
}
qq.logEvent("category_viewed", categoryDetails);
```

### Product Viewed

You may choose to have the following fields:

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject productDetails = new JSONObject();
try {
    productDetails.put("id", "123");
    productDetails.put("name", "Nikon Camera");
    productDetails.put("image_url", "http://mysite.com/products/123.png");
    productDetails.put("deep_link", "myapp/products?id=123");
    productDetails.put("type", "new");
    productDetails.put("category", "electronics");
    productDetails.put("brand", "Nikon");
    productDetails.put("color", "white");
    productDetails.put("size", "small");
    productDetails.put("price", 6999);
} catch (JSONException e) {
}
qq.logEvent("product_viewed", productDetails);
```

### Product Added to Cart:

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject productDetails = new JSONObject();
try {
    productDetails.put("id", "123");
    productDetails.put("name", "Nikon Camera");
```

(continues on next page)

(continued from previous page)

```
productDetails.put("image_url", "http://mysite.com/products/123.png");
productDetails.put("deep_link", "myapp//products?id=123");
productDetails.put("type", "new");
productDetails.put("category", "electronics");
productDetails.put("brand", "Nikon");
productDetails.put("color", "white");
productDetails.put("size", "small");
productDetails.put("price", 6999);
} catch (JsonException e) {
}
}
qg.logEvent("product_added_to_cart", productDetails);
```

#### Product Added to Wishlist:

```
QG qg = QG.getInstance(getApplicationContext());
JSONObject productDetails = new JSONObject();
try {
    productDetails.put("id", "123");
    productDetails.put("name", "Nikon Camera");
    productDetails.put("image_url", "http://mysite.com/products/123.png");
    productDetails.put("deep_link", "myapp//products?id=123");
    productDetails.put("type", "new");
    productDetails.put("category", "electronics");
    productDetails.put("brand", "Nikon");
    productDetails.put("color", "white");
    productDetails.put("size", "small");
    productDetails.put("price", 6999);
} catch (JsonException e) {
}
}
qg.logEvent("product_added_to_wishlist", productDetails);
```

#### Product Purchased:

```
QG qg = QG.getInstance(getApplicationContext());
JSONObject productDetails = new JSONObject();
try {
    productDetails.put("id", "123");
    productDetails.put("name", "Nikon Camera");
    productDetails.put("image_url", "http://mysite.com/products/123.png");
    productDetails.put("deep_link", "myapp//products?id=123");
    productDetails.put("type", "new");
    productDetails.put("category", "electronics");
    productDetails.put("brand", "Nikon");
    productDetails.put("color", "white");
    productDetails.put("size", "small");
    productDetails.put("price", 6999);
} catch (JsonException e) {
}
}
qg.logEvent("product_purchased", productDetails, 6999);
/* Or if you do not want to pass the third argument, you can simply write
qg.logEvent("product_purchased", productDetails);*/
```

#### Checkout Initiated:

```
QG qg = QG.getInstance(getApplicationContext());
JSONObject checkoutDetails = new JSONObject();
try {
```

(continues on next page)

(continued from previous page)

```
        checkoutDetails.put("num_products", 2);
        checkoutDetails.put("cart_value", 12998.44);
        checkoutDetails.put("deep_link", "myapp://myapp/cart");
    } catch (JsonException e) {
    }
    qq.logEvent("checkout_initiated", checkoutDetails);
```

**Checkout Completed:**

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject checkoutCompleted = new JSONObject();
try {
    checkoutDetails.put("num_products", 2);
    checkoutDetails.put("cart_value", 12998.44);
    checkoutDetails.put("deep_link", "myapp://myapp/cart");
} catch (JsonException e) {
}
qq.logEvent("checkout_completed", checkoutDetails, 12998.44);
/* Or if you do not want to pass the third argument, you can simply write
qq.logEvent("product_purchased", productDetails);*/
```

**Product Rated:**

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject rating = new JSONObject();
try {
    rating.put("id", "1232");
    rating.put("rating", 2);
} catch (JsonException e) {
}
qq.logEvent("product Rated", rating);
```

**Searched:**

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject search = new JSONObject();
try {
    search.put("id", "1232");
    search.put("rating", 2);
} catch (JsonException e) {
}
qq.logEvent("product Rated", rating);
```

**Reached Level:**

```
QG qq = QG.getInstance(getApplicationContext());
JSONObject level = new JSONObject();
try {
    level.put("level", 23);
} catch (JsonException e) {
}
qq.logEvent("level", rating);
```

**Your custom events**

Apart from above predefined events, you can create your own custom events, and have custom parameters in them:

```

QG qq = QG.getInstance(getApplicationContext());
JSONObject json = new JSONObject();
try {
    json.put("my_param", "some value");
    json.put("some_other_param", 123);
    json.put("what_ever", 1234.23);
} catch (JSONException e) {
}
qq.logEvent("my_custom_event", json);

```

### 3.3.5 Retrieving stored notifications

We provide the facility to store the notifications that you send. To enable notification storage, please contact us at [app@qgraph.io](mailto:app@qgraph.io). We automatically store the notifications which arrive at the SDK, and you can access them at any point of time. Here is how you access stored notifications:

```
JSONArray storedNotifications = QG.getInstance(context).getStoredNotifications();
```

Different notifications have different fields. All of them have a title and message. They may also have imageUrl (URL of icon image), bigImageUrl (URL of the big image), deepLink and some other fields depending on the type of the notification.

### 3.3.6 Configuring Batching

Our SDK batches the network requests it makes to QGraph server, in order to optimize network usage. It flushes data to the server every 15 seconds, or when number data points exceed 100.

You can force the SDK to flush the data to server any time by calling the following function:

```
QG.getSharedInstance(context).flush();
```

### 3.3.7 InApp Notifications

InApp notifications work by default and you do not have to do anything specific.

In case you wish to disable in-app notifications in some Activity, call:

```
QG.getInstance(context).hideInApp(Activity activityInWhichInAppIsToBeHidden)
```

Note that `hideInApp(activity)` should be called before `onStart()` of activity in which you wish to hide in-app gets called.

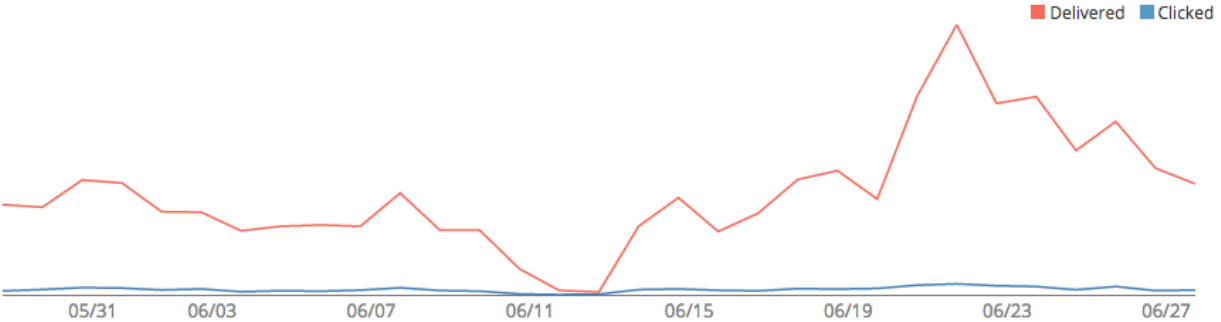
### 3.3.8 Event Attribution

To track how QG notifications are affecting the metrics on your app, we attribute some of your app events to QG notifications. We support two types of attributions: view through attribution and click through attribution. We view-through attribute an event to a notification if the event happens within 1 hour (this can be configured) of a user receiving a notification. We click-through attribute an event to a notification if the event happens within 24 hours (this can be configured) of a user receiving a notification.

You can see the attribution metrics on the performance page of the campaigns:

Campaign Performance - 3DaysAfterInstall - SLIM

SENT	DELIVERED	CLICKS	ATTRIBUTED EVENTS	ATTRIBUTED EVENT VALUES
228175	148337	7558	119382	0
	(65.01%)	(5.10%)		
		7558 Standard	348 product_removed_from_cart	0 product_removed_from_cart
			463 qg_exception	0 qg_exception
			24290 product_viewed	0 product_viewed
			595 product_added_to_cart	0 product_added_to_cart
			55 notification_browsed	0 notification_browsed
			5992 category_viewed	0 category_viewed
			81 checkout_completed	0 checkout_completed
			101 product_update_in_cart	0 product_update_in_cart
			34 product_purchased	0 product_purchased
			87423 app_launched	0 app_launched



You can change view through attribution window by using following function:

```
QG.getInstance(context).setAttributionWindow(long seconds);
```

You can change click through attribution window by using following function:

```
QG.getInstance(context).setClickAttributionWindow(long seconds);
```

## 3.4 Notification checklist

### 3.4.1 Launcher image

Make sure that you have an image called `ic_launcher.png` in your `drawable/` folder. We use this image to display as icon image if you don't set an icon image explicitly. This image should be 192px x 192px or larger, with an aspect ratio of 1:1.

### 3.4.2 Notification image

Make sure that you have an image called `ic_notification.png` in your `drawable/` folder. This is the image shown in the status bar when a notification arrives. As per Android guidelines (<http://developer.android.com/design/patterns/notifications.html>) this image should be a white image on a transparent background. The size of this image should be 72px x 72px or larger, with an aspect ratio of 1:1. This is what `ic_notification.png` should look like: [https://developer.android.com/samples/MediaBrowserService/res/drawable-hdpi/ic\\_notification.png](https://developer.android.com/samples/MediaBrowserService/res/drawable-hdpi/ic_notification.png)

### 3.4.3 Recommended sizes of images

Follow are the recommended sizes of images:

1. Big Image Notification - Big image should be 1024px x 512px or larger, with an aspect ratio close to 2:1
2. Icon Image - Icon image should be 192px x 192px or larger, with aspect ratio of 1:1
3. Carousel Notification - Recommended image size is 600px x 600px, with aspect ratio of 1:1
4. Slider Notification - 1024px x 512px or larger, with aspect ratio close to 2:1
5. Static Banner Notification - 1024px x 170px with an aspect ratio of 6:1
6. Animated Banner Notification - a series of images of 1024px x 170px with an aspect ratio of 6:1

Depending on the screen's resolution android crops the image to fit it into the container. For this, we recommend that you do not have any text in the 10% margins of Big Image and Carousel.

### 3.4.4 If you use your own Service to extend GCMListenerService

If you use your own service that extends `GCMListenerService`, following code snippet must be added in your service:

```
@Override
public void onMessageReceived(String from, Bundle data) {
    if (data.containsKey("message") && QG.isQGMessage(data.getString("message"))) {
        Context context = getApplicationContext();
        Intent intent = new Intent(context, NotificationJobIntentService.class);
```

(continues on next page)

(continued from previous page)

```
        intent.setAction("QG");
        intent.putExtras(data);
        JobIntentService.enqueueWork(context, NotificationJobIntentService.class, ↵
↵1000, intent);
        return;
    }
}
```

### 3.4.5 Receiving key value pairs in activity

If you have set key value pairs in the campaign you can get them in the activity. Let's say you passed a key valled myKey in the campaign, then you can get its value as following:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = getIntent();
    Bundle bundle = intent.getExtras();
    String val = null;
    if (bundle != null) {
        val = bundle.getString("myKey");
    }

    /* More code */
}
```



For using quantumgraph iOS-SDK, do the following steps.

1. Install our quantumgraph iOS-SDK
2. Generate a .p12 file
3. Using iOS-SDK

## 4.1 Installing iOS SDK Library

### 4.1.1 Using Cocoapods

The easiest way to integrate quantumgraph iOS SDK into your iOS project is to use CocoaPods.

1. Install CocoaPods using `gem install cocoapods`
2. If you are using Cocoapods for the first time, run `pod setup` to create a local CocoaPods spec mirror.
3. Create a file named `Podfile` in your Xcode project directory and add the following line in it:

```
pod 'quantumgraph'
```

Alternatively, in the terminal, in the xcode project directory, type:

```
pod init
```

Your pod file is created and now you can add `pod 'quantumgraph'` to the specific targets you want.

4. Run `pod install` in Xcode project directory. Cocoapods will download and install the quantumgraph iOS-SDK library and create a new Xcode workspace. From now on you should use this workspace.

### 4.1.2 Manual installation

Download the SDK from here:

For Objective C: <http://app.qgraph.io/static/sdk/ios/QGSdk-ObjC-3.3.4.zip>

For Swift: <http://app.qgraph.io/static/sdk/ios/QGSdk-Swift-3.3.4.zip>

- In your Xcode project, Go to File, add new Group to your project and name it as QGSdk.
- Add libQGSdk.a and QGSdk.h in QGSdk group
- Go to Project -> Target -> Build Phases. In the section “Link Binary with Libraries”, add following frameworks:
  - AdSupport.framework
  - SystemConfiguration.framework
  - CoreTelephony.framework
  - CoreLocation.framework
  - ImageIO.framework
  - MobileCoreServices.framework
  - libz.tbd

We track location only if you initialize location service. If you don’t add location usage key in info.plist file, we don’t track the location of the user.

However, you do not need to add these frameworks if you use cocoapods.

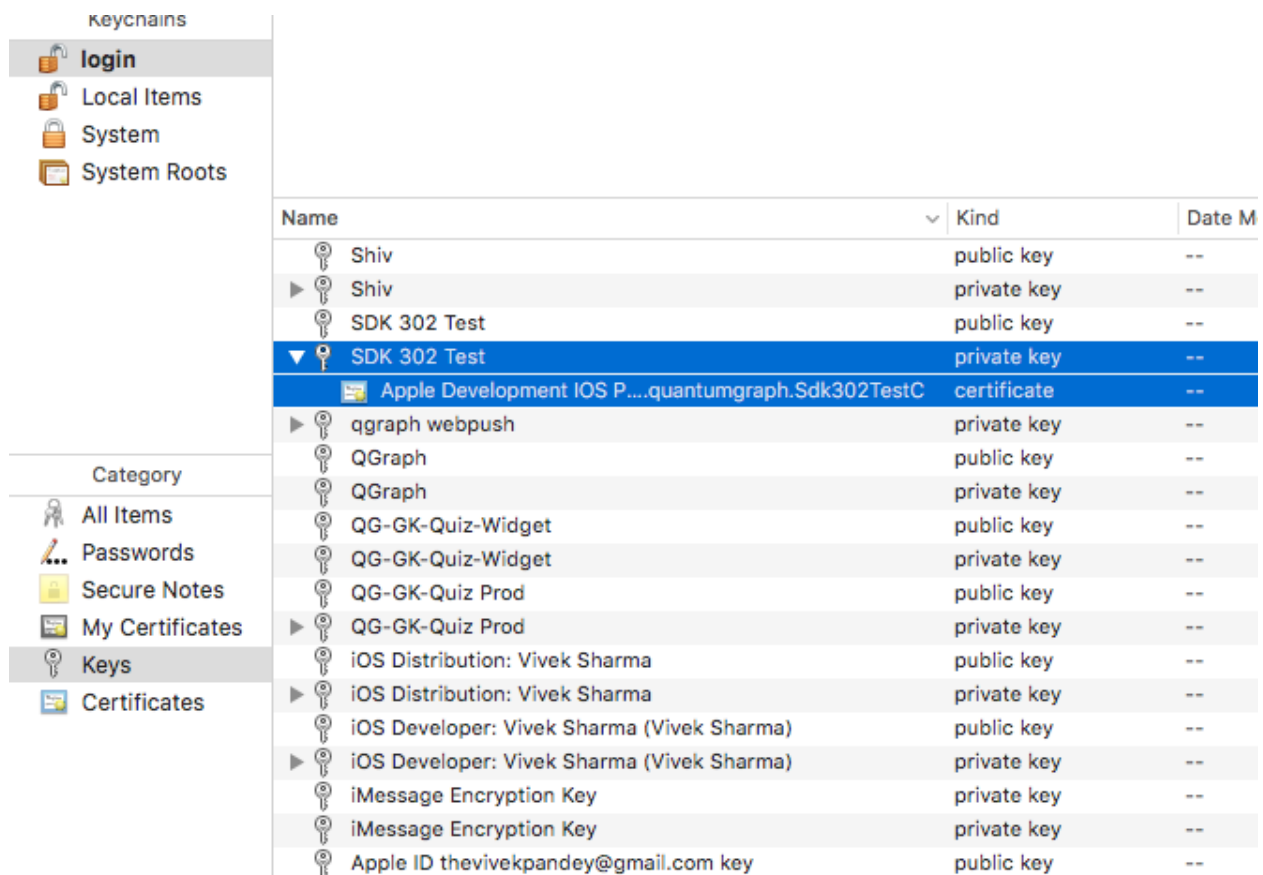
## 4.2 Generating .p12 file

### 4.2.1 Generating SSL Certificate for APP ID

1. Log in to the iOS Dev Center and select the *Certificates, Identifiers and Profiles*
2. Go to **App IDs** in the **Identifiers** Section of the sidebar and select your app if automatically created. Skip to Step 6.
3. To create new App click + and fill the details for App ID, App Services (Check the push notification Checkbox) and Explicit App ID(Should be same as Bundle ID in your App)
4. You will be asked to verify the details of the app id, if everything seems okay click Register.
5. In the *Push Notification* row there are two orange lights that say “Configurable” in the Development and Distribution column.
6. Select your App ID and click on **EDIT**.
7. If Push Notification is not enabled, enable it to make it configurable.
8. Select the Create Certificate in the Development/Production SSL Certificate
9. In the next step it will ask you for generating a CSR

## 4.2.2 Generating the Certificate Signing Request

1. Open Keychain Access on your Mac and choose the menu option *Certificate Assistant -> Request a Certificate* from a Certificate Authority
2. Enter some descriptive name for Common Name (Give your app name appended by QGraph preferably to identify it)
3. Check Save to disk option and click continue. It saves a .certSigningRequest file.
4. In the Keys section of the Keychain Access, a new private key would have appeared with Common name specified
5. In the “App IDs” section in the apple developer account, choose the CSR that you generated to create the push certificate
6. Click Continue and download the certificate
7. Double click on the downloaded certificate. This will add your certificate to your private key in your keychain
8. Go to Keys section in the Keychain and find your private key
9. You should be able to expand the private key and find your certificate with it. Select both the private key and the certificate after expanding (as shown in the snapshot)



10. Right click on it to export it as .p12 file. Make sure you are exporting 2 items as shown
11. Name your file as your\_app\_name and save it with file format .p12
12. You will be prompted to enter a password. You can directly click Ok or add any password to it. If you add any password please remember it and send it along with your .p12 file.

Name	Kind	Date M
Shiv	public key	--
Shiv	private key	--
SDK 302 Test	public key	--
SDK 302 Test	private key	--
Apple Development iOS P....quantum	certificate	--
qgraph webpush	te key	--
QGraph	c key	--
QGraph	te key	--
QG-GK-Quiz-Widget	c key	--
QG-GK-Quiz-Widget	te key	--
QG-GK-Quiz Prod	public key	--
QG-GK-Quiz Prod	private key	--
iOS Distribution: Vivek Sharma	public key	--

Copy 2 items

Delete 2 items

Export 2 items...

Get Info

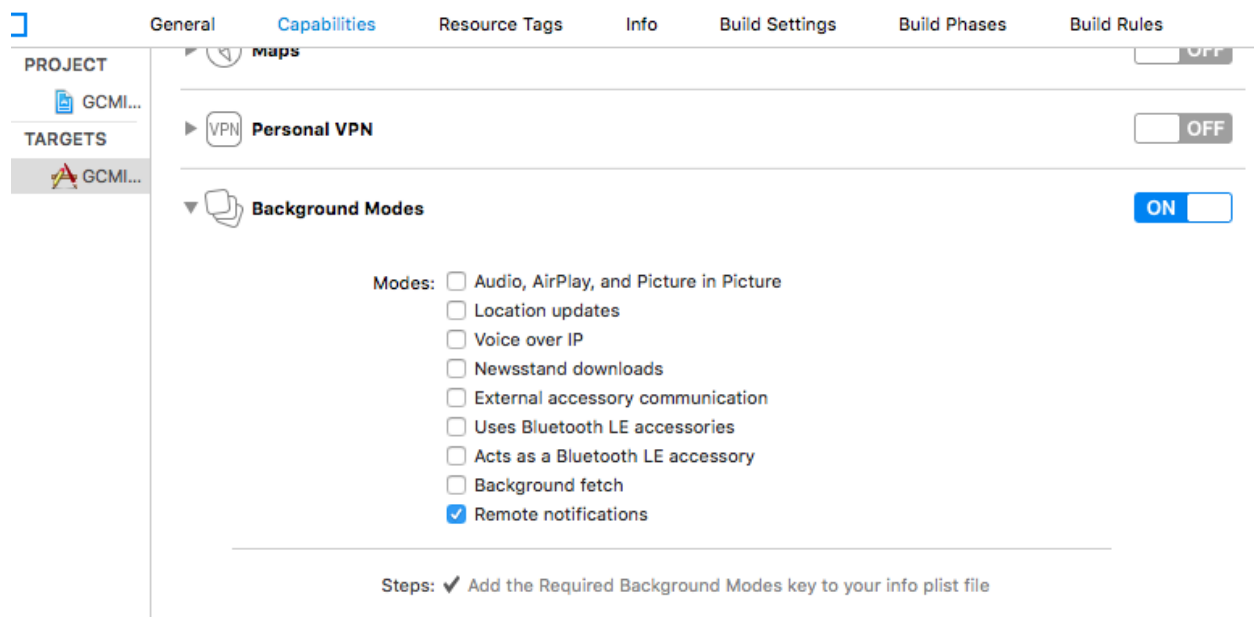
13. In the next step, you will require your system password to finally save the file.

Your p12 file is ready to be exported. Upload it in your account. Go to “Integration” -> “iOS” to upload p12 files.

## 4.3 Using iOS SDK - Objective C

### 4.3.1 Change required for APNS Token and User Tracking

It is required by QG SDK that you enable *Background Mode* in the *Capabilities* section of the main app target. After enabling background modes, select **Remote Notification** as shown in the snapshot.



### 4.3.2 AppDelegate Changes

To initialise the library, in AppDelegate add `#import "QGSdk.h"`

In `didFinishLaunchingWithOptions` method of AppDelegate, add the following code for registering for remote notification:

```
(BOOL)application:(UIApplication *)application_
↳didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    if (floor(NSFoundationVersionNumber) < NSFoundationVersionNumber_iOS_8_0) {
        // here you go with iOS 7
        [[UIApplication sharedApplication] registerForRemoteNotificationTypes:_
↳(UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound |_
↳UIRemoteNotificationTypeAlert)];
    } else {
        // registering push notification in ios 8 and above
        UIUserNotificationType types = UIUserNotificationTypeAlert |_
↳UIUserNotificationTypeSound |
        UIUserNotificationTypeBadge;
        UIUserNotificationSettings *settings = [UIUserNotificationSettings_
↳settingsForTypes:types
        categories:nil];
        [[UIApplication sharedApplication] registerUserNotificationSettings:settings];
    }
    //replace <your app id> with the one you received from QGraph
    [[QGSdk sharedInstance] onStart:@"<YOUR APP ID>" setDevProfile:NO];

    return YES;
}
```

Note that `[[UIApplication sharedApplication] registerForRemoteNotifications]` is called by our SDK for iOS 8 and iOS 9.

For development profile, set Boolean to YES in the following method:

```
[[QGSdk sharedInstance] onStart:@"<your app id>" setDevProfile:YES];
```

Just build and run the app to make sure that you receive a message that app would like to send push notification. If you get code signing error, make sure that proper provisioning profile is selected

Add the following code in AppDelegate.m to get the device token for the user:

```
- (void)application:(UIApplication*)application_
↳didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken
{
    NSLog(@"My token is: %@", deviceToken);
    [[QGSdk sharedInstance] setToken:deviceToken];
}

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

QGSdk `setToken` method will log user's token so that you can send push notification to the user.

### 4.3.3 Handling Push Notification

Notifications are delivered while the app is in foreground, background or not running state. We can handle them in the following delegate methods.

If the remote notification is tapped, the system launches the app and the app calls its delegate's `application:didFinishLaunchingWithOptions:` method, passing in the notification payload (for remote notifications). Although `application:didFinishLaunchingWithOptions:` is not the best place to handle the notification, getting the payload at this point gives you the opportunity to start the update process before your handler method is called.

For remote notifications, the system also calls the `application:didReceiveRemoteNotification:fetchCompletionHandler:` method of the app delegate.

You can handle the notification and its payload as described:

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

    // Payload can be handled in this way
    NSDictionary *notification = [launchOptions_
↳objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];
    if (notification) {
        // you custom methods...
    }
    return YES;
}
```

The notification is delivered when the app is running in the foreground. The app calls the `application:didReceiveRemoteNotification:fetchCompletionHandler:` method of the app delegate. (If `application:didReceiveRemoteNotification:fetchCompletionHandler:` is not implemented, the system calls `application:didReceiveRemoteNotification:`.) However, it is advised to use `application:didReceiveRemoteNotification:fetchCompletionHandler:` method to handle push notification.

Implementation:

```
- (void)application:(UIApplication *)application_
↳didReceiveRemoteNotification:(NSDictionary *)userInfo
    fetchCompletionHandler:(void (^)(UIBackgroundFetchResult result))handler {
    // Please make sure you add this method
    [[QGSdk sharedInstance] application:application_
↳didReceiveRemoteNotification:userInfo];

    handler(UIBackgroundFetchResultNoData);
    NSLog(@"Notification Delivered");
}
```

You can also handle background operation using the above method once remote notification is delivered. For this make sure, wake app in background is selected while creating a campaign to send the notification.

If you have implemented `application:didReceiveRemoteNotification:` add method `[[QGSdk sharedInstance] application:application_ didReceiveRemoteNotification:userInfo];` inside it. Your implementation should look like:

```
- (void)application:(UIApplication *)application_
↳didReceiveRemoteNotification:(NSDictionary *)userInfo {
    [[QGSdk sharedInstance] application:application_
↳didReceiveRemoteNotification:userInfo];
```

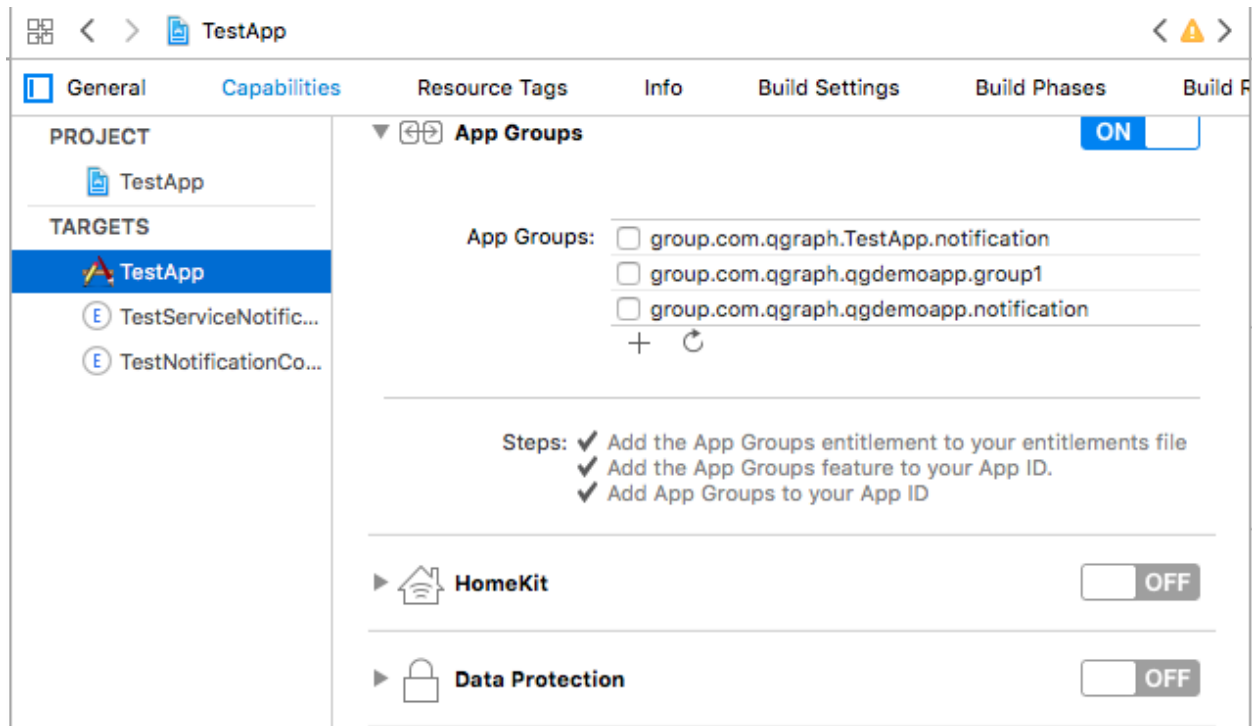
(continues on next page)

(continued from previous page)

}

### 4.3.4 Changes for iOS 10

For integrating QGraph notification SDK, you need to add Capabilities **APP GROUPS**. Go to Project > Main Target > **Capabilities**. Check on App Groups and add a group as below. Use your bundle id to create App Group. For example, if your bundle id is `com.company.appname`, App Group could be `group.com.company.appname.xyz`.



You need App Group so that data can be shared between extensions. Use that App Group name in `onStart:withAppGroup:setDevProfile:` in App Delegate.

### 4.3.5 AppDelegate Changes for iOS 10

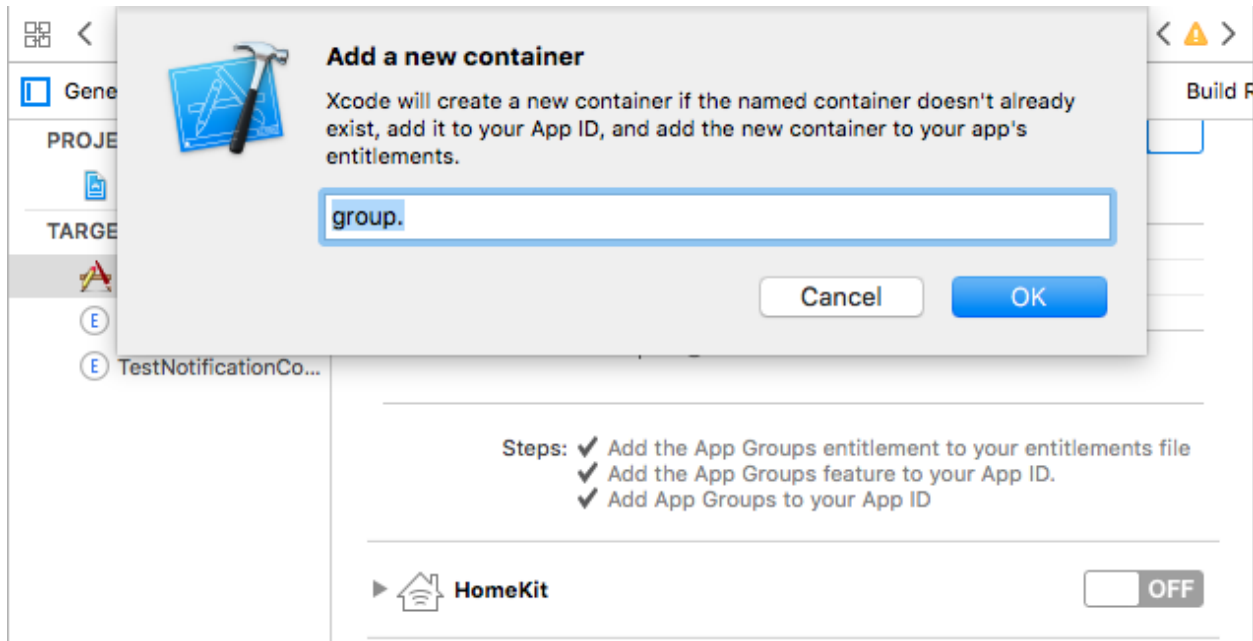
Add framework **UserNotifications** to app target and import in app delegate

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

//Define macros for checking iOS version
#define SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(v)  ([[UIDevice currentDevice]
↳systemVersion] compare:v options:NSNumericSearch] != NSOrderedAscending)
#define SYSTEM_VERSION_LESS_THAN(v)                 ([[UIDevice currentDevice]
↳systemVersion] compare:v options:NSNumericSearch] == NSOrderedAscending)

- (BOOL)application:(UIApplication *)application
↳didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
```

(continues on next page)



(continued from previous page)

```

QGSdk *qgsdk = [QGSdk sharedInstance];

[qgsdk onStart:@"<app_id>" withAppGroup:@"group.com.company.product.extension"
↳setDevProfile:true];

if (SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(@"10.0")) {
    UNAuthorizationOptions options = (UNAuthorizationOptions)
↳(UNAuthorizationOptionAlert | UNAuthorizationOptionBadge |
↳UNAuthorizationOptionSound | UNAuthorizationOptionCarPlay);

    UNUserNotificationCenter *center = [UNUserNotificationCenter
↳currentNotificationCenter];
    center.delegate = self;

    NSSet *categories = [NSSet initWithObjects:[qgsdk
↳getQGSlderPushActionCategoryWithNextButtonTitle:nil withOpenAppButtonTitle:nil],
↳nil];
    [center setNotificationCategories:categories];

    [center requestAuthorizationWithOptions:options completionHandler:^(BOOL
↳granted, NSError *error){
        NSLog(@"GRANTED: %i, Error: %@", granted, error);
    }];
} else if (SYSTEM_VERSION_LESS_THAN(@"10.0")) {
    UIUserNotificationType types = UIUserNotificationTypeAlert |
↳UIUserNotificationTypeSound |
    UIUserNotificationTypeBadge;
    UIUserNotificationSettings *settings = [UIUserNotificationSettings
↳settingsForTypes:types
↳categories:nil];
    [[UIApplication sharedApplication] registerUserNotificationSettings:settings];

```

(continues on next page)



(continued from previous page)

```

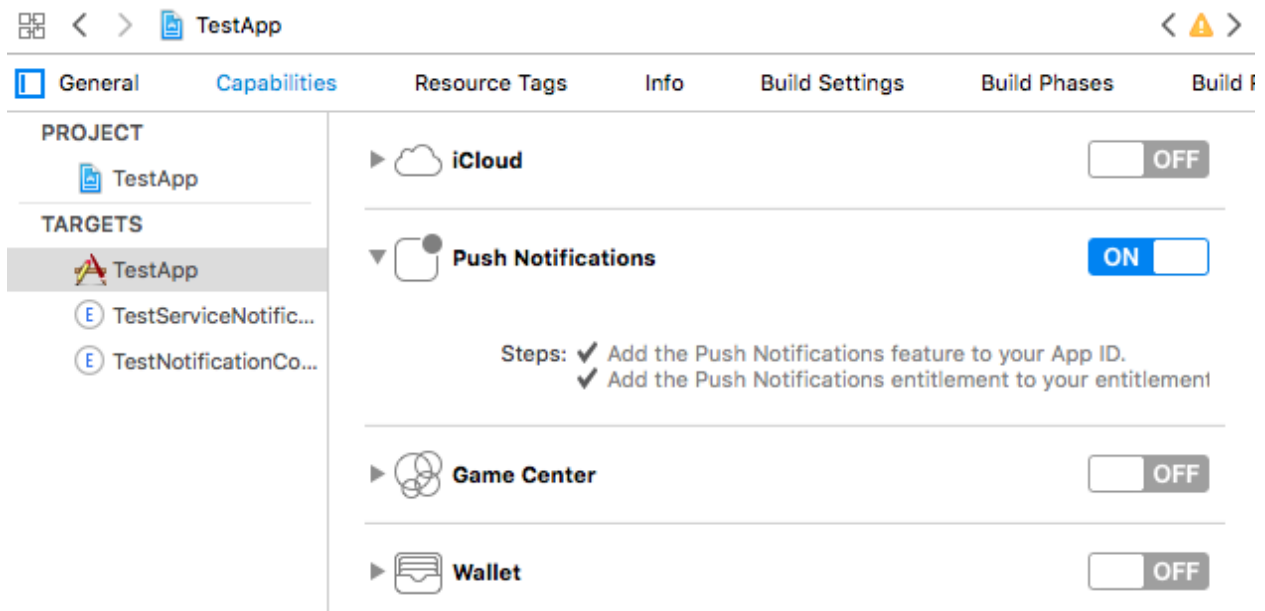
    }
    return YES;
}

```

**NOTE:** If you have your own existing notification action category for iOS 10, you can add it along with Graph CAROUSEL/SLIDER category implemented as above. For the carousel and slider push action buttons, you can also specify button titles. Next button will be used to animate the carousel/slider and Open App Button will open the app with deeplink if any.

### 4.3.6 Handling Push Notification in iOS 10

There are new delegate methods introduced in iOS 10 to track notification and display in foreground state as well. To track notifications in background state, you need to enable background mode in the capabilities. Above all these you need to activate push notification in the capabilities. This will add entitlement files to your app target.



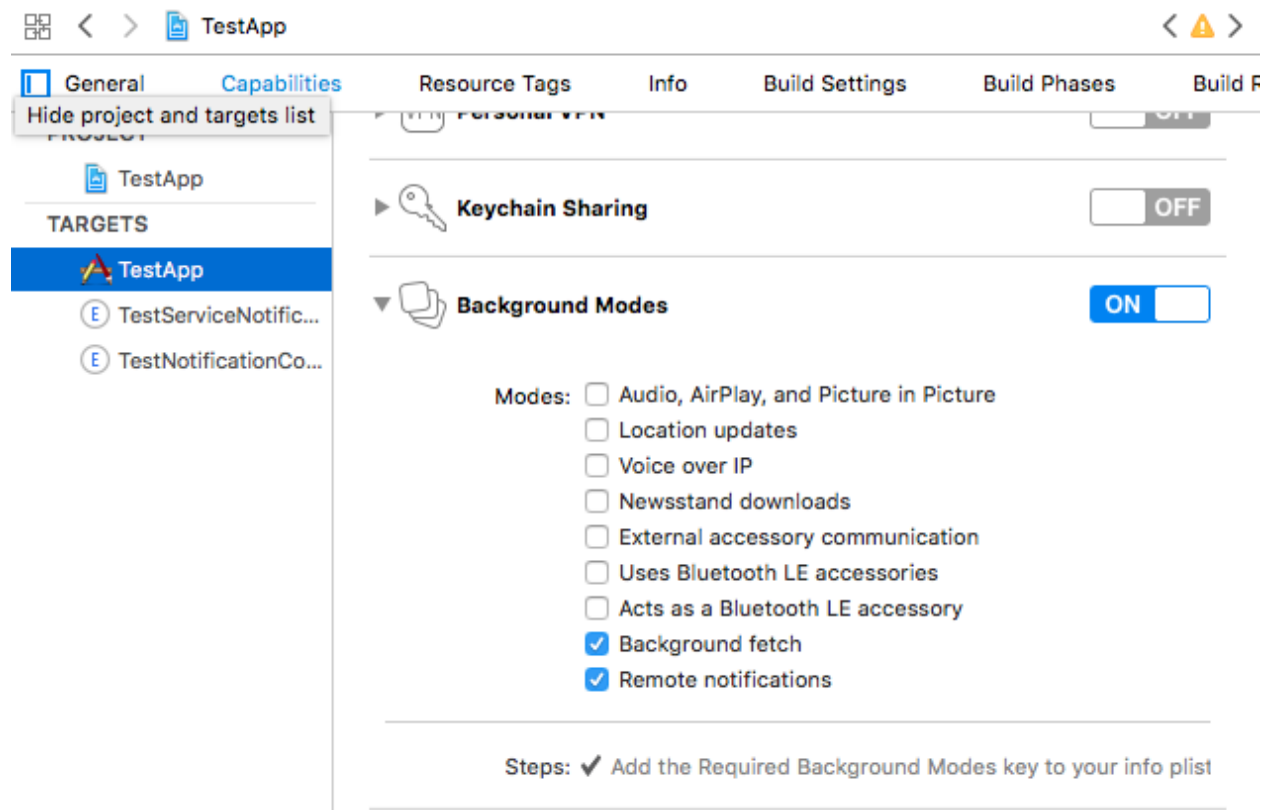
1. You might have already included this method. Please make sure `[[QGSdk sharedInstance] application:application didReceiveRemoteNotification:userInfo];` is added in it. It is required to track notifications.

```

//used for silent push handling
//pass completion handler UIBackgroundFetchResult accordingly
- (void)application:(UIApplication *)application_
↳didReceiveRemoteNotification:(nonnull NSDictionary *)userInfo_
↳fetchCompletionHandler:(nonnull void (^)(UIBackgroundFetchResult))completionHandler
↳{
    [[QGSdk sharedInstance] application:application_
↳didReceiveRemoteNotification:userInfo];
    completionHandler(UIBackgroundFetchResultNoData);
}

```

2. The method will be called on the delegate only if the application is in the foreground. If the method is not implemented or the handler is not called in a timely manner then the notification will not be presented. The



application can choose to have the notification presented as a sound, badge, alert and/or in the notification list. This decision should be based on whether the information in the notification is otherwise visible to the user.

```
- (void)userNotificationCenter:(UNUserNotificationCenter *)center_
↳willPresentNotification:(UNNotification *)notification withCompletionHandler:(void_
↳(^) (UNNotificationPresentationOptions options))completionHandler {
    [[QGSdk sharedInstance] userNotificationCenter:center_
↳willPresentNotification:notification];

    [UIApplication sharedApplication].applicationIconBadgeNumber = 0;
    UNNotificationPresentationOptions option = UNNotificationPresentationOptionBadge_
↳| UNNotificationPresentationOptionSound | UNNotificationPresentationOptionAlert;

    completionHandler(option);
}
```

3. The method will be called on the delegate when the user responded to the notification by opening the application, dismissing the notification or choosing a *UNNotificationAction*. The delegate must be set before the application returns from *applicationDidFinishLaunching:*.

**NOTE:** This method is specifically required for carousel and slider push to work. Also used to track notification\_clicked event for QGraph push.

```
- (void)userNotificationCenter:(UNUserNotificationCenter *)center_
↳didReceiveNotificationResponse:(UNNotificationResponse *)response_
↳withCompletionHandler:(void(^)())completionHandler {
    [[QGSdk sharedInstance] userNotificationCenter:center_
↳didReceiveNotificationResponse:response];
}
```

(continues on next page)

(continued from previous page)

```
completionHandler();
}
```

### 4.3.7 Handling Deeplink for QGraph Push

For Push notifications deeplinks should be handled in the method *didReceiveNotificationResponse:withCompletionHandler:* as described below. You can get the deeplink url and then pass it to *openUrl:* and then you should get a callback in the *application:openUrl:options* where you can handle the opening of a specific page.

```
- (void)userNotificationCenter:(UNUserNotificationCenter *)center_
↳didReceiveNotificationResponse:(UNNotificationResponse *)response_
↳withCompletionHandler:(void(^)())completionHandler {
    NSDictionary *userInfo = response.notification.request.content.userInfo;
    if ([userInfo objectForKey:@"deepLink"]) {
        NSURL *url = [NSURL URLWithString:userInfo[@"deepLink"]];
        dispatch_async(dispatch_get_main_queue(), ^{
            [[UIApplication sharedApplication] openUrl:url];
        });
    }
    [[QGSdk sharedInstance] userNotificationCenter:center_
↳didReceiveNotificationResponse:response];
    completionHandler();
}
```

For any deeplink specified in In-App campaigns, you should get a callback in the below method. You need to handle it on your own to open any specific page.

```
- (BOOL)application:(UIApplication *)app openUrl:(NSURL *)url options:(NSDictionary
↳<NSString *,id> *)options {
    NSLog(@"deeplink");
    return true;
}
```

### 4.3.8 Adding Extensions for iOS Push with Attachment and QGraph Carousel and Slider Push

In iOS 10, two frameworks has been introduced for handling push notification with content. You can have a push notification with image, gif, audio and video. Apart from that you can also have your custom UI for notifications. For this, payload can be modified and used to download content before the notification is drawn. You simply need to follow the below steps to add two of the extensions targets for handling these notifications: **Service Extension** and **Content Extension**.

Before proceeding make sure to download all the QGraph files to be used here. You should have these files with you

1. QGNotificationSdk
2. QGNotificationServiceExtension
3. QGNotificationContentExtension

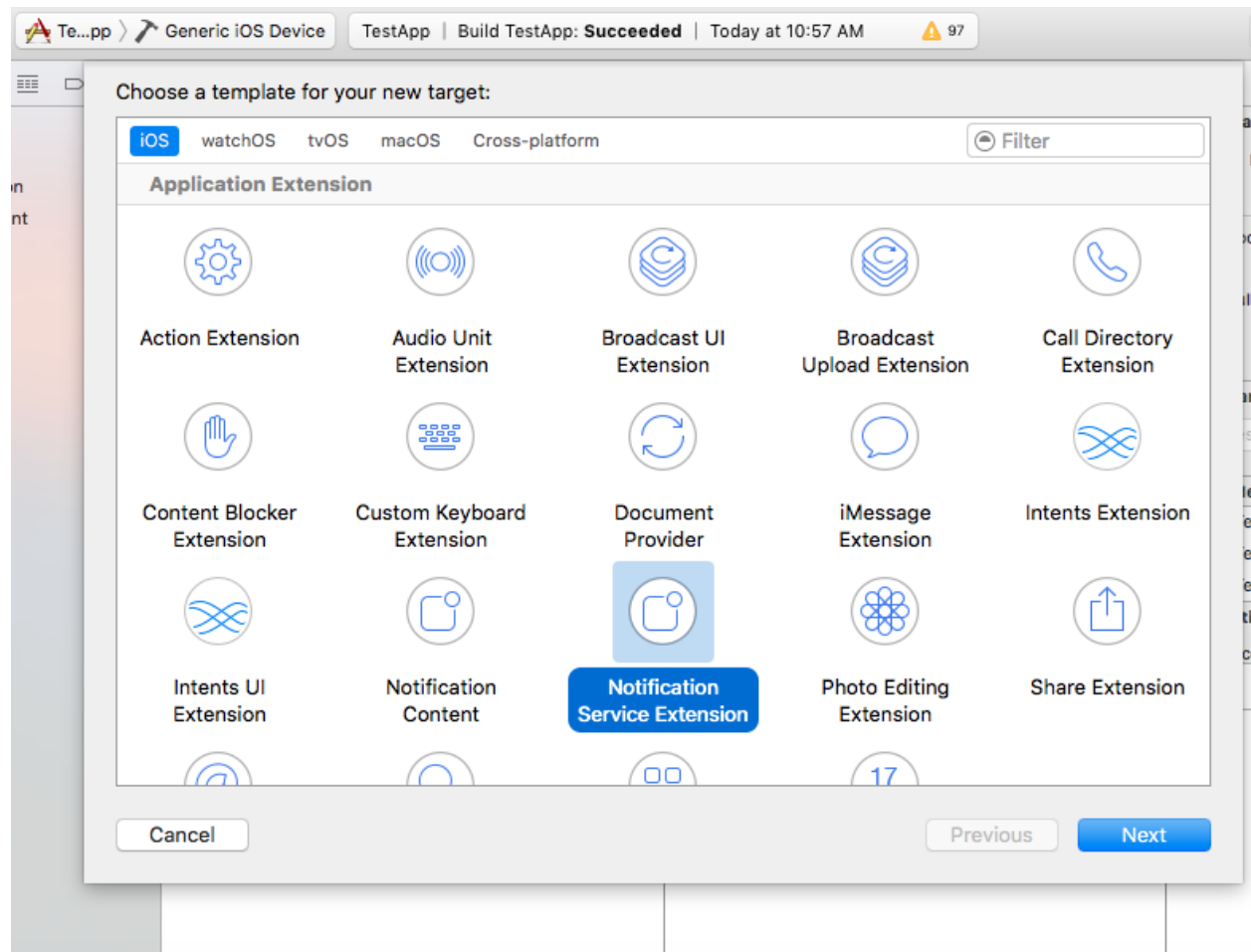
NOTE: These files are to be used with service and content extensions only. Do not add them to main app target.

### 4.3.9 Notification Service Extension

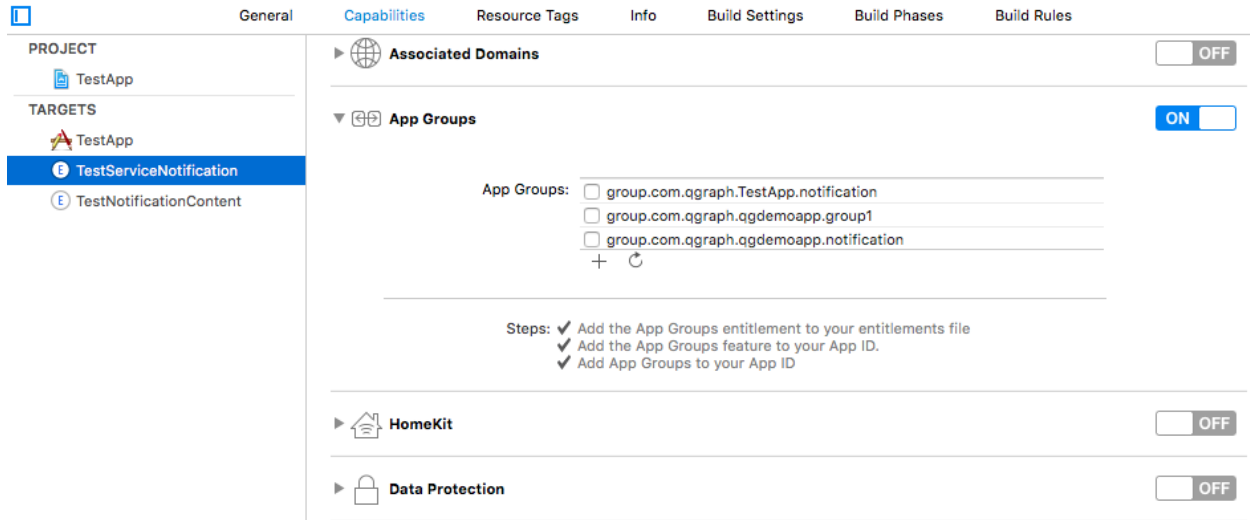
Service extension is basically the target extension where you get a callback when a push is delivered to the device. You can download and create attachments here. If you fail to download the content and pass it to `contentHandler` within certain time, default standard notification will be drawn.

#### 4.3.9.1 Adding Service extension

1. Add an iOS target and choose Notification Service extension and proceed. Add a product name and Finish. When created you will be **prompted to activate the target**. Once activated, you can see 3 files added, `NotificationService` (.h and .m ) and `Info.plist`.



2. Please delete the `NotificationService.h` and `NotificationService.m` files.
3. Add files from `QGNotificationServiceExtension`
4. Go to project navigator and select the *Service Extension Target*
5. Select *Capabilities* and check on *App Group* and select the *APP GROUP* which you added to your main app target.
6. Go to `NotificationService.m` and change your app group



```
static NSString *APP_GROUP = @"group.com.company.product.extension";
```

#### 4.3.9.2 Adding Content Extension

1. Add an iOS target and choose Notification Content extension and proceed. Add a product name and Finish. When created you will be **prompted to activate the target**. Once activated, you can see 4 files added, NotificationViewController (.h and .m), MainInterface.storyboard and Info.plist.
2. Please delete NotificationViewController and MainInterface.storyboard.
3. Add these files from **QGNotificationContentExtension**.
4. As done above, enable App Groups and select the same app group through capabilities of the content extension target.
5. Go to NotificationViewController.m and change your app group

```
static NSString *APP_GROUP = @"group.com.company.product.extension";
```

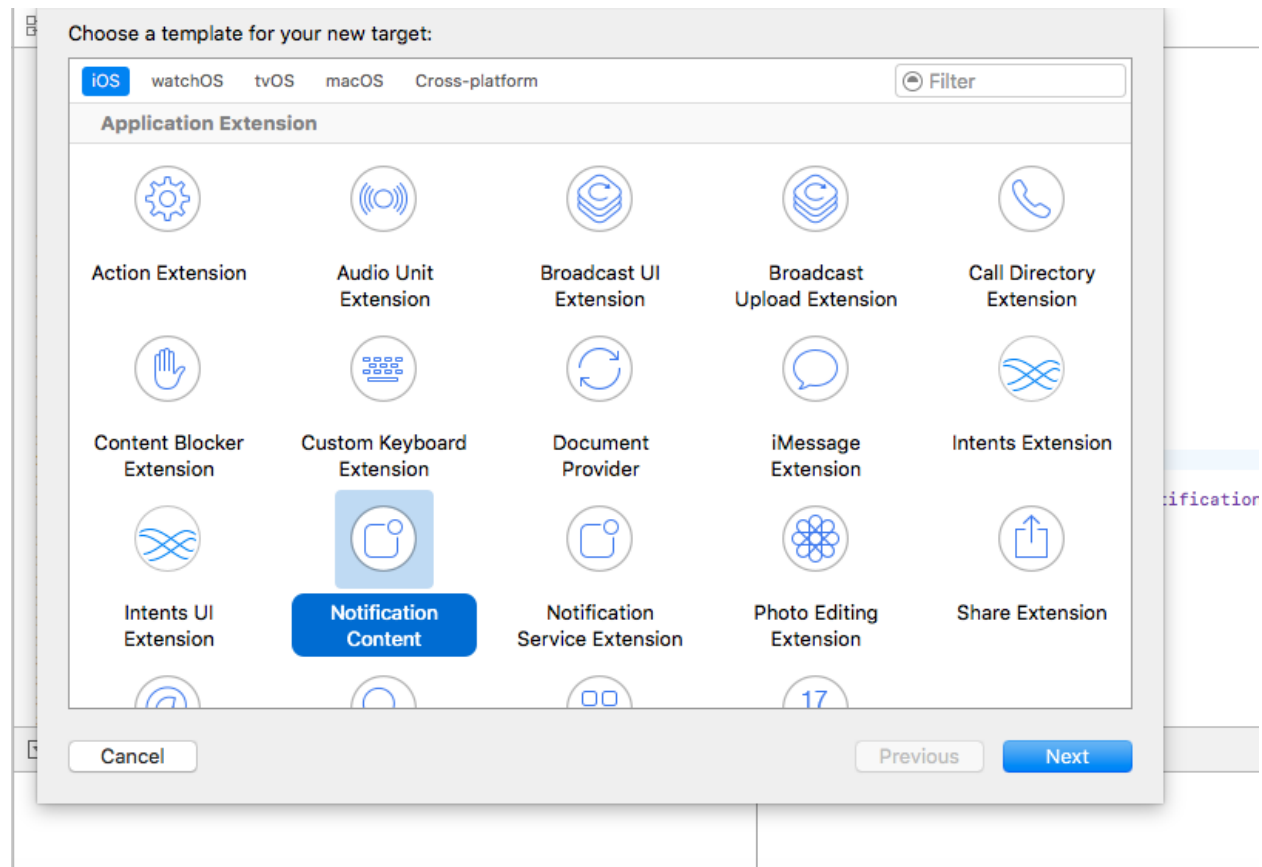
6. Go to Info.plist and add **UNNotificationExtensionDefaultContentHidden** (Boolean) - YES and **UNNotificationExtensionCategory** (string) - **QGCAROUSEL** in NSExtensionAttributes dict of NSExtension dict as shown in the screenshot.
7. Add *QuartzCore.framework* in this target.
8. Add **QGNotificationSdk** to both extension targets. Do not add it to main app target.

#### NOTE:

1. Please make sure **APP\_GROUP** used in all the three targets are same.
2. Set the deployment target to 10.0 in both the extensions.
3. Remove **-ObjC/\$(inherited)** (if it exists) from build settings of service and content extension targets.

#### 4.3.10 Click Through and View Through Attribution

QGraph SDK attributes events for each notification clicked or viewed. Events are attributed on the basis of time interval specified for all log events.



Key	Type	Value
▼ Information Property List	Dictionary	(10 items)
Localization native development region	String	en
Bundle display name	String	TestNotificationContent
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	XPC!
Bundle versions string, short	String	1.0
Bundle version	String	1
▼ NSExtension	Dictionary	(3 items)
▼ NSExtensionAttributes	Dictionary	(3 items)
UNNotificationExtensionDefaultContentHidden	Boolean	YES
UNNotificationExtensionCategory	String	QGCAROUSEL
UNNotificationExtensionInitialContentSizeRatio	Number	1
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.usernotifications.content-extension

Currently, click through attribution works for push notification clicked (sent via QGraph) and InApp notification clicked. View through attribution works only in the case of InApp notifications.

By default click through attribution window (time interval) is set to 86400 seconds (24 hrs) and view through attribution window is set to 3600 seconds (1 hr). You can change this window any time using following apis:

```
// to set click through attribution window
- (void)setClickAttributionWindow:(NSInteger)seconds;
// to set view through attribution window
- (void)setAttributionWindow:(NSInteger)seconds;
```

To set a custom value, pass the time interval in seconds. e.g.: to set click attribution window to be 12 hrs:

```
[[QGSdk sharedInstance] setClickAttributionWindow:43200];
```

To disable any of the click through or view through attribution, pass the value 0. E.g.:

```
[[QGSdk sharedInstance] setAttributionWindow:0];
```

### 4.3.11 Configuring Batching

Our SDK batches the network requests it makes to QGraph server, in order to optimize network usage. By default, it flushes data to the server every 15 seconds in release builds, and every second in debug builds. This interval is configurable using the following method:

```
[[QGSdk sharedInstance] setFlushInterval:<flush interval in seconds>];
```

Further, you can force the SDK to flush the data to server any time by calling the following function:

```
[[QGSdk sharedInstance] flush];
```

Furthermore, you can invoke a completion handler after flush using function:

```
[[QGSdk sharedInstance] flushWithCompletion:^(
    //some method
)];
```

### 4.3.12 Matching mobile app users with mobile web users

Our SDK can help you track your mobile app users across your app and mobile web. If you want to enable this functionality, you need to add **Safari Services Framework** in your app.

If you have added Safari Services Framework in your app, but would like to *disable* our tracking, use the following function:

```
[[QGSdk sharedInstance] disableUserTrackingForSafari];
```

### 4.3.13 In app Notification

QGraph SDK supports InApp notification starting in sdk version 2.0.0. InApp notification are supported in two types: Textual and Image. Visit your QGraph account to create InApp Campaigns.

These notifications are shown based on the log events app sends through our sdk and the matching conditions of the InApp Campaigns. Make sure to send appropriate log event (with parameter or valueToSum if any) for InApp notifications to work.

By default, InApp notifications are enabled. You can enable/disable it anytime using following method in the sdk:

```
- (void)disableInAppCampaigns:(BOOL)disabled;
```

eg. to disable:

```
[[QGSdk sharedInstance] disableInAppCampaigns:YES];
```

Disabling it will restrict the device to get any new InApp campaigns. It will also disable InApp notification to be drawn.

For All InApp Notification, you can configure a deep link url from the dashboard while creating an InApp campaign.

There is tap event defined on textual and image InApps. When the user taps on text on textual InApp or clicks on image in the image InApp and if there is a valid deep link setup, you will get a call back in your AppDelegate.m in the following method:

```
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary
↳<NSString *,id> *)options;
```

or:

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
↳sourceApplication:(nullable NSString *)sourceApplication annotation:(id)annotation;
↳(Deprecated in iOS_9)
```

Here you can implement your deep link with the url.

### 4.3.14 Registering Your Actionable Notification Types

Actionable notifications let you add custom action buttons to the standard iOS interfaces for local and push notifications. Actionable notifications give the user a quick and easy way to perform relevant tasks in response to a notification. Prior to iOS 8, user notifications had only one default action. In iOS 8 and later, the lock screen, notification banners, and notification entries in Notification Center can display one or two custom actions. Modal alerts can display up to four. When the user selects a custom action, iOS notifies your app so that you can perform the task associated with that action.

For defining a notification action and its category, and to handle actionable notification, please refer the description in the apple docs. ([Click here](#))

Action Category can be set in the dashboard while sending notification. While configuring to send notification through campaigns, use the categories defined in the app.

### 4.3.15 Logging user profile information

User profiles are information about your users, like their name, city, date of birth or any other information that you may wish to track. You log user profiles by using one or more of the following functions:

```
- (void)setUserId:(NSString *)userId;
```

Other methods you may use to pass user profile parameters to us:



```
- (void)setUserId:(NSString *)userId;
- (void)setName:(NSString *)name;
- (void)setFirstName:(NSString *)name;
- (void)setLastName:(NSString *)name;
- (void)setCity:(NSString *)city;
- (void)setEmail:(NSString *)email;
- (void)setDayOfBirth:(NSNumber *)day;
- (void)setMonthOfBirth:(NSNumber *)month;
- (void)setYearOfBirth:(NSNumber *)year;
```

Other than these method, you can log your own custom user parameters. You do it using:

```
- (void)setCustomKey:(NSString *)key withValue:(id)value;
```

For example, you may wish to have the user's current rating like this:

```
[[QGSdk sharedInstance] setCustomKey:@"current rating" withValue:@"123"];
```

### 4.3.16 Logging events information

Events are the activities that a user performs in your app, for example, viewing the products, playing a game or listening to a music. Each event has follow properties:

1. Name. For instance, the event of viewing a product is called `product_viewed`
2. Optionally, some parameters. For instance, for event `product_viewed`, the parameters are `id` (the id of the product viewed), `name` (name of the product viewed), `image_url` (image url of the product viewed), `deep_link` (a deep link which takes one to the product page in the app), and so on.
3. Optionally, a “value to sum”. This value will be summed up when doing campaign attribution. For instance, if you pass this value in your checkout completed event, you will be able to view stats such as a particular campaign has been responsible to drive Rs 84,000 worth of sales.
4. Optionally, the currency of value to sum. Currency needs to be a 3 digit code A currency, as described in [this page](#).

You log events using the function `logEvent()`. It comes in four variations

- `(void)logEvent:(NSString *)name`
- `(void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters`
- `(void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters withValueToSum:(NSNumber *) valueToSum`
- `(void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters withValueToSum:(NSNumber *) valueToSum withValueToSumCurrency:(NSString *)vtsCurr`

Once you log event information to use, you can segment users on the basis of the events (For example, you can create a segment consisting of users have not launched for past 7 days, or you can create a segment consisting of users who, in last 7 days, have purchased a product whose value is more than \$1000)

You can also define your events, and your own parameters for any event. However, if you do that, you will need to sync up with us to be able to segment the users on the basis of these events or customize your creatives based on these events.

You can use the following method to pass event information to us:

```
- (void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters;
```

Here is how you set up some of the popular events.

### Registration Completed

This event does not have any parameters:

```
[[QGSdk sharedInstance] logEvent:@"registration_completed" withParameters:nil];
```

### Category Viewed

This event has one parameter:

```
NSMutableDictionary *categoryDetails = [[NSMutableDictionary alloc] init];
[CategoryDetails setObject:@"apparels" forKey:@"category"];

[[QGSdk sharedInstance] logEvent:@"category_viewed"
↳withParameters:categoryDetails];
```

### Product Viewed

You may choose to have the following fields:

```
NSMutableDictionary *productDetails = [[NSMutableDictionary alloc] init];
[productDetails setObject:@"123" forKey:@"id"];
[productDetails setObject:@"Nikon Camera" forKey:@"name"];
[productDetails setObject:@"http://mysite.com/products/123.png" forKey:@"image_url"];
[productDetails setObject:@"myapp//products?id=123" forKey:@"deep_link"];
[productDetails setObject:@"black" forKey:@"color"];
[productDetails setObject:@"electronics" forKey:@"category"];
[productDetails setObject:@"small" forKey:@"size"];
[productDetails setObject:@"6999" forKey:@"price"];
[[QGSdk sharedInstance] logEvent:@"product_viewed" withParameters:productDetails];
```

### Product Added to Wishlist:

```
NSMutableDictionary *productDetails = [[NSMutableDictionary alloc] init];
[productDetails setObject:@"123" forKey:@"id"];
[productDetails setObject:@"Nikon Camera" forKey:@"name"];
[productDetails setObject:@"http://mysite.com/products/123.png" forKey:@"image_url"];
[productDetails setObject:@"myapp//products?id=123" forKey:@"deep_link"];
[productDetails setObject:@"black" forKey:@"color"];
[productDetails setObject:@"electronics" forKey:@"category"];
[productDetails setObject:@"Nikon" forKey:@"brand"];
[productDetails setObject:@"small" forKey:@"size"];
[productDetails setObject:@"6999" forKey:@"price"];
[[QGSdk sharedInstance] logEvent:@"product_added_to_wishlist"
↳withParameters:productDetails];
```

### Product Purchased:

```
NSMutableDictionary *productDetails = [[NSMutableDictionary alloc] init];
[productDetails setObject:@"123" forKey:@"id"];
[productDetails setObject:@"Nikon Camera" forKey:@"name"];
[productDetails setObject:@"http://mysite.com/products/123.png" forKey:@"image_url"];
[productDetails setObject:@"myapp//products?id=123" forKey:@"deep_link"];
[productDetails setObject:@"black" forKey:@"color"];
[productDetails setObject:@"electronics" forKey:@"category"];
```

(continues on next page)

(continued from previous page)

```
[productDetails setObject:@"small" forKey:@"size"];
[productDetails setObject:@"6999" forKey:@"price"];
```

and then:

```
[[QGSdk sharedInstance] logEvent:@"product_purchased"
↳withParameters:productDetails];
```

or:

```
[[QGSdk sharedInstance] logEvent:@"product_purchased"
↳withParameters:productDetails withValueToSum price];
```

#### Checkout Initiated:

```
NSMutableDictionary *checkoutDetails = [[NSMutableDictionary alloc] init];
[checkoutDetails setObject:@"2" forKey:@"num_products"];
[checkoutDetails setObject:@"12998.44" forKey:@"cart_value"];
[checkoutDetails setObject:@"myapp://myapp/cart" forKey:@"deep_link"];
[[QGSdk sharedInstance] logEvent:@"checkout_initiated"
↳withParameters:checkoutDetails];
```

#### Product Rated:

```
NSMutableDictionary *productRated = [[NSMutableDictionary alloc] init];

[productRated setObject:@"1232" forKey:@"id"];
[productRated setObject:@"2" forKey:@"rating"];
[[QGSdk sharedInstance] logEvent:@"product_rated" withParameters:productRated];
```

#### Searched:

```
NSMutableDictionary *searchDetails = [[NSMutableDictionary alloc] init];
[searchDetails setObject:@"1232" forKey:@"id"];
[searchDetails setObject:@"Nikon Camera" forKey:@"name"];
[[QGSdk sharedInstance] logEvent:@"searched" withParameters:searched];
```

#### Reached Level:

```
NSMutableDictionary *level = [[NSMutableDictionary alloc] init];
[level setObject:@"23" forKey:@"level"];
[[QGSdk sharedInstance] logEvent:@"level" withParameters:level];
```

#### Your custom events

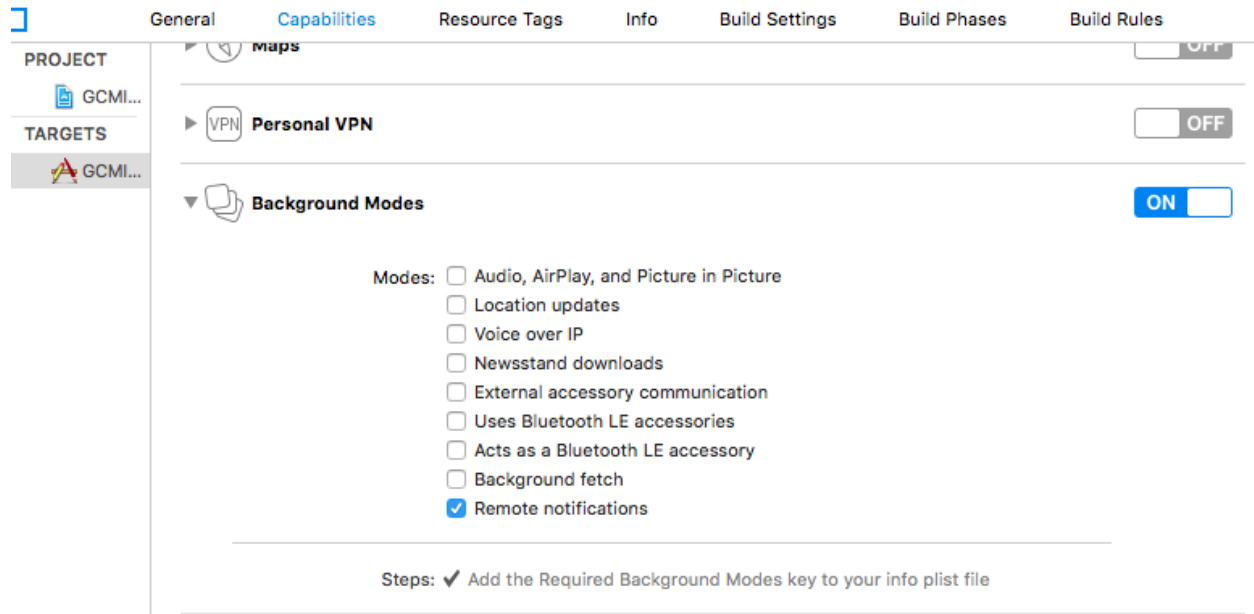
Apart from above predefined events, you can create your own custom events, and have custom parameters in them:

```
NSMutableDictionary *event = [[NSMutableDictionary alloc] init];
[event setObject:@"2" forKey:@"num_products"];
[event setObject:@"some_value" forKey:@"my_param"];
[event setObject:@"123" forKey:@"some_other_param"];
[[QGSdk sharedInstance] logEvent:@"my_custom_event" withParameters:event];
```

## 4.4 Using iOS SDK - Swift (3.0)

### 4.4.1 Change required for APNS Token and User Tracking

It is required by QG SDK that you enable *Background Mode* in the *Capabilities* section of the main app target. After enabling background modes, select **Remote Notification** as shown in the snapshot.



### 4.4.2 Adding bridging headers

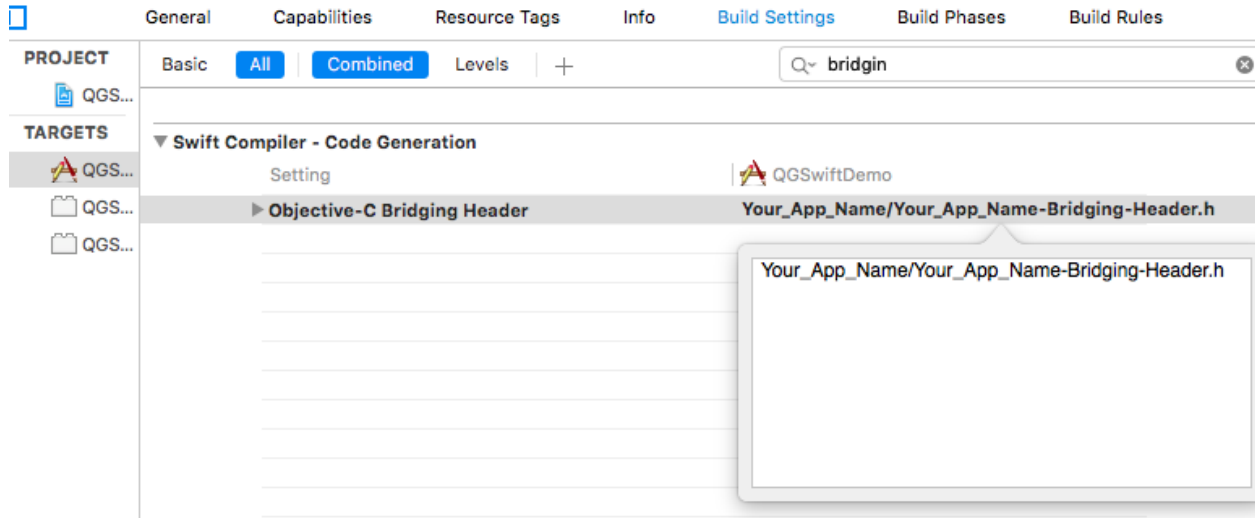
1. In Xcode, create the header file and name it by your product module name followed by `adding-Bridging-Header.h`. File name should look like `Project_Name-Bridging-Header.h`. Please make sure this header file is in root path of the project (although you can keep it anywhere).
2. Now Click on project tab to open *Build Settings*. In your project *target* -> *Build Setting*, search for Objective-C Bridging Header and add path of the `Project_Name-Bridging-Header.h`. (`Project_Name/Project_Name-Bridging-Header.h`)
3. Import SDK header file in the bridging header file. Your file should look like this:

```
#ifndef Project_Name_Bridging_Header_h
#define Project_Name_Bridging_Header_h
#import "QGSdk.h"
#endif /* Project_Name_Bridging_Header_h */
```

### 4.4.3 App Delegate Changes

In `didFinishLaunchingWithOptions` method of `AppDelegate`, initialise the sdk using `onStart()` method add the following code for registering for remote notification:

NOTE: Add `UserNotifications.framework` and import `UserNotifications` in `AppDelegate` for iOS 10 notification.



Also add `UNUserNotificationCenterDelegate` in `AppDelegate`. iOS 10 implementation is documented below:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions_
↳launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    let QG = QGSdk.getSharedInstance()
    QG?.onStart("<your_app_id>", setDevProfile: true)

    let settings = UIUserNotificationSettings(types: [.alert, .badge, .sound],_
↳categories: nil)
    UIApplication.shared.registerUserNotificationSettings(settings)

    return true
}
```

Note that `UIApplication.shared.registerForRemoteNotifications()` is called by our SDK for iOS 8 and above to track APNs Token for user tracking.

Just build and run the app to make sure that you receive a message that app would like to send push notification. If you get code signing error, make sure that proper provisioning profile is selected Add the following code in `AppDelegate.m` to get the device token for the user:

```
func application(_ application: UIApplication,_
↳didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
    let QG = QGSdk.getSharedInstance()
    print("My token is: \(deviceToken.description)")
    QG?.setToken(deviceToken as Data!)
}

func application(_ application: UIApplication,_
↳didFailToRegisterForRemoteNotificationsWithError error: Error) {
    print("Failed to get token, error: %@", error.localizedDescription)
}
```

#### 4.4.4 Handling Push Notification

Notifications are delivered while the app is in foreground, background or not running state. We can handle them in the following delegate methods.

If the remote notification is tapped, the system launches the app and the app calls its delegate's `didFinishLaunchingWithOptions:` method, passing in the notification payload (for remote notifications). Although `didFinishLaunchingWithOptions:` is not the best place to handle the notification, getting the payload at this point gives you the opportunity to start the update process before your handler method is called.

For remote notifications, the system also calls the `didReceiveRemoteNotification:fetchCompletionHandler:` method of the app delegate.

The notification is delivered when the app is running in the foreground. The app calls the `application:didReceiveRemoteNotification:fetchCompletionHandler:` method of the app delegate. This method is called if the app is running in background or suspended state. (If `application:didReceiveRemoteNotification:fetchCompletionHandler:` is not implemented, the system calls `application:didReceiveRemoteNotification:.`) However, it is advised to use `application:didReceiveRemoteNotification:fetchCompletionHandler:` method to handle push notification.

Implementation:

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable : Any], fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {
    let QG = QGSdk.sharedInstance()
    // to enable track click on notification
    QG?.application(application, didReceiveRemoteNotification: userInfo)
    completionHandler(UIBackgroundFetchResult.noData)
}
```

You can also handle background operation using the above method once remote notification is delivered. For this make sure, wake app in background is selected while creating a campaign to send the notification. Also, enable *BACKGROUND MODE* in capabilities and select Remote Notification.

**Background Modes**

ON

**Modes:**

- ☐ Audio, AirPlay, and Picture in Picture
- ☐ Location updates
- ☐ Voice over IP
- ☐ Newsstand downloads
- ☐ External accessory communication
- ☐ Uses Bluetooth LE accessories
- ☐ Acts as a Bluetooth LE accessory
- ☒ Background fetch
- ☒ Remote notifications

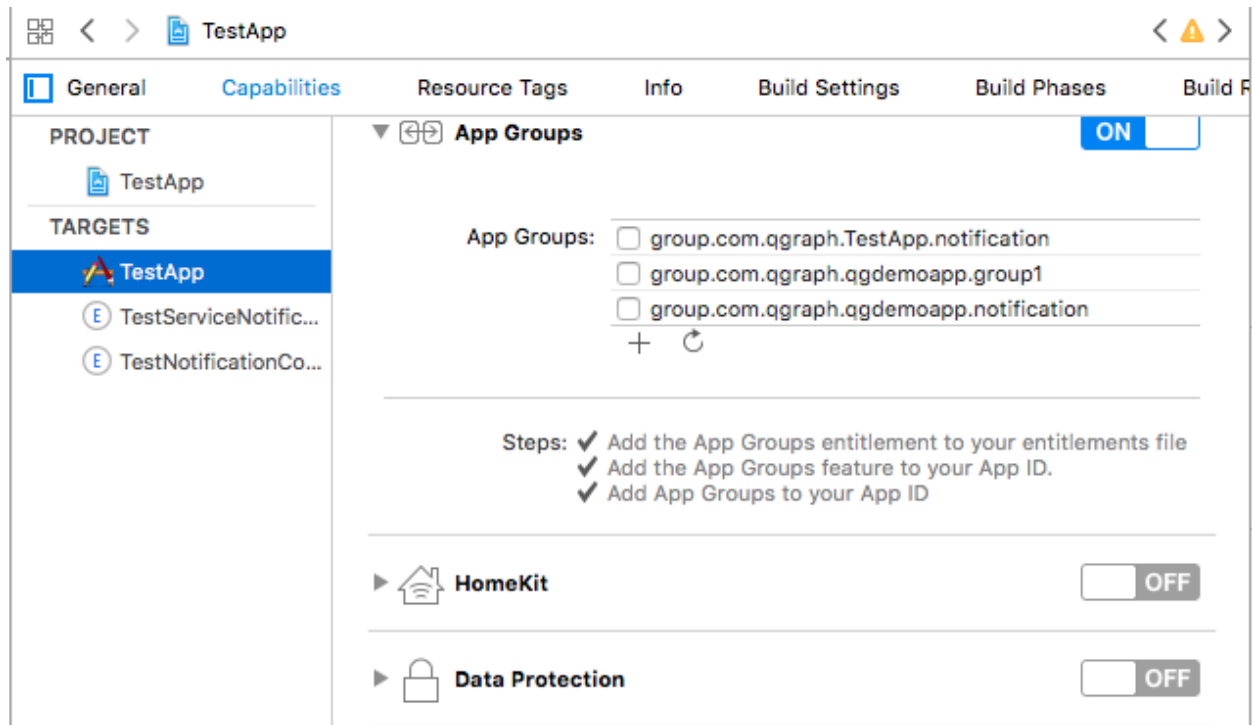
**Steps:** ✓ Add the Required Background Modes key to your info plist file

If you have implemented `application:didReceiveRemoteNotification:` add method `QGSdk.sharedInstance().application(application, didReceiveRemoteNotification: userInfo)` inside it. Your implementation should look like:

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable : Any]) {
    let QG = QGSdk.getSharedInstance()
    // to enable track click on notification
    QG?.application(application, didReceiveRemoteNotification: userInfo)
}
```

#### 4.4.5 Changes for iOS 10

Your basic integration for iOS 8 and 9 is complete. From iOS 10 and above two new frameworks has been introduced for notifications. For integrating QGraph notification SDK, you need to add Capabilities *APP GROUPS*. Go to *Project > Main Target > Capabilities*. Check on App Groups and add a group as below. Use your bundle id to create App Group. For example, if your bundle id is `com.company.appname`, App Group could be `group.com.company.appname.xyz`.



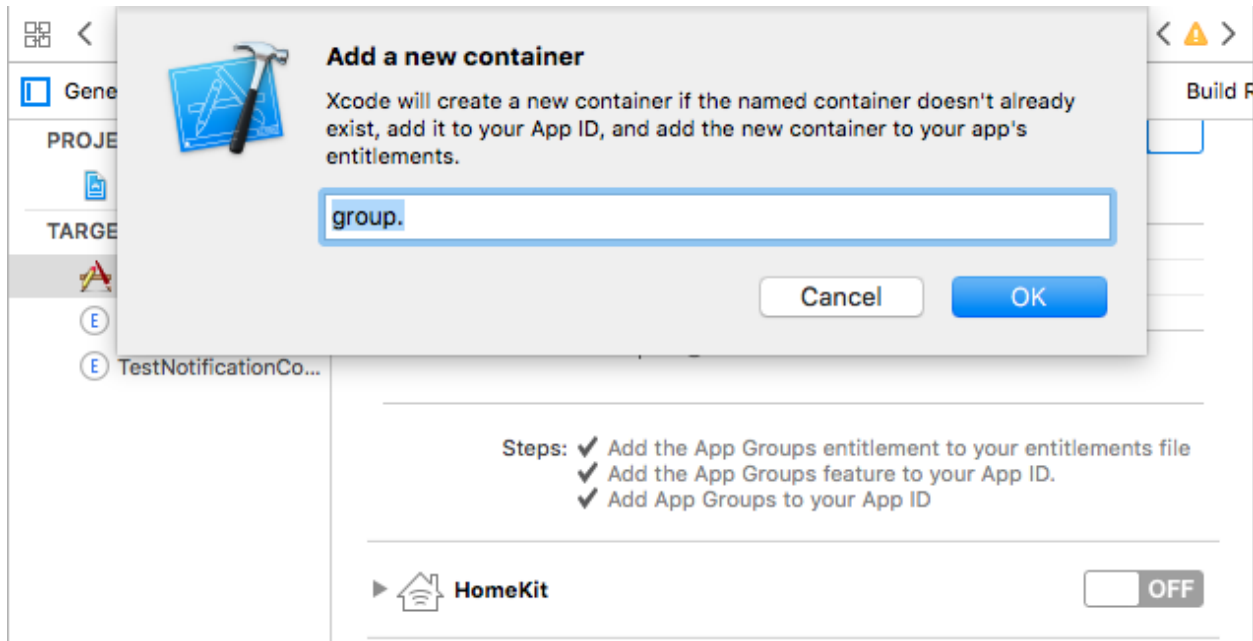
You need App Group so that data can be shared between extensions. Use that App Group name in `onStart:withAppGroup:setDevProfile:` in App Delegate.

#### 4.4.6 AppDelegate Changes for Swift Apps for iOS 10

Add framework `UserNotifications` to app target and import in app delegate. Also add `UNNotificationCenterDelegate` in it:

```
import UIKit
import UserNotifications
```

(continues on next page)



(continued from previous page)

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate,
↳ UNUserNotificationCenterDelegate {

    var window: UIWindow?

    let APP_GROUP = "group.com.company.product.extension"

    func application(_ application: UIApplication, didFinishLaunchingWithOptions_
↳ launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.

        let QG = QGSdk.getSharedInstance()
        QG?.onStart("<your_app_id>", withAppGroup: APP_GROUP, setDevProfile: true)

        if #available(iOS 10.0, *) {
            let center = UNUserNotificationCenter.current()
            center.delegate = self

            // adding category for QGraph Carousel and Slider Push
            let categories = NSSet(object: QG!.
↳ getQGSlderPushActionCategory(withNextButtonTitle: nil, withOpenAppButtonTitle:
↳ nil)) as! Set<UNNotificationCategory>
            center.setNotificationCategories(categories)

            center.requestAuthorization(options: [.badge, .carPlay, .alert, .sound])
↳ { (granted, error) in
                print("Granted: \(granted), Error: \(error)")
            }

        } else {
            // Fallback on earlier versions
            let settings = UIUserNotificationSettings(types: [.alert, .badge, .sound],
↳ categories: nil)
```

(continues on next page)



(continued from previous page)

```

        UIApplication.shared.registerUserNotificationSettings(settings)
    }

    return true
}
}

```

**NOTE:** If you have your own existing notification action category for iOS 10, you can add it along with QGraph *CAROUSEL/SLIDER* category. For the carousel and slider push action buttons, you can also specify button titles. Next button will be used to animate the carousel/slider and Open App Button will open the app with deeplink if any.

## 4.4.7 Handling Push Notification in iOS 10

There are new delegate methods introduced in iOS 10 to track notification and display in foreground state as well. To track notifications in background state, you need to enable background mode in the capabilities. Above all these you need to activate push notification in the capabilities. This will add entitlement files to your app target.

1. You might have already included this method. Please make sure `QGSdk.sharedInstance().application(application, didReceiveRemoteNotification: userInfo)` is added in it. It is required to track notifications:

```

func application(_ application: UIApplication, didReceiveRemoteNotification_
↳userInfo: [AnyHashable : Any], fetchCompletionHandler completionHandler:
↳@escaping (UIBackgroundFetchResult) -> Void) {
    let QG = QGSdk.sharedInstance()
    // to enable track click on notification
    QG?.application(application, didReceiveRemoteNotification: userInfo)
    completionHandler(UIBackgroundFetchResult.noData)
}

```

2. The below method will be called on the delegate only if the application is in the foreground. If the method is not implemented or the handler is not called in a timely manner then the notification will not be presented. The application can choose to have the notification presented as a sound, badge, alert and/or in the notification list. This decision should be based on whether the information in the notification is otherwise visible to the user:

```

@available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent_
↳notification: UNNotification, withCompletionHandler completionHandler:
↳@escaping (UNNotificationPresentationOptions) -> Void) {
    QGSdk.sharedInstance().userNotificationCenter(center, willPresent:
↳notification)
    completionHandler([.alert, .badge, .sound]);
}

```

3. The method will be called on the delegate when the user responded to the notification by opening the application, dismissing the notification or choosing a *UNNotificationAction*. The delegate must be set before the application returns from `applicationDidFinishLaunching:`.

**NOTE:** This method is specifically required for carousel and slider push to work. Also used to track notification\_clicked event for QGraph push:

```

@available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response:
↳UNNotificationResponse, withCompletionHandler completionHandler: @escaping () ->
↳Void) {

```

(continues on next page)

(continued from previous page)

```

    QGSdk.getSharedInstance().userNotificationCenter(center, didReceive: response)
    completionHandler()
}

```

#### 4.4.8 Handling Deeplink for QGraph Push

For Push notifications deeplinks should be handled in the method *didReceiveNotificationResponse:withCompletionHandler:* as described below. You can get the deeplink url and then pass it to *openUrl:* and then you should get a callback in the *application:openUrl:options* where you can handle the opening of a specific page.

```

@available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response:
↳UNNotificationResponse, withCompletionHandler completionHandler: @escaping () ->
↳Void) {
    let userInfo = response.notification.request.content.userInfo
    if (userInfo["deepLink"] != nil) {
        let url = URL.init(string: userInfo["deepLink"] as! String)
        DispatchQueue.main.async {
            UIApplication.shared.openURL(url!)
        }
    }

    QGSdk.getSharedInstance().userNotificationCenter(center, didReceive: response)
    completionHandler()
}

```

For any deeplink specified in In-App campaigns, you should get a callback in the below method. You need to handle it on your own to open any specific page.

```

func application(_ app: UIApplication, open url: URL, options:
↳[UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
    print("deeplink called")
    return true
}

```

Finally, after adding all the above methods your app delegate should look like:

```

import UIKit
import UserNotifications

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate,
↳UNUserNotificationCenterDelegate {

    var window: UIWindow?

    let APP_GROUP = "group.com.company.product.extension"

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
↳launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.

        let QG = QGSdk.getSharedInstance()

```

(continues on next page)

(continued from previous page)

```

    QG?.onStart("<your_app_id>", withAppGroup: APP_GROUP, setDevProfile: true)

    if #available(iOS 10.0, *) {
        let center = UNUserNotificationCenter.current()
        center.delegate = self

        let categories = NSSet(object: QG!.
→getQGSliderPushActionCategory(withNextButtonTitle: nil, withOpenAppButtonTitle:
→nil)) as! Set<UNNotificationCategory>
        center.setNotificationCategories(categories)

        center.requestAuthorization(options: [.badge, .carPlay, .alert, .sound])
→{ (granted, error) in
            print("Granted: \(granted), Error: \(error)")
        }

        } else {
            // Fallback on earlier versions
            let settings = UIUserNotificationSettings(types: [.alert, .badge, .sound],
→ categories: nil)
            UIApplication.shared.registerUserNotificationSettings(settings)
        }

        return true
    }

    func application(_ application: UIApplication,
→didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        let QG = QGSdk.getSharedInstance()
        print("My token is: \(deviceToken.description)")
        QG?.setToken(deviceToken as Data!)
    }

    func application(_ application: UIApplication,
→didFailToRegisterForRemoteNotificationsWithError error: Error) {
        print("Failed to get token, error: %@", error.localizedDescription)
    }

    func application(_ application: UIApplication, didReceiveRemoteNotification
→userInfo: [AnyHashable : Any], fetchCompletionHandler completionHandler: @escaping
→(UIBackgroundFetchResult) -> Void) {
        let QG = QGSdk.getSharedInstance()
        // to enable track click on notification
        QG?.application(application, didReceiveRemoteNotification: userInfo)
        completionHandler(UIBackgroundFetchResult.noData)
    }

    @available(iOS 10.0, *)
    func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive
→response: UNNotificationResponse, withCompletionHandler completionHandler:
→@escaping () -> Void) {
        QGSdk.getSharedInstance().userNotificationCenter(center, didReceive: response)
        completionHandler()
    }

    @available(iOS 10.0, *)

```

(continues on next page)

(continued from previous page)

```
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent:
↳ notification: UNNotification, withCompletionHandler completionHandler: @escaping
↳ (UNNotificationPresentationOptions) -> Void) {
    QGSdk.sharedInstance().userNotificationCenter(center, willPresent:
↳ notification)

    completionHandler([.alert, .badge, .sound]);
}

func application(_ app: UIApplication, open url: URL, options:
↳ [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
    print("deeplink called")
    return true
}
}
```

#### 4.4.9 Adding Extensions for iOS Push with Attachment and QGraph Carousel and Slider Push

In iOS 10, two frameworks has been introduced for handling push notification with content. You can have a push notification with image, gif, audio and video. Apart from that you can also have your custom UI for notifications. For this, payload can be modified and used to download content before the notification is drawn. You simply need to follow the below steps to add two of the extensions targets for handling these notifications: Service Extension and Content Extension. Before proceeding make sure to download all the QGraph files to be used here. You should have these files with you

1. QGNotificationSdk
2. QGNotificationServiceExtension
3. QGNotificationContentExtension

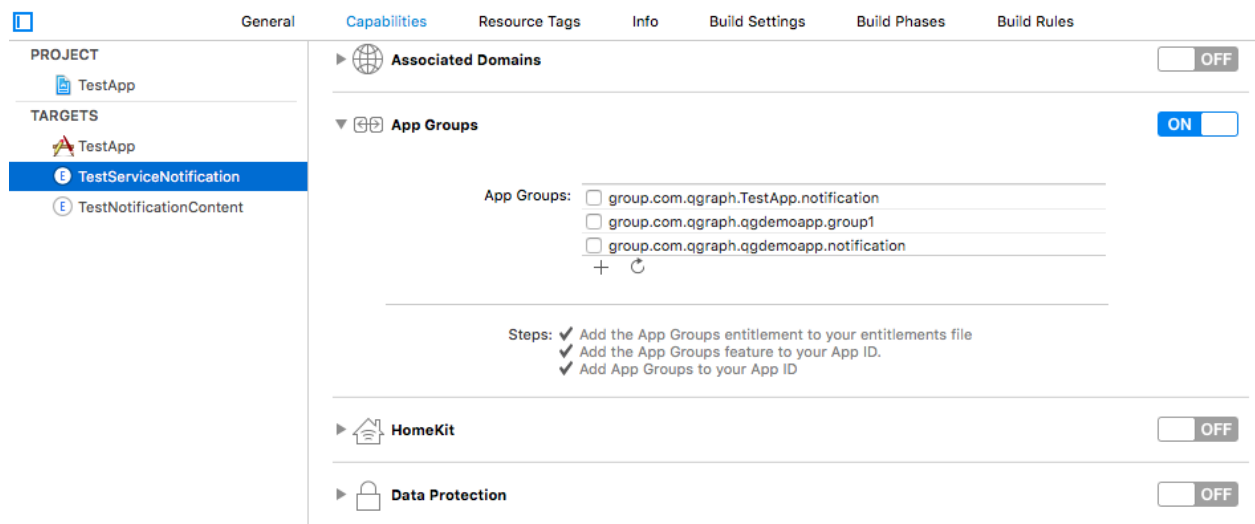
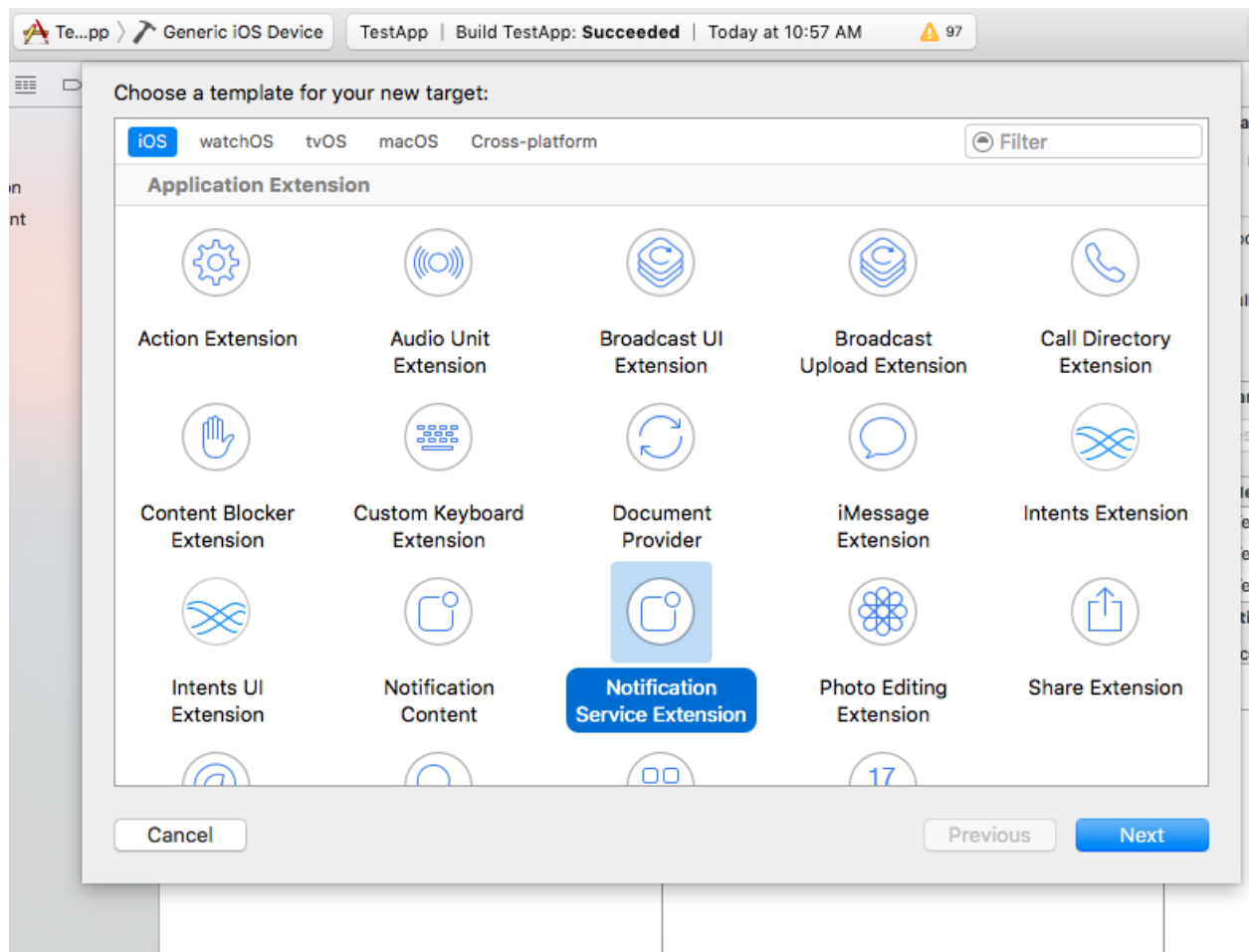
**NOTE:** These files are to be used with service and content extensions only. Do not add them to main app target.

#### 4.4.10 Notification Service Extension

Service extension is basically the target extension where you get a callback when a push is delivered to the device. You can download and create attachments here. If you fail to download the content and pass it to contentHandler within certain time, default standard notification will be drawn.

#### 4.4.11 Adding Service extension

1. Add an iOS target and choose Notification Service extension and proceed. Add a product name and Finish. When created you will be **prompted to activate the target**. Once activated, you can see 2 files added, NotificationService.swift and Info.plist in the created target.
2. Delete the NotificationService.swift file from the service extension target.
3. Add file NotificationService.swift from downloaded folder *QGNotificationServiceExtension*
4. Go to project navigator and select the *Service Extension Target*
5. Select *Capabilities* and check on *App Group* and select the *APP GROUP* which you added to your main app target.

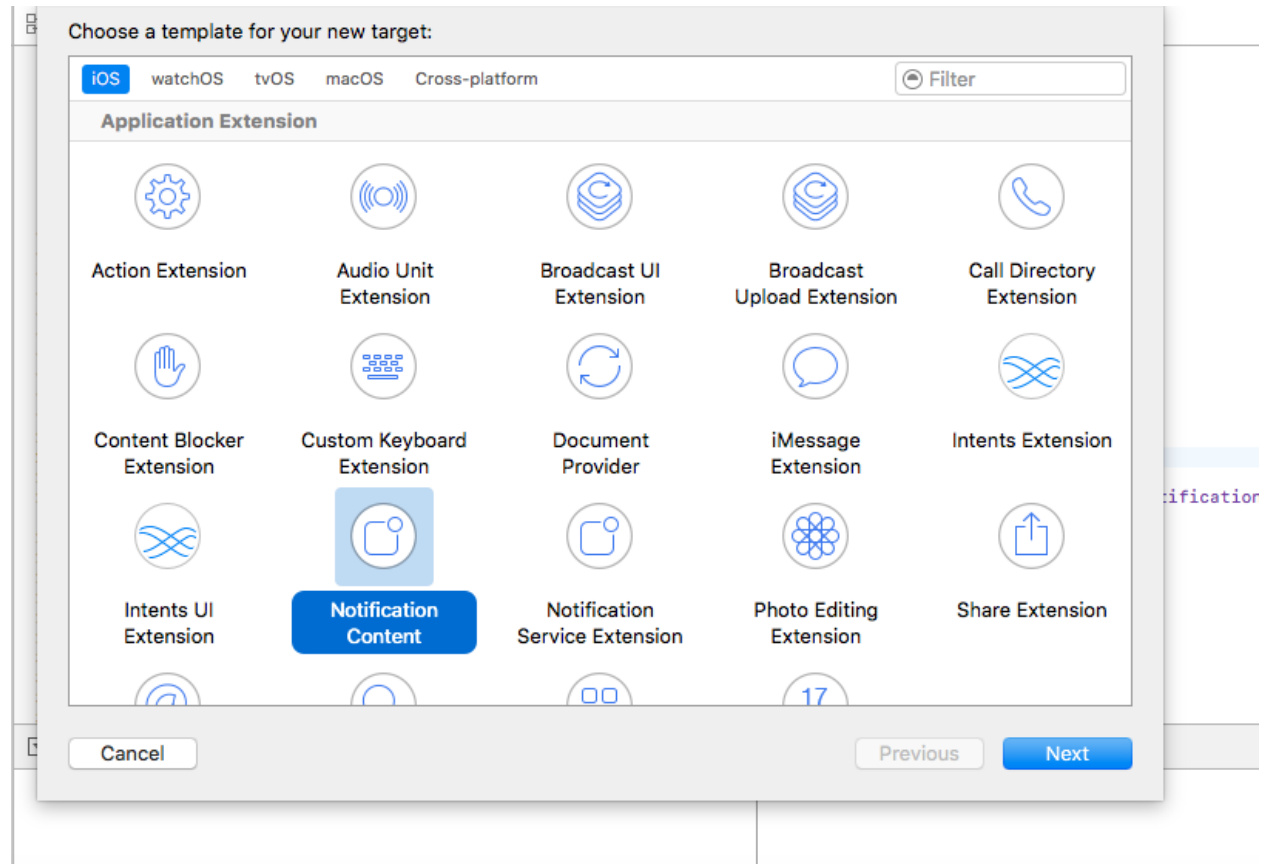


- Go to `NotificationService.swift` and change your app group:

```
let APP_GROUP = "group.com.company.product.extension"
```

#### 4.4.12 Adding Content Extension

- Add an iOS target and choose Notification Content extension and proceed. Add a product name and Finish. When created you will be **prompted to activate the target**. Once activated, you can see 3 files added, *NotificationViewController.swift*, *MainInterface.storyboard* and *Info.plist*.



- Delete *NotificationViewController.swift* and *MainInterface.storyboard*.
- Add files *NotificationViewController.swift* and *MainInterface.storyboard* from downloaded folder *QGNotificationContentExtension*.
- As done above, enable App Groups and select the same app group through capabilities of the content extension target.
- Go to *NotificationViewController.m* and change your app group:

```
let APP_GROUP = "group.com.company.product.extension"
```

- Go to *Info.plist* and add **UNNotificationExtensionDefaultContentHidden** (Boolean) - YES and **UNNotificationExtensionCategory** (string) - **QGCAROUSEL** in `NSExtensionAttributes` dict of `NSExtension` dict as shown in the screenshot.
- Add *QuartzCore.framework* in this target.

Key	Type	Value
▼ Information Property List	Dictionary	(10 items)
Localization native development region	String	en
Bundle display name	String	TestNotificationContent
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	XPC!
Bundle versions string, short	String	1.0
Bundle version	String	1
▼ NSExtension	Dictionary	(3 items)
▼ NSExtensionAttributes	Dictionary	(3 items)
UNNotificationExtensionDefaultContentHidden	Boolean	YES
UNNotificationExtensionCategory	String	QGCAROUSEL
UNNotificationExtensionInitialContentSizeRatio	Number	1
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.usernotifications.content-extension

8. Add QGNotificationSdk to both extension targets. Do not add it to main app target.

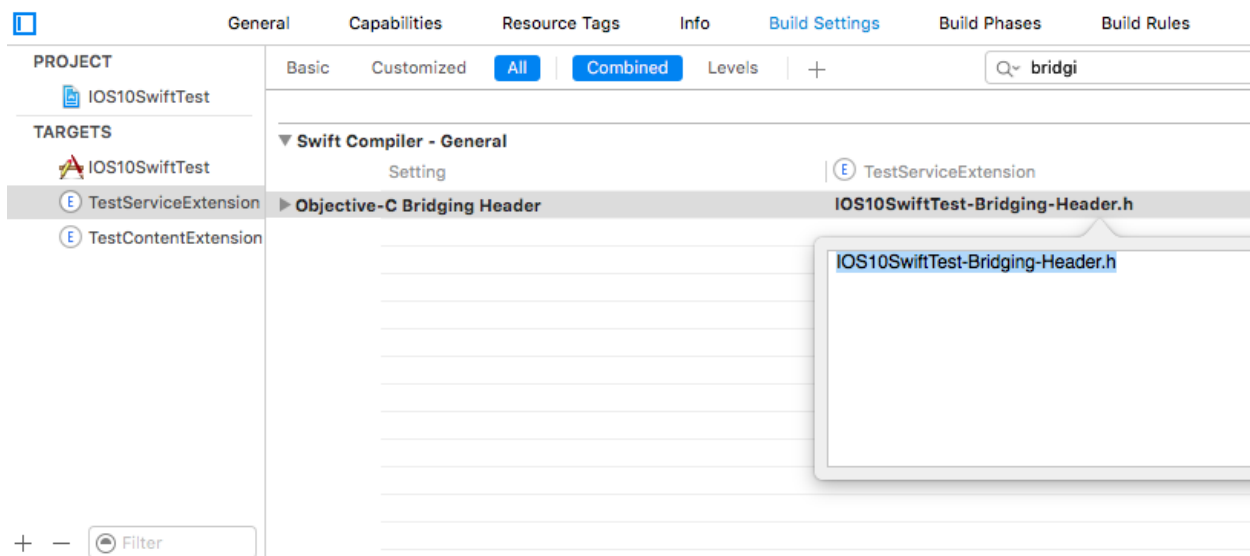
NOTE:

1. Please make sure **APP\_GROUP** used in all the three targets are same.
2. Set the deployment target to 10.0 in both the extensions.
3. Remove **-ObjC/\$(inherited)** (if it exists) from build settings of service and content extension targets.

**IMP:** You need to add *QGNotificationSdk.h* and *iCarousel.h* in Bridging-Header, so that these objective C files can be used in your extension targets.

Go to *Project -> Extension Targets -> Build Setting -> Objective-C Bridging Header*

Add the path to your bridging-header-file similar to Main App Target.



Note: Add bridging-header-file in any one of the extensions (service extension or content extension) and then add the file path of bridging header in both the extensions.

### 4.4.13 Click Through and View Through Attribution

QGraph SDK attributes events for each notification clicked or viewed. Events are attributed on the basis of time interval specified for all log events.

Currently, click through attribution works for push notification clicked (sent via QGraph) and InApp notification clicked. View through attribution works only in the case of InApp notifications.

By default click through attribution window (time interval) is set to 86400 seconds (24 hrs) and view through attribution window is set to 3600 seconds (1 hr). You can change this window any time using following apis:

```
// to set click through attribution window
- (void)setClickAttributionWindow:(NSInteger)seconds;
// to set view through attribution window
- (void)setAttributionWindow:(NSInteger)seconds;
```

To set a custom value, pass the time interval in seconds. e.g.: to set click attribution window to be 12 hrs:

```
QGSdk.sharedInstance().setClickAttributionWindow(43200)
```

To disable any of the click through or view through attribution, pass the value 0. E.g.:

```
QGSdk.sharedInstance().setAttributionWindow(0)
```

### 4.4.14 Configuring Batching

Our SDK batches the network requests it makes to QGraph server, in order to optimize network usage. By default, it flushes data to the server every 15 seconds in release builds, and every second in debug builds. This interval is configurable using the following method:

```
QGSdk.sharedInstance().flushInterval = <flush interval in seconds>
```

Further, you can force the SDK to flush the data to server any time by calling the following function:

```
QGSdk.sharedInstance().flush()
```

Furthermore, you can invoke a completion handler after flush using function:

```
QGSdk.sharedInstance().flush(completion: {
    //some method
})
```

### 4.4.15 Matching mobile app users with mobile web users

Our SDK can help you track your mobile app users across your app and mobile web. If you want to enable this functionality, you need to add **Safari Services Framework** in your app.

If you have added Safari Services Framework in your app, but would like to disable our tracking, use the following function:



```
QGSdk.sharedInstance().disableUserTrackingForSafari()
```

#### 4.4.16 In app Notification

QGraph SDK supports InApp notification starting in sdk version 2.0.0. InApp notification are supported in two types: Textual and Image. Visit your QGraph account to create InApp Campaigns.

These notifications are shown based on the log events app sends through our sdk and the matching conditions of the InApp Campaigns. Make sure to send appropriate log event (with parameter or valueToSum if any) for InApp notifications to work.

By default, InApp notifications are enabled. You can enable/disable it anytime using following method in the sdk:

```
- (void)disableInAppCampaigns:(BOOL)disabled;
```

eg. to disable:

```
QGSdk.sharedInstance().disable(inAppCampaigns: true)
```

Disabling it will restrict the device to get any new InApp campaigns. It will also disable InApp notification to be drawn.

For All InApp Notification, you can configure a deep link url from the dashboard while creating an InApp campaign.

There is tap event defined on textual and image InApps. When the user taps on text on textual InApp or clicks on image in the image InApp and if there is a valid deep link setup, you will get a call back in your *AppDelegate.m* in the following method:

```
func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool
```

Here you can implement your deep link with the url.

#### 4.4.17 Registering Your Actionable Notification Types

Actionable notifications let you add custom action buttons to the standard iOS interfaces for local and push notifications. Actionable notifications give the user a quick and easy way to perform relevant tasks in response to a notification. Prior to iOS 8, user notifications had only one default action. In iOS 8 and later, the lock screen, notification banners, and notification entries in Notification Center can display one or two custom actions. Modal alerts can display up to four. When the user selects a custom action, iOS notifies your app so that you can perform the task associated with that action.

For defining a notification action and its category, and to handle actionable notification, please refer the description in the apple docs. ([please click here](#))

Action Category can be set in the dashboard while sending notification. While configuring to send notification through campaigns, use the categories defined in the app.

#### 4.4.18 Logging user profile information

User profiles are information about your users, like their name, city, date of birth or any other information that you may wish to track. You log user profiles by using one or more of the following functions:

```
- (void)setUserId:(NSString *)userId;
```

Other methods you may use to pass user profile parameters to us:

```
- (void)setUserId:(NSString *)userId;
- (void)setName:(NSString *)name;
- (void)setFirstName:(NSString *)name;
- (void)setLastName:(NSString *)name;
- (void)setCity:(NSString *)city;
- (void)setEmail:(NSString *)email;
- (void)setDayOfBirth:(NSNumber *)day;
- (void)setMonthOfBirth:(NSNumber *)month;
- (void)setYearOfBirth:(NSNumber *)year;
```

Other than these method, you can log your own custom user parameters. You do it using:

```
- (void)setCustomKey:(NSString *)key withValue:(id) value;
```

For example, you may wish to have the user's current rating like this:

```
QGSdk.sharedInstance().setCustomKey("current rating", withValue: "123")
```

## 4.4.19 Logging events information

Events are the activities that a user performs in your app, for example, viewing the products, playing a game or listening to a music. Each event has follow properties:

1. Name. For instance, the event of viewing a product is called `product_viewed`.
2. Optionally, some parameters. For instance, for event `product_viewed`, the parameters are `id`(the id of the product viewed), `name` (name of the product viewed), `image_url` (image url of the product viewed), `deep_link` (a deep link which takes one to the product page in the app), and so on.
3. Optionally, a “value to sum”. This value will be summed up when doing campaign attribution. For instance, if you pass this value in your checkout completed event, you will be able to view stats such as a particular campaign has been responsible to drive Rs 84,000 worth of sales. You log events using the function `logEvent()`. It comes in four variations.

```
(void)logEvent:(NSString *)name

(void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters

(void)logEvent:(NSString *)name withValueToSum:(NSNumber *) valueToSum

(void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters_
↪withValueToSum:(NSNumber *) valueToSum
```

Once you log event information to use, you can segment users on the basis of the events (For example, you can create a segment consisting of users have not launched for past 7 days, or you can create a segment consisting of users who, in last 7 days, have purchased a product whose value is more than \$1000)

You can also define your events, and your own parameters for any event. However, if you do that, you will need to sync up with us to be able to segment the users on the basis of these events or customize your creatives based on these events.

You can use the following method to pass event information to us:

```
- (void)logEvent:(NSString *)name withParameters:(NSDictionary *)parameters;
```

Here is how you set up some of the popular events.

#### 4.4.19.1 Registration Completed

This event does not have any parameters:

```
QGSdk.sharedInstance().logEvent("registration_completed", withParameters: nil)
```

#### 4.4.19.2 Category Viewed

This event has one parameter:

```
let categoryDetails = ["category": "apparels"]
QGSdk.sharedInstance().logEvent("category_viewed", withParameters: categoryDetails)
```

#### 4.4.19.3 Product Viewed

You may choose to have the following fields:

```
var productDetails : [String:String] = [:]
productDetails["id"] = "123"
productDetails["name"] = "Nikon Camera"
productDetails["image_url"] = "http://mysite.com/products/123.png"
productDetails["deep_link"] = "myapp://products?id=123"
productDetails["color"] = "black"
productDetails["category"] = "electronics"
productDetails["size"] = "small"
productDetails["price"] = "6999"
QGSdk.sharedInstance().logEvent("product_viewed", withParameters: productDetails)
```

#### 4.4.19.4 Product Added to Wishlist

```
var productDetails : [String:String] = [:]
productDetails["id"] = "123"
productDetails["name"] = "Nikon Camera"
productDetails["image_url"] = "http://mysite.com/products/123.png"
productDetails["deep_link"] = "myapp://products?id=123"
productDetails["color"] = "black"
productDetails["category"] = "electronics"
productDetails["size"] = "small"
productDetails["price"] = "6999"
QGSdk.sharedInstance().logEvent("product_added_to_wishlist", withParameters:↳
↳productDetails)
```

#### 4.4.19.5 Product Purchased

```
var productDetails : [String:String] = [:]
productDetails["id"] = "123"
productDetails["name"] = "Nikon Camera"
productDetails["image_url"] = "http://mysite.com/products/123.png"
productDetails["deep_link"] = "myapp://products?id=123"
productDetails["color"] = "black"
productDetails["category"] = "electronics"
productDetails["size"] = "small"
productDetails["price"] = "6999"
```

and then:

```
QGSdk.getSharedInstance().logEvent("product_purchased", withParameters:↳
↳productDetails)
```

or:

```
QGSdk.getSharedInstance().logEvent("product_purchased", withParameters:↳
↳productDetails, withValueToSum: price)
```

#### 4.4.19.6 Checkout Initiated

```
var checkoutDetails : [String:String] = [:]
checkoutDetails["num_products"] = "2"
checkoutDetails["cart_value"] = "12998.44"
checkoutDetails["deep_link"] = "myapp://myapp/cart"
QGSdk.getSharedInstance().logEvent("checkout_initiated", withParameters:↳
↳checkoutDetails)
```

#### 4.4.19.7 Product Rated

```
var productRated : [String:String] = [:]
productRated["id"] = "1232"
productRated["rating"] = "2"
QGSdk.getSharedInstance().logEvent("product Rated", withParameters: productRated)
```

#### 4.4.19.8 Searched

```
var searchDetails : [String:String] = [:]
searchDetails["id"] = "1232"
searchDetails["name"] = "2"
QGSdk.getSharedInstance().logEvent("searched", withParameters: searchDetails)
```

#### 4.4.19.9 Reached Level

```
let level = ["level" : "23"]
QGSdk.getSharedInstance().logEvent("level", withParameters: level)
```

#### 4.4.19.10 Your custom events

Apart from above predefined events, you can create your own custom events, and have custom parameters in them:

```
var event : [String:String] = [:]
event["num_products"] = "2"
event["my_param"] = "some_value"
event["some_other_param"] = "123"
QGSdk.getSharedInstance().logEvent("my_custom_event", withParameters: event)
```



### 5.1 Background and Terminology

For easier integration, first let us understand some fundamentals. This description holds true for Google Chrome and Firefox browsers. Fundamentals are the same for Safari, though the details differ.

Browsers support webpush only for HTTPS sites. We provide a workaround if you have HTTP site by providing you with a backing HTTPS site.

We give you two files: `qg-service-worker.js` and `manifest.json` and a snippet of javascript code. You install the files in the root folder of your web server and put javascript code in various web pages of your website. Our tag downloads some more javascript code from our servers.

When a user comes to your website, our code asks the browser to request the user to grant us the permission to send her web push notifications. If the user agrees, the browser returns us a token (you can think of it like an address of the browser) which the code transfers to QuantumGraph servers. Using this token, QuantumGraph servers can send the web push notification to users.

Our SDK (which gets downloaded from the javascript code that we provide you) also communicates to QuantumGraph servers the URLs the users are accessing. As a result, you can, using our web panel, send a push to the users who have browsed a particular URL. Using our SDK, you can also send us other attributes of the user (such as user id, email, city) and then send notifications based on those attributes.

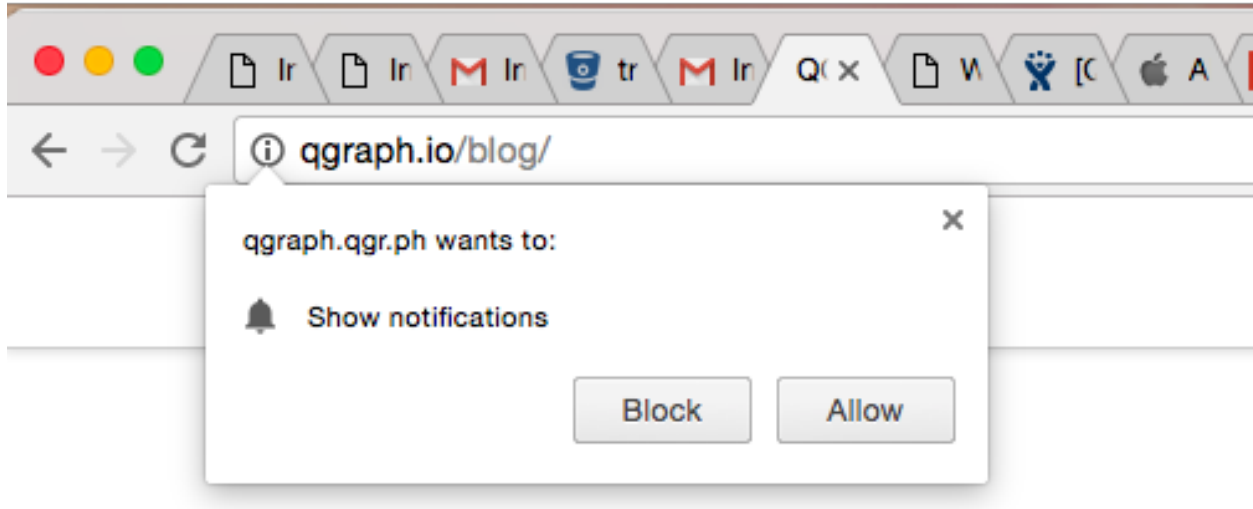
To better understand the documentation that follows, it is useful to understand some terminology used later.

#### **System Prompt**

System prompt is that prompt that appears when the browser requests the user to grant a certain website permission to show her webpush. It looks like this:

If the user clicks “Allow” then the browser gives a device token to the QG SDK. If the user clicks “Block” then the browsers does not give the device token to the QG SDK.

#### **Fake prompt**



If you display a system prompt to the user and she blocks your website, then the browser does not allow you to display the system prompt again (unless the user explicitly modifies the notification settings by herself). Thus, before displaying the system prompt, it may be good to ask for a “pre approval”, by displaying a fake prompt. If the user allows the notifications in the fake prompt, only then you display the system prompt. If the user disallows the notifications in the fake prompt, you can show the fake prompt after some time (1 hr, 1 day, or more) and request the user again.

This is what a fake prompt looks like. You can customize the text as per your needs.

### Overlay

When displaying the system prompt, you may wish to deemphasize the content of your website, to direct the user attention to the system prompt. This may be done by an overlay screen, which looks like this. You can customize the text as per your needs.

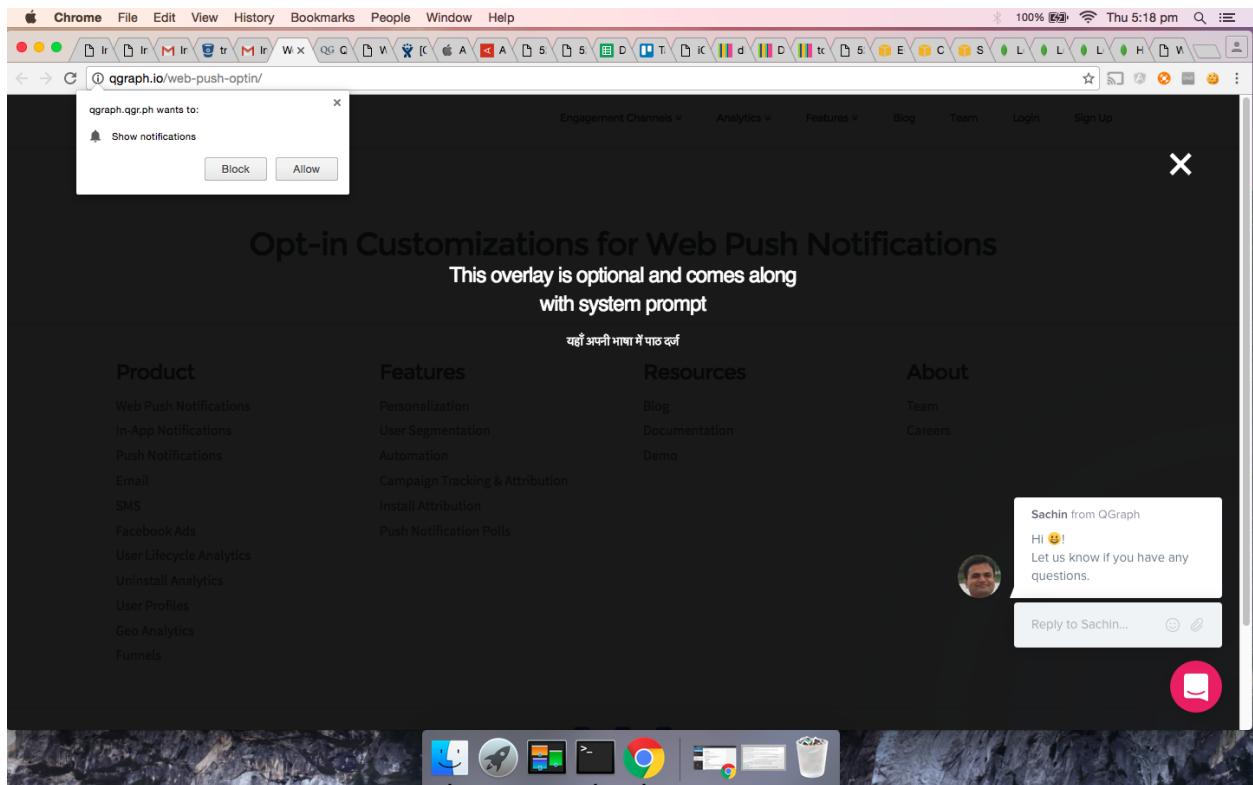
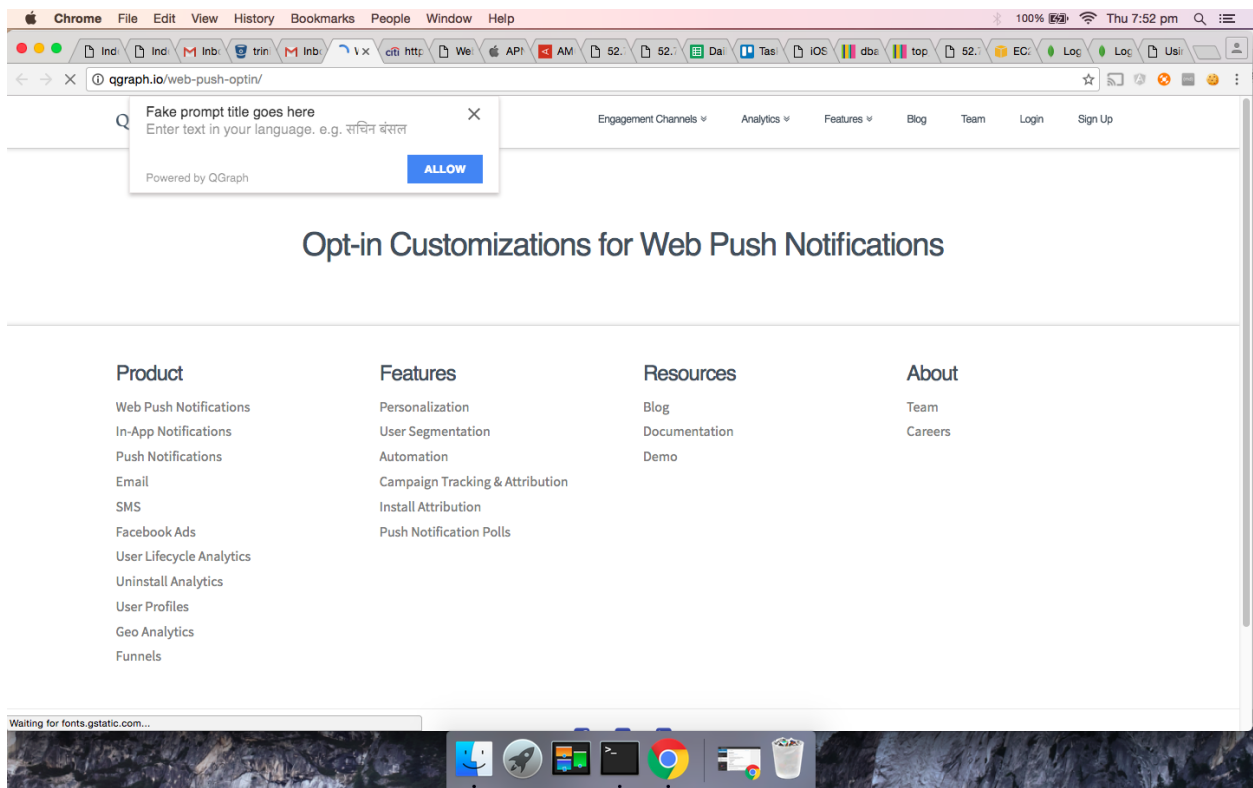
### Thank you prompt

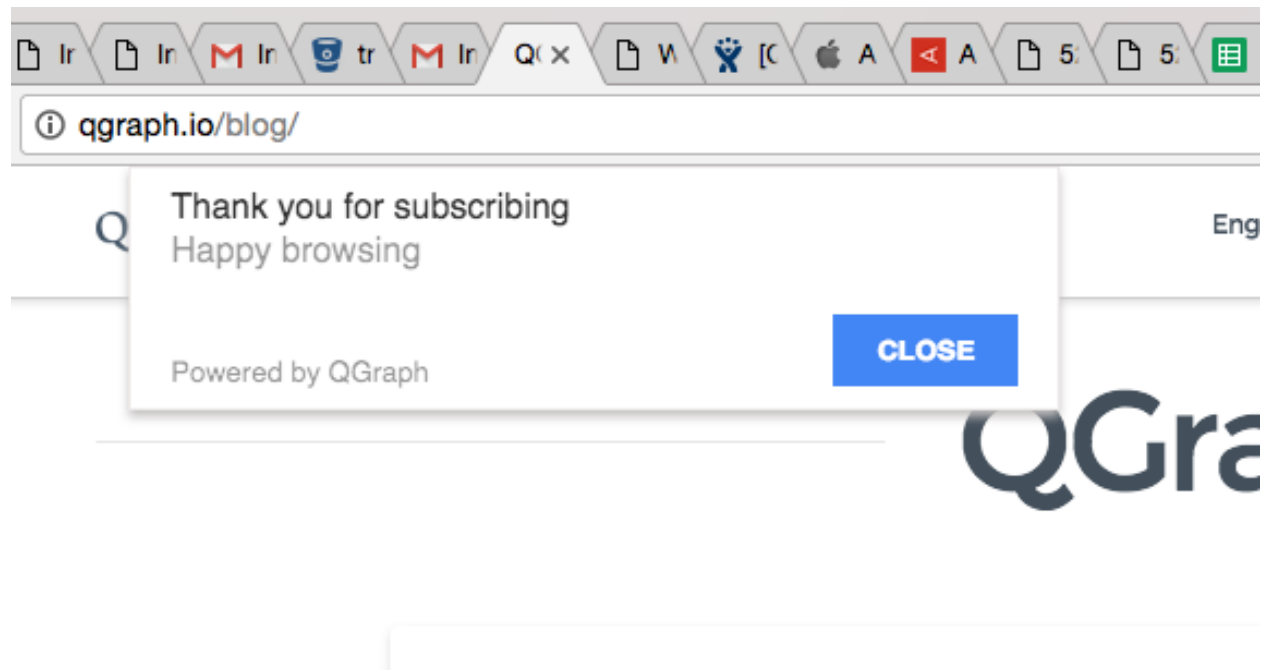
You can display a thank you message once the user has granted permission to send the web push. You can customize this message, and change its location. It looks like this.

## 5.2 Installing Web Pixel

Here we describe how you can integrate QGraph’s pixel on your website. While testing web pixel, you should make sure that you are *not* in incognito mode of web browser, since web push functionality does not work in incognito mode.







### 5.2.1 If your site is HTTPS

For HTTPS integration, you can also refer to this video:

1. Login to <http://app.qgraph.io>
2. Go to “Setup” -> “Integrations” section on the side bar menu. Select “Web” from the three channels presented to you.
3. Enter your website’s URL and follow the instructions provided. In step 2 of integration, you will be provided your app id. Keep a note of it.
4. Download the following files
  - (a) `https://cdn.qgr.ph/qg-sw.<your app id>.js`
  - (b) `https://cdn.qgr.ph/manifest.json`

Copy above files to the root folder of your website. To make sure that you have done this correctly, verify if following files are accessible

- (a) `https://yourwebsite.com/qg-sw.<your app id>.js`
- (b) `https://yourwebsite.com/manifest.json`

In case you want to use your GCM key, put your sender id in `manifest.json` above instead of the sender already present in the file. In this case you also need to enter your GCM key during the intergration. (We need this GCM key to send web push notifications)

Add the following lines inside of head or body tag of your web page:

```
<script type="text/javascript">
!function(q,g,r,a,p,h,js){
  if(q.qg)return;
  js = q.qg = function() {
    js.callmethod ? js.callmethod.call(js, arguments) : js.queue.push(arguments);
  };
}
```

(continues on next page)

(continued from previous page)

```
js.queue = [];
p=g.createElement(r);p.async=!0;p.src=a;h=g.getElementsByTagName(r)[0];
h.parentNode.insertBefore(p,h);
}(window,document,'script','//cdn.qgr.ph/qgraph.<your app id>.js');
</script>
```

You can customize various aspects of how and when various prompts appear in our website. Here are the things that you can change:

*Fake Prompt* You can set up whether or not you want fake prompt to be displayed. The default setting is yes, with a default fake prompt being displayed. You can also set the title, message, button texts and button colors shown in the fake prompt. You can also set the location of the fake prompt (either top left or top center).

*Delay*: Controls how many seconds after the user has come on the website, do we show system prompt/fake prompt as applicable. Default value is 0.

*Seconds between prompts*: You can controls the number of seconds between two fake prompts, in case the user declines the request for notification on the first fake prompt. Default value is 3600.

*Requesting by self*: If set, the QGraph SDK does not show the system prompt or the fake prompt by itself. It is your code which decides when to do that. You can show the prompt any time by this code:

```
qg("prompt-push")
```

Note that if you use this parameter, *delay* and *secondsBetweenPrompts* are not considered. Default value is *false*.

*Overlay*: You can display an overlay while system prompt is displayed. You can set title and message for the overlay.

## 5.2.2 If your site is HTTP

In case your site is HTTP, you need a backing HTTPS site to enable push notifications. You can either have your backing HTTPS domain, or use a QGraph provided HTTPS domain. The notification text displays what domain it is coming from: thus using your own domain provides a better experience to your users. However, that involves extra work too, and thus many websites choose to use QGraph provided HTTPS domain, which leads to simpler integration.

1. Login to <http://app.qgraph.io>
2. Go to “Setup” -> “Integrations” section on the side bar menu. Select “Web” from the three channels presented to you.
3. Enter your website’s URL. Make a choice whether you want to have your own HTTPS domain as backing domain, or want to use a QGraph provided domain. Make a note of it.
4. (Skip this step if you are using QGraph provided HTTPS domain) In case, you want your own HTTPS domain, download the following files

(a) *<https://cdn.qgr.ph/qg-sw.<your app id>.js>*

(b) *<https://cdn.qgr.ph/manifest.json>*

(c) *<https://cdn.qgr.ph/notify.html>*

Copy above files to the root folder of your HTTPS website. To make sure that you have done this correctly, verify if following files are accessible

(a) *<https://yourwebsite.com/qg-sw.<your app id>.js>*

(b) *<https://yourwebsite.com/manifest.json>*

(c) *<https://yourwebsite.com/notify.html>*

Add the following lines inside of head or body tag of your web page:

```
<script type="text/javascript">
!function(q,g,r,a,p,h,js){
  if(q.qg)return;
  js = q.qg = function() {
    js.callmethod ? js.callmethod.call(js, arguments) : js.queue.push(arguments);
  };
  js.queue = [];
  p=g.createElement(r);p.async=!0;p.src=a;h=g.getElementsByTagName(r)[0];
  h.parentNode.insertBefore(p,h);
}(window,document,'script','//cdn.qgr.ph/qgraph.<your app id>.js');
</script>
```

You can customize various aspects of how and when various prompts appear in our website. Here are the things that you can change:

*Fake Prompt* You can set up whether or not you want fake prompt to be displayed. The default setting is yes, with a default fake prompt being displayed. You can also set the title, message, button texts and button colors shown in the fake prompt. You can also set the location of the fake prompt (either top left or top center).

*Delay*: Controls how many seconds after the user has come on the website, do we show system prompt/fake prompt as applicable. Default value is 0.

*Seconds between prompts*: You can controls the number of seconds between two fake prompts, in case the user declines the request for notification on the first fake prompt. Default value is 3600.

*Requesting by self*: If set, the QGraph SDK does not show the system prompt or the fake prompt by itself. It is your code which decides when to do that. You can show the prompt any time by this code:

```
qg("prompt-push")
```

Note that if you use this parameter, *delay* and *secondsBetweenPrompts* are not considered. Default value is *false*.

*Overlay*: You can display an overlay while system prompt is displayed. You can set title and message for the overlay.

*Thank you prompt*: You can control the title, message and location of thank you prompt. Thank you prompt is required for HTTP integrations.

## 5.3 Logging Data

QG web SDK provides you ways to send us data about the users. Once you send us the data you can segment on the basis of that data (E.g. send a web push to users meeting certain criterion) and customize on the basis of that data (E.g. insert the image of the product that the user has seen, or the image of the product that you recommend for the user). You can send us two types of data: the attributes of a user, like email, name, city etc. (what we call profile information) and the data related to the activity that the user is doing.

### 5.3.1 Logging profile information

You log profile information using *identify* functionality of the function *qg*. For instance:

```
qg("identify", {"email": "myemail@somedomain.com"});
```

logs the email of the user. You can set multiple properties at once, like this:

```
qg("identify", {"email": "myemail@somedomain.com", "first_name": "John", "last_name":  
↪ "Doe"});
```

### 5.3.2 Logging event information

You log events using *event* functionality of the function `qg`. Following code logs an event *product\_viewed*:

```
qg("event", "product_viewed");
```

You can have parameters related to the events. For example, following code logs an event *product\_viewed* with parameters `product_id`, `name` and `price`:

```
qg("event", "product_viewed", {"product_id": 123, "name": "Adidas shoes", "price":  
↪ 4000});
```



### 6.1 Passing Dates and Times to QGraph Servers

Sometimes you may wish to pass date or time to QGraph servers. For instances, you may wish to pass the date of journey to QGraph servers. QGraph SDKs (iOS, android and web) take only integers, real numbers, strings and booleans as parameters. Thus, dates and times need to be formatted as strings before they can be passed to QGraph. This section describes the format which is understood by QGraph servers.

#### 6.1.1 Format for Date

Format for date is YYYY-MM-DD. Thus, July 12, 2017 would be passed as string "2017-07-12".

#### 6.1.2 Format for Time

Format for time is HH:MM:SS where HH is in 24 hour format. Thus, 3:10:06 PM is to be formatted as "15:10:06" and 2:04:42 AM is to be formatted as "02:04:42".

#### 6.1.3 Format for Datetime

There are two formats for datetime:

1. Timezone unaware datetime is of the format YYYY-MM-DDTHH:MM:SS. For example, "2017-07-12T15:10:06".
2. Timezone aware datetime format is YYYY-MM-DDTHH:MM:SS[+/-]HH:MM. For example, "2017-07-12T15:10:06+05:30" is a datetime in IST timezone, while "2017-07-12T15:10:06-08:00" is a datetime in PST timezone.

## 6.2 Passing data to QGraph from your servers

The usual way for QGraph to get your users' data is through our SDKs. As your users' interact with your android, iOS or web app, you call some functions of QGraph SDK and transmit the activities to QGraph servers.

In some cases, you would need to pass your users' data to QGraph through your servers. For example, if you are an ecommerce website, you may wish to pass QGraph an event when a customer order has been dispatched, or when an order has been delivered to your customer. This is so that you may run a triggered campaign on, say, order delivery, sending a notification to your user when the order has been delivered.

You can pass us two kinds of information about your users:

1. Profile information:

This is information like name, city, or birthday of the user.

2. Event information:

This is information about an event like an order has been picked up, or an order has been delivered.

To pass data to QGraph servers, you make an HTTP POST request to the following URL:

```
https://api.quantumgraph.com/qga/clients-data/
```

Note that in one call, you can transfer the information about one user only.

POST body format is as following:

```
{
  "appId": <your app id>
  "appSecret": <your app secret>,
  "identifier": <an identifier to identify the user>
  "identifier_value": <value of the identifier>,
  "device": <device type of the user>
  "profiles": <profile information of the user, if any>,
  "events": <even information, if any>
}
```

Let us consider these fields one by one.

*appId* is the unique identifier for your account. It is available from “Account Settings” page of your account.

*appSecret* is the unique secret for your app. It is also available from “Account Settings” page of your account.

*identifier* is the attribute on the basis of which you identify the user. It can be one of the following: (a) *user\_id*, as set by calling the function *setUserId()* from within the SDK (b) *email*, as set by calling the function *setEmail()* from within the SDK (c) *IDFA*, for iOS users, (d) *advertisingId*, for android users. Note that if you are identifying the users through *user\_id* or *email*, you should have passed the *user\_id* or *email* of the user to QGraph by calling *setUserId()* or *setEmail()* from within the SDK.

*identifier\_value* is the value of the *identifier* for the user for whom the request is being made.

*device* is one of *android*, *ios*, *web* or *fb*.

*profiles* is an optional key. Its value is a dictionary consisting of key value paris that you want to set for the particular user. An example profile entry would be:

```
{
  "first_name": "John",
  "last_name": "Doe"
}
```



*events* is an optional key. Its value is a list consisting of dictionaries. Each dictionary contains an *eventName* and an optional dictionary consisting *parameters* which in turn consists of parameter name and values. Here is an example consisting of a single event:

```
[{
  "eventName": "user_registered"
}]
```

And here is an example consisting of two events, one with a parameters and the other without parameters:

```
[{
  "eventName": "user_registered"
},
{
  "eventName": "order_placed",
  "parameters": {
    "order_id": "3j3dkd4k50",
    "order_value": 253
  }
}]
```

For example, here is one complete request that you may make:

```
POST https://api.quantumgraph.com/qga/clients-data/
{
  "appId": "ad55582957817e511c3d",
  "appSecret": "2dbc2fd2358e1ealb7a6bc08ea647b9a337ac92d",
  "identifier": "email",
  "identifier_value": "johndoe@gmail.com",
  "profiles": {
    "first_name": "John",
    "last_name": "Doe"
  },
  "events": [{
    "eventName": "order_placed",
    "parameters": {
      "order_id": "3j3dkd4k50",
      "order_value": 253
    }
  }]
}
```



Our web UI is present at <http://app.qgraph.io/>

Using our web UI, you can

1. Follow on boarding steps to send a test notification to your device.
2. Create segments and campaigns to send notifications to your app users.
3. See analytics related to the notifications that you sent.

**In the UI, you see four sections**

1. Last Modified Users
2. Latest Activities
3. Campaigns
4. Segments

## 7.1 1. Recent Users

This section shows profile of last few users which have been active on the app. Once you do some activity on the app, you should be able to see a user corresponding to you in the app. (You can recognize yourself using some of the fields that you have been logging, like email or username)

Here are the fields that you should know about:

1. *GCM ID*: This is an identifier generated by google which uniquely identifies an app on a device. It needs to be present for QGraph to be able to send a notification to a user.
2. *user Id*: This is your identification of the user. It would usually be an email or user id of the user.
3. *Other fields*: This shows other user profile fields that may have set using functions like `setName()`, `setCity()` etc, or `setCustomUserParameter()`.

## 7.2 2. Recent Activities

Here you can see the information regarding last few events which have happened on the app.

You should perform some activity in the app which logs a new event, and check that you can see the event.

## 7.3 3. Segments

A segment is a set of users. In this tab, you specify the criteria for a segment. Users which satisfy all the criteria specified will constitute the segment.

If you do not specify any criteria, segment will contain all the app users.

## 7.4 4. Campaigns

Once you have created one or more segments, you are ready to create a campaign. You choose a creative type. Choose one of “Base”, “Icon”, “Big Image” or “Rolling”. You fill in all the fields.

As a first step, just choose the “Icon” or “Big Image” type and specify various fields. In this step, fill the same strings in “custom” and “default” fields. Select one of the segments, and click “Add Campaign”. Don’t worry, no notifications will go out yet.

You can click “Run” to send the notifications to the users. Refresh the screen after some time and you should see some stats about how many notifications were sent. If you are a part of the segment, you should receive a notification too.

As a second step, you can customize the message in two ways:

1. Suppose you have passed us variables in the user profiles called “name” and “city”. Then, if while creating a campaign, in the “custom” title or message you write:

Hello {{name}}, you live in {{city}}.

Then, for each user, {{name}} and {{city}} will be substituted by the actual variable values. For those users in the segment for whom one of these variables is not present, the “default” string provided by you will be used.

2. Suppose you provided us an event called *product\_viewed* and in the parameters you specified:

```
{
  'name': 'abccamera',
  'pid': 123,
  'price': 15999,
  'image_url': 'http://mysite.com/abc-camera.jpg'
  'redirect_url': 'myapp://myapp.com/123'
}
```

Then you can refer to “name”, “price”, “image\_url”, “redirect\_url” as {{product\_viewed.0.name}}, {{product\_viewed.0.price}}, {{product\_viewed.0.image\_url}}, {{product\_viewed.0.redirect\_url}} respectively.

You can use the first two variables to customize the message, like this:

Hello {{name}}, the prices of {{product\_viewed.0.name}} have fallen by 40%!

And you can specify the last two variables to specify the image and redirect url for the notification.

We provide programmatic access to several features of our platform.

1. Log in on <https://app.qgraph.io> and click your name on the top right of the screen. In the drop down menu that comes, select “Account Settings”. Note down the “API Token” for your account.
2. You need to set Authorization key as "Token: <your API token>" in the header of your http requests. For instance, if using curl, you do it like this:

```
curl -H "Authorization: Token abcd" <relevant api url>
```

## 8.1 Sending notifications

First you need to create a campaign. Go to <https://app.qgraph.io>, log in, go to “Campaigns” tab and create a new campaign. You can fill any value in the campaign, they will be overridden by what you provide in the api call.

Once you have created the campaign, proceed to edit that campaign. URL of the performance page looks like: [https://app.qgraph.io/#/edit\\_campaign/<campaign id>](https://app.qgraph.io/#/edit_campaign/<campaign id>). Note down the campaign id for your campaign.

Next you can make a HTTP POST request at <https://api.qgraph.io/api/v2/send-notification/>. The POST body is of the following format:

```
{
  "cid": <campaign id>,

  "registration_ids": ["regid 1", "regid 2", ..., "regid n" (upto 500 registration_
↪ids)]
  or
  "user_ids": ["userid 1", "userid 2", ..., "userid n" (upto 500 userid)]
  or
  "emails": ["email 1", "email 2", ..., "email n" (upto 500 emails)]
  or
  "segment_id": <segment id>
```

(continues on next page)

(continued from previous page)

```

"message": <message in the format described below>
"os": "android" or "ios-dev" or "ios-prod" or "web"
}

```

You need to provide one of `registration_ids`, `user_ids`, `emails` or `segment_id`. In case you provide segment id, specified segment id must be a valid segment in your account, and notifications will go to that segment. To find segment id of a given segment, proceed to edit that segment. URL of the segment page is of the format `https://app.qgraph.io/#/edit_segment/<segment id>`.

If you want to send notification to android devices, use `android` for key `os`. If you want to send notification to ios devices, use `ios-dev` or `ios-prod`, depending on whether you want us to use development profile or production profile. (You should have uploaded the respective `.pem` file to us)

For a simple android notification, message is of the following format:

```

{
  "type": "basic",
  "title": <title of the notification>,
  "message": <body of the notification>,
  "imageUrl": <url of the icon image> (optional),
  "bigImageUrl": <url of the big image> (optional),
  "deepLink": <deep link of notification> (optional)
  "actions": [{ "id": 1, "text": "<button 1 text>", "deepLink": "<deep link if any>"
↵,
                { "id": 2, "text": "<button 2 text>", "deepLink": "<deep link if any>"
↵,
                { "id": 3, "text": "<button 3 text>", "deepLink": "<deep link if any>"
↵} <optional>
}

```

A note about action buttons: If you would like a campaign with action buttons to be a poll campaign (where, on button press, response of the user is recorded, but the app does not open), set the key `poll` to `true` in the `message`. You can send 1, 2 or 3 actions, and deep link within each button is optional.

For ios notification, message is of the following format:

```

{
  "aps": {
    "alert": {
      "title": <title of the notification>,
      "body": <body of the notification>
    }
  },
  "deepLink": <deep link> (optional)
}

```

For banner notification (available only in android), message is of the following format:

```

{
  "type": "banner",
  "title": <title of the notification>,
  "message": <body of the notification>,
  "contentImageUrl": <url of banner image>,
  "deepLink": <deep link of notification> (optional)
}

```

For carousel notification (available only in android), message is of the following format:

```
{
  "type": "carousel",
  "title": <title of the notification>,
  "message": <body of the notification>,
  "deepLink": <deep link of notification> (optional),
  "qg_prev_button": "https://cdn.qgraph.io/img/left.png",
  "qg_next_button": "https://cdn.qgraph.io/img/right.png",
  "carousel": [{ "image": "<URL of the image>",
                  "deepLink": "<deep link for image, if any>",
                  "title": "<title of image (optional)>",
                  "message": "<message of image (optional)>" },
                ... you can have up to 10 such elements ]
}
```

Note that you can customize previous and next buttons by using your own image URL.

For slider notification (available only in android), message is of the following format:

```
{
  "type": "slider",
  "title": <title of the notification>,
  "message": <body of the notification>,
  "deepLink": <deep link of notification>, (optional)
  "qg_prev_button": "https://cdn.qgraph.io/img/left.png",
  "qg_next_button": "https://cdn.qgraph.io/img/right.png",
  "slider": [{ "image": "<URL of the image>",
                "deepLink": "<deep link for image, if any>" },
              ... you can have up to 10 such elements ]
}
```

Note that you can customize previous and next buttons by using your own image URL.

For animated banner notification (available only in android), message is of the following format:

```
{
  "title": <title of the notification>,
  "message": <body of the notification>,
  "deepLink": <deep link of notification>, (optional)
  "gifPlayButton": "https://cdn.qgraph.io/img/video_button.png",
  "type": "animation"
  "animation": {
    "millisecondsToRefresh": <duration between two frames in milliseconds>,
    "images": [url1, url2, ..., url n]
  }
}
```

If os is “web”, message is of the following format:

```
{
  "title": <title of the notification>,
  "body": <body of the notification>,
  "icon": <url of the icon image>
}
```

### 8.1.1 Specifying key value pairs

You can specify key value pairs in (both android and ios) notifications. To do this, include a key `qgPayload` in your message dictionary. `qgPayload` should contain key-value pairs. For example, a sample message for android would be:

```
{
  "type": "basic",
  "title": <title of the notification>,
  "message": <body of the notification>,
  "imageUrl": <url of the icon image> (optional),
  "bigImageUrl": <url of the big image> (optional),
  "deepLink": <deep link of notification> (optional)
  "qgPayload": {
    "key1": "some value",
    "key2": 123
  }
}
```

Key value pairs can then be extracted in your activity as described here: <http://docs.qgraph.io/en/latest/integrating-android-sdk.html#receiving-key-value-pairs-in-activity>

## 8.2 Getting user profiles

Send a GET request to <https://app.qgraph.io/api/get-user-profiles/>. For instance, if your token is `abcd`, the relevant call in curl would be:

```
curl -H "Authorization: Token abcd" https://app.qgraph.io/api/get-user-profiles/
```

### 8.2.1 Specifying start and end dates

You can optionally provide parameters `start_date` and `end_date` to the API call. If these parameters are provided, the API fetches entries only for the users who have installed the app on or after `start_date`, but on or before `end_date`. The format of the both the arguments is `yyyy-mm-dd`. A sample call would be:

```
curl -H "Authorization: Token <your token>" https://app.qgraph.io/api/get-user-
↪profiles/?start_date=2015-12-22&end_date=2015-12-25
```

For faster response times, you should retrieve the data for small date ranges.

### 8.2.2 Specifying OS

You can specify the ios for which you want to retrieve data. You specify this by providing a query parameter `os` whose values can be `android` (for android), `ios-prod` (for ios using production profile), or `ios-dev` (for ios using development profile). Default value for `os` is `android`. Here is an example of using this variable:

```
curl -H "Authorization: Token <your token>" https://app.qgraph.io/api/get-user-
↪profiles/?start_date=2015-12-22&end_date=2015-12-25&os=android
```



### 8.2.3 Specifying specific fields to retrieve

You can get following fields using the api:

1. *firstSeen*: Date when the user installed your app
2. *mTime*: Latest date when the user accessed your app
3. *monthlyActivity*: Number of days in last 30 days when the user accessed your app
4. *email*: email of the user, if available
5. *qgCity*: city of the user, if available
6. *uninstallTime*: date when we detected that the user has uninstalled your app
7. *user\_id*: the user id set by `setUserId()` function of the SDK
8. *qgType*: tells whether the install is a fresh one or a reinstall
9. *qgSrc*: source of the install, if available
10. *gcmId*: gcm registration id of the user in case of android and device token in case of ios
11. *deviceId*: device id of the user
12. *advId*: advertiser id of the user

You can specify what specific fields you want. For instance, if you want to get *firstSeen*, *uninstallTime* and *gcmId* of all the users who installed your app between December 1, 2015 and December 3, 2015, the relevant curl call would be:

```
curl -H "Authorization: Token <your token>" https://app.qgraph.io/api/get-user-
  ↳ profiles/?start_date=2015-12-01&end_date=2015-12-03&fields=firstSeen,uninstallTime,
  ↳ gcmId
```

For faster response times, you should retrieve only the fields that you need.

## 8.3 Create a user uploaded segment

You usually create a user uploaded segment by manually uploading a file in the Segment -> Add New -> Uploaded Segment. However, you can also do it using our API. Uploading a segment is a two step process:

First you need to upload the segment file. This file needs contain one field value (such as email) per line. You upload it by a command similar to this:

```
curl -H 'Authorization: Token <your token>' \
  -H 'content-type: multipart/form-data' \
  -F file=@/path/to/your/file\
  https://app.qgraph.io/qganalyzedata/upload-segment-file/
```

This gives an output like:

```
{"filename": "upload.csv1495733409.41"}
```

Here `upload.csv1495733409.41` is the temporary name of the file that has been created on the server. You will need this name in the second step.

Secondly, you use above outputted temporary filename to create a segment, like this:

```
curl -X POST\  
  -H 'Authorization: Token <your token>'\  
  -H 'appId: <your app id>' \  
  -H 'content-type: application/json'\  
  -d '{"name": "<name of the segment>", "description": "<description of the_  
↪segment>", "filename": "<filename produced in step 1>", "field": "<field name whose_  
↪values are present in the file>"}'\  
  https://app.qgraph.io/qganalyzedata/upload_segment/
```

For instance, if the uploaded file contained email, a sample command to upload the file would be:

```
curl -X POST\  
  -H 'Authorization: Token <your token>'\  
  -H 'appId: <your app id>'\  
  -H 'content-type: application/json'\  
  -d '{"name": "my uploaded segment", "description": "This is a bunch of emails",  
↪"filename": "upload.csv1495733409.41", "field": "email"}'\  
  https://app.qgraph.io/qganalyzedata/upload_segment/
```

Segment is created as a result of this request.

## 9.1 Android

AAR file for GCM integration is available at <http://jcenter.bintray.com/com/quantumgraph/sdk/QG/2.3.4.1/QG-2.3.4.1.aar>

AAR file for FCM integration is available at <http://jcenter.bintray.com/com/quantumgraph/sdk/QG/4.3.1/QG-4.3.1.aar>

But we recommend that you follow maven based integration outlined in android integration instructions.

## 9.2 iOS

Download QGraph SDK for iOS from here:

For Objective C: <http://app.qgraph.io/static/sdk/ios/QGSdk-ObjC-3.3.4.zip>

For Swift: <http://app.qgraph.io/static/sdk/ios/QGSdk-Swift-3.3.4.zip>

We recommend that you follow cocoapods based integration outlined in iOS integration instructions.