# piStats Documentation

***Release 0.0.1***

**Gaurav Batra**

**Apr 06, 2018**

# Contents:


**1 Key Concepts** **1**
- 1.1 Property-Id . . . 1
- 1.2 CMS Integration . . . 1
- 1.3 Referral Analytics . . . 2
- 1.4 Push Notifications . . . 3
- 1.5 Author Performance Index (API) . . . 3

**2 Analytics Integration** **5**
- 2.1 CMS Integration . . . 5
- 2.2 Javascript . . . 6
- 2.3 Android . . . 7
- 2.4 iOS - ObjectiveC . . . 8
- 2.5 iOS - Swift . . . 9

**3 Push Notification Integration** **13**
- 3.1 Javascript . . . 13
- 3.2 Android . . . 14
- 3.3 iOS - Objective-C . . . 17
- 3.4 iOS - Swift . . . 19

CHAPTER 1

# Key Concepts

## 1.1 Property-Id

Depending on how your app or websites are structured you might want to consider creating a single or multiple property-id's. Rule of thumb is, assets that you want to track in one analytics dashboard should have one property id.

### 1.1.1 Example 1: One language - multiple assets

One property id: In this scenario the analytics for all the assets would be combined under one dashboard. Multiple property id: If each asset has a different property id, each asset would have it's own dashboard.

### 1.1.2 Example 2: Multiple languages - multiple assets

One property id: Same as example above, all the languages and assets would share the same dashboard. But language is one of the special attributes which can be used on each dashboard to filter the data. Multiple property id: We can either create one property per asset (an asset can share multiple languages) or we can create one property id per language.

## 1.2 CMS Integration

CMS integration is one of the unique features of piStats. By integrating with the CMS directly piStats is able to provide key insights which are enriched with CMS data. For example, a single article can be viewed through different URLs, but piStats can consolidate the analytics for that article because it uses a unique identifier for each article.

Similarly great insights can be derived into the author performance and effectiveness using the meta information from CMS.

## 1.3 Referral Analytics

An event happens when a user interacts with the website or mobile App. Each event has a referrer, for example a user might click on a post on FB and land on your site, in which case the referrer would be "Facebook.com". But that is not the only information captured by piStats, in this example

- ReferralType: would be 'Social', since user came from a social referrer
- ReferralOrigin: would be "facebook"
- ReferralMedium: would be the URL of facebook, e.g. "http://m.facebook.com/".

Hierarchy of referrers can be used to track other things as well. For example how many people read an article from a specific widget, in which case the referrers can be used as:

- ReferralType: 'Internal'
- ReferralOrigin: <Page Name>
- ReferralMedium: <Name of Widget, section or menu>
- ReferralIndex: <relative index of the item clicked by user>

This structure is pretty versatile and can be used to track a variety of user interactions.

### 1.3.1 Example 1: Tracking a menu item

Let's say we want to track how efffectiveness of our menu items. We want to figure out which of menu's are used most by the users and does their relative positioning affects their usage.

We can track how many clicks each menu item on "Home Page" gets as follows:

When the user clicks the first menu item on home page, we can send the following while calling the load event:

- ReferrerType: 'Internal'
- ReferrerOrigin: 'HomePage'
- ReferrerMedium: 'TopMenu'
- ReferrerIndex: '1'

Similarly for the 5th item on Menu we can send the following:

- ReferrerType: 'Internal'
- ReferrerOrigin: 'HomePage'
- ReferrerMedium: 'TopMenu'
- ReferrerIndex: '5'

### 1.3.2 Example 2: Tracking a widget

Let's say we want to track how a specific widget, which has multiple stories displayed in a caraousel, on the 'IndiaNews' page. To track a widget like this we would send the following information:

- ReferrerType: 'Internal'
- ReferrerOrigin: 'IndiaNews'
- ReferrerMedium: 'PopulatStoriesCaraousel'
- ReferrerIndex: '3'

## 1.4 Push Notifications

### 1.4.1 Topic

Represents a user segment that can be targeted to send push notifications. For example: A topic can be as broad as, 'HindiAndroid', i.e. all Hindi language users on Android or it can be as specific as "All Delhi users who read the President election news on Android"

### 1.4.2 User preferences

In some apps user preferences drive the topic subscription and unsubscription. A couple of common examples:

- User can choose categories for which he wants to get push notifications. For example a user might only be interested in getting notifications for Politics, Bollywood and Sports. Each of the categories in this case would correspond to one topic. When user sets his preferences, app can trigger a subscription to these topics for the user.
- User can choose locations (City) for which he might want to get the push notifications. Same as in the above case, each city would represent 1 topic and app can trigger subscription for the user on one or more topics.

### 1.4.3 Reach

Push notificaion reach, is the total number of push notifications sent multiplied by total subscribers. For example, if the app has 100,000 active subscribers and 5 notificaions were sent out to all users, then the reach would be 500,000 (5 x 100,000).

## 1.5 Author Performance Index (API)

Author performance index indicates how well an authors is doing, in comparison to other authors. To calculate the API for each author is a two step process, first we take the total pageviews for the author and divide it by the number of stories published by the author. Once we have the ratios for each author we normalize the ratios by dividing each ratio by total sum of ratios.

Let's look at an example, let's say in a single day we have the following author data:

| Author | Stories Published | Pageviews |
|---|---|---|
| Author1 | 5 | 50,000 |
| Author2 | 10 | 35,000 |
| Author3 | 7 | 27,000 |
| Author4 | 17 | 17,000 |
| Author5 | 1 | 5,000 |

Step:1 Ratio of pageviews to published stories for each author

- Author1 = 50,000/5 = 10,000
- Author2 = 35,000/10 = 3,500
- Author3 = 27,000/7 = 3857.14
- Author4 = 17,000/17 = 1,000
- Author5 = 5,000/1 = 5,000

Total (10,000 + 3,500 + 3857.14 + 1,000 + 5,000) = 23357.14

Step:2 Normalize the performance index for each author

- Author1 API (10,000 / 23,357) = 0.42
- Author2 API (3,500 / 23,357) = 0.14
- Author3 API (3857.14 / 23,357) = 0.16
- Author4 API (1,000 / 23,357) = 0.04
- Author5 API (5,000 / 23,357) = .21

In this example Author1 has the best performance, followed by Author3 and so on.

CHAPTER 2

# Analytics Integration

## 2.1 CMS Integration

When a new article is published you can use the following API, to push it to piStats:

API Endpoint: http://events.pi-stats.com/content

```
{
  "articleId" : ,
  "language" : ,
  "format" : ,
  "thumbnail" : ,
  "propertyId" : ,
  "title" : ,
  "url" : ,
  "publicationDate" : ,
  "authorName" : ,
  "authorEmail" : ,
  "authorDesignation" : ,
  "authorImage" : ,
  "categoryList" : [],
  "sectionList" : [],
  "editorList" : [],
  "tagList" : [],
  "body" : '',
  "description" :   ''
}
```

| Field | Description | Datatype | Example |
|---|---|---|---|
| articleId | Unique id of the article | string | 'c059D45' |
| language | Article Language | string | 'english', 'en' , 'hindi' |
| format | Article format/type | string | 'video', 'gallery' |
| thumbnail | Article thumbnail URL article | string | 'http://imageurl' |
| propertyId | Property id | string | 'client-live' |
| title | Article headline | string | 'Election Results Live' |
| url | Article URL | string | 'http://article.com' |
| publicationDate | Article publication date (UTC format) | date | '2017-05-15T12:09:09Z' |
| loginAuthorName | Author name | string | 'Author Name' |
| authorEmail | Author email | string | 'author@greatnews.com' |
| authorDesignation | Author designation | string | 'editor' |
| authorImage | Author image url | string | 'http://authorimageur' |
| categoryList | Article category list | list of string | ['India', 'Business'] |
| sectionList | Article section list | list of string | ['video', 'gallery'] |
| editorList | Article author list (multiple contributors) | list of string | ['author1', 'author2'] |
| body | Article body | string | 'article body' |
| description | A short description of Article | string | 'Election Results Live' |

## 2.2 Javascript

Tracking events on web is a two step process:

1. Include Javascript library: To start tracking the analytics events, please include the JavaScript library on all the pages which should be tracked. The library would be loaded asynchronously with the given code snippet.
    - JavaScript Library: pistats-lib-analytics.pi-stats.com/library.min.js
        - init( <property-id> , true ) : This method needs to be called on every page.
            * <property-id>: pass property-id as the first argument
            * <true/false>: pass 'true' to enable automatic tracking for the page. 'false' to track manually.
        - load(): If 'false' is passed in the 'init' method, then load() method needs to be called manually on each page after init method has been called.
2. Expose variables on page: Expose the following variables on each page. piStats would automatically pick up these variables.
    - 'site_lang' : Language of the site, for example 'hindi', 'tamil' etc.
    - 'site_story_id': Unique article id which is being read. On pages where article id might not be available, a unique id can be returned. For example, on home page article id can be 'homepage'.

### 2.2.1 Note:

Before calling the init and load methods, please make sure the piStats library has loaded. The best place to include the library is the 'head' section and the methods can be called anywhere after that, for example in the footer.

```
<script>
  var site_lang=<language>;
  var site_story_id=<story_id>;
  $.getScript( "http://dlvkf5067xruv.cloudfront.net/library_analytics.min.js",
↪function( data, status, jqxhr ) {
```

```
  });
</script>
```

```
piStats.init(<property-id>);
piStats.load("Page Load");
```

Another way to do this is to combine the calls, and call the init and load methods on the library callback, as follows:

```
<script>
   var site_lang=<language>;
   var site_story_id=<story_id>;
   $.getScript( "http://dlvkf5067xruv.cloudfront.net/library_analytics.min.js",
↪function( data, status, jqxhr ) {
      var site_story_id = site_story_id;
      piStats.init(<property-id>);
      piStats.load('PageLoad');
    });
</script>
```

## 2.3 Android

1. Add the following permissions:

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

2. Add piStats SDK to your app:
   - In Android studio 'File > New > New Module' → Select 'Import .jar/.aar' package
   - import piStats.aar
   - In build.gradle add:

```
compile project(':pistats')
```

3. Add the following in 'onCreate()' of the application

```
PiStats.getInstance().init(this, <PropertyId>);
```

4. Registration :

```
public class PistatsRegistrationReceiver extends BroadcastReceiver {

@Override
 public void onReceive(Context context, Intent intent)
 {
   Event event = new Event();
     event.setPropertyId(<propertyId>);
     if (!TextUtils.isEmpty(<Language>))
         event.setLanguage(<Language>);
     PiStats.getInstance().registrationEvent(event);
 }
}
```

- Register the broadcast receiver in manifest

```
<receiver
  android:name=".PistatsRegistrationReceiver"
  android:enabled="true"
  android:exported="true"
  tools:ignore="ExportedReceiver">

   <intent-filter>
      <action android:name="com.pistats.registration.action"/>
   </intent-filter>
</receiver>
```

5. Load: This event is used to track all the different pages/screens a user visits. This would be the most frequently used event in the app. Not only does this event keep track of user browsing history but also combines referral information as well.

Note: For details about referral parameters please refer to: Referral Analytics

```
  LoadEvent event = new LoadEvent();

  event.setEventName("PageLoad");
  event.setPropertyId(<PropertyId>);
  event.setLangauageUser(<Language>);
  event.setContentId(<contentId>);
  event.setReferrerType(<ReferrerType>);
  event.setReferrerOrigin(<ReferrerOrigin>);
  event.setReferrerMedium(<ReferrerMedium>);
  event.setReferrerIndex(<ReferrerIndex>);

PiStats.getInstance().pageLoadEvent(event);
```

## 2.4 iOS - ObjectiveC

1. Add the following permissions in the app's plist file:

- Location

2. Add piStats SDK to the X-code project
3. Add Firebase SDK: Please follow the instructons in the link below: https://firebase.google.com/docs/cloud-messaging/ios/client
4. Import piStats in AppDelegate.m

```
import <pistats/PistatsManage.h>
```

5. Initialize piStats in application:didFinishLaunchingWithOptions

```
(BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions

{

 [[PistatsManage getInstance]initWithProperty:PiStatsPropertyid];

}
```

6. Implement observer: Add kPiStatsObserverNotification in application:didFinishLaunchingWithOptions

```
(BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
  {
        [PistatsManage getInstance];
        [[NSNotificationCenter defaultCenter]addObserver:self␣
↪selector:@selector(pistatObserverCall:)
       name:kPiStatsObserverNotification object:nil];
  }
(void)pistatObserverCall:(NSNotification*)notification
  {
        [self registrationCallForPistats];
  }
```

After piStats SDK has been initialized we need to implement:

1. Registration :

- After we have received the FCM registration token we need to register the device with piStats for push notification.

```
NewEvent *event           = [[NewEvent alloc]init];

     event.Language               = @"Language name",
     event.PropertyId             = PiStatsPropertyid;
     event.deviceUDID             = @"device UDID";

[[PistatsManage getInstance]registerForDevice:PiStatsPropertyid WithEvent:event];
```

2. Load: This event is used to track all the different pages/screens a user visits. This would be the most frequently used event in the app. Not only does this event keep track of user browsing history but also combines referral information as well.

Note: For details about referral parameters please refer to: Referral Analytics

```
NewEvent *event       = [[NewEvent alloc]init];

    event.EventName       = @"PageLoad";
    event.Language        = @"Language name";
    event.ReferrerOrigin  = @"internal";
    event.ReferrerMedium  = @"page";
    event.ReferrerType    = @"type";
    event.EventTimestamp  = @"Timestamp" /* ISO date and time formate
    event.PropertyId      = PiStatsPropertyid;
    event.ContentID       = @"ContentID";
    event.deviceUDID      =  @"device UDID";

[[PistatsManage getInstance]loadviewController:self WithEvent:event];
```

## 2.5 iOS - Swift

1. Add the following permissions in the app's plist file:

- Location
- Notifications

2. Add piStats SDK to the X-code project and import two files in project bridging header

```
#import <pistats/PistatsManage.h>
#import <pistats/NewEvent.h>
```

3. Add Firebase SDK: Please follow the instructons in the link below: https://firebase.google.com/docs/cloud-messaging/ios/client

4. Import piStats in AppDelegate

```
import pistats
```

5. Initialize piStats in application:didFinishLaunchingWithOptions

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions␣
↪launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool
   {
     PistasManage.getInstance()(property:PiStatsPropertyid)
   }

}
```

6. Implement observer: Add kPiStatsObserverNotification in application:didFinishLaunchingWithOptions

```
 NotificationCenter.default.addObserver(self, selector: Selector(("pistatObserverCall:
↪")), name: NSNotification.Name.piStatsObserver, object: nil)

func pistatObserverCall(_ notification: Notification) {

    self.registrationCallForPistats()
    self.subscribeCallForPistats()
}
```

After piStats SDK has been initialized we can track the following type of events:

1. Registration :

- After we have received the FCM registration token we need to register the device with piStats for push notification.

```
if let fcmTokenDefaultValue = userDefault.value(forKey:pistatsFCMToken)
    {
        fcmTokenDefaultValue = deviceTokenValue as! String
    }
        let notifnEvent = NewEvent()
        notifnEvent.language = currentLanguage()
        notifnEvent.fcmToken = fcmToken  /* Firebase token Mandatory  */
        if let deviceTokenValue = userDefault.value(forKey:pistatsDeviceToken)
        {
            notifnEvent.deviceUDID = deviceTokenValue as! String
        }
        notifnEvent.proprtyId = pistatsPropertyId
        notifnEvent.eventTimestamp  = getTimeandDateInISO()

  (PistatsManage.getInstance() as AnyObject).register(forDevice: notifnEvent)
```

2. Load: This event is used to track all the different pages/screens a user visits. This would be the most frequently used event in the app. Not only does this event keep track of user browsing history but also combines referral information as well.

```
let event = NewEvent()
   event.eventTimestamp = currentFormattedDate()
   event.language = currentLanguage()
   event.proprtyId = pistatsPropertyId
   event.referrerOrigin = RefferOrigin.HOME
   event.referrerMedium = sectionName
   event.contentID = newsID
 (PistatsManage.getInstance() as AnyObject).loadviewController(self, with:event)
```

CHAPTER 3

# Push Notification Integration

Push notification functionality depends upon piStats analytics, so please make sure the analytic integration is also completed.

## 3.1 Javascript

### 3.1.1 Pre-requisite

Before integrating push notifications, piStats library should be integrated with the site. Here are detailed steps PiStats Integration

### 3.1.2 https:// implementation

1. Integrate FCM library: https://firebase.google.com/docs/cloud-messaging/js/client
2. Create a custom popup on the website which would have two Actions "Allow" and "Block".
3. Create a domain level cookie and store user action "Allow" or "Block". Doing so, let's us ask the user again after a certain period of time. For example if the user had "Blocked" the notifications we can store it in cookie for 30 days, and after 30 days when the user comes back we can ask him again.
4. If the user chooses to "Block" the push notifications, store it in cookie.
5. If the user chooses to "Allow" the push notifications, store it in cookie and then open the FCM allow popup
   (a) On the callback of FCM Allow, call FCM methods to Generate FCM token. Here is the link FCM Token generation
   (b) On the callback of FCM token, we need to call piStats subscriptions passing in the FCM token and names of the topics to subscribe and unsubscribe:

```
piStats.subscribeNotification(<fcmToken>,<subscriptionTopicArray>,
↪<unsubscribeTopicArray>)
```

## 3.2 Android

1. Add the following permissions:

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

2. **Add Firebase SDK: Please follow the instructons in the link below:** https://firebase.google.com/docs/cloud-messaging/android/client

3. Add piStats SDK to your app:
   - In Android studio 'File > New > New Module' → Select 'Import .jar/.aar' package
   - import piStats.aar
   - In build.gradle add:

```
compile project(':pistats')
```

4. Add the following in 'onCreate()' of the application

```
PiStats.getInstance().init(this, <PropertyId>);
```

After piStats SDK has been initialized we can track the following type of events:

1. Registration :

- After we have received the FCM registration token we need to register the device with piStats for push notification.

```
public class PistatsRegistrationReceiver extends BroadcastReceiver {

@Override
 public void onReceive(Context context, Intent intent) {
     if (!TextUtils.isEmpty(<language>))
         subscriptionEventForPiStat(<language>);
     Event event = new Event();
     event.setPropertyId(<PropertyId>);
     event.setLanguage(<language>);
     event.setRegistrationId(<FCM token>);
     event.setCustId(<customerId>);
     PiStats.getInstance().registrationEvent(event);
 }

 public static void subscriptionEventForPiStat(String language) {
     if (<user opted out for pushNotificaiton>)
         return;
     try {
         Event event = new Event();
         event.setPropertyId(<PropertyId>);
         event.setLanguage(language);
         event.setRegistrationId(<FCMToken>);
         ArrayList<String> arrayListForSubscription = new ArrayList<>();
         arrayListForSubscription.add(<TopicToSubcribe>);
         arrayListForSubscription.add(<TopicToSubcribe>);
         ......
         ......
```

```
        event.setTopicArrayListForSubscribe(arrayListForSubscription);

        PiStats.getInstance().subscriptionEvent(event);
    } catch (Exception e) {

    }
}
```

}

- Register the broadcast receiver in manifest

```
<receiver
  android:name=".PistatsRegistrationReceiver"
  android:enabled="true"
  android:exported="true"
  tools:ignore="ExportedReceiver">

   <intent-filter>
      <action android:name="com.pistats.registration.action"/>
   </intent-filter>
</receiver>
```

- Call registration event

```
Event event = new Event();

event.setPropertyId(<PropertyId>);
event.setLangauageUser(<Language>);
event.setRegistrationId(<FCMToken>);
event.setCustId(<customerId>);

PiStats.getInstance().registrationEvent(event);
```

2. Subscription: Subscription event is used to subscribe and unsubscribe users to push notification topics

```
Event event = new Event();

event.setPropertyId(<PropertyId>);
       event.setLangauageUser(<Language>);
       event.setRegistrationId(<FCMToken>);
       ArrayList<String> arrayListForSubscription=new ArrayList<>();

 /* Subscribe to topics */

     arrayListForSubscription.add(<Topic1>);
     arrayListForSubscription.add(<Topic2>);
     arrayListForSubscription.add(<Topic3>);
     event.setTopicArrayListForSubscribe(arrayListForSubscription);
     ArrayList<String> arrayListForUnSubscription = new ArrayList<>();

 /* Unsubscribe from topics */

 arrayListForUnSubscription.add(<Topic1>);
     arrayListForUnSubscription.add(<Topic2>);
     arrayListForUnSubscription.add(<Topic3>);
     arrayListForUnSubscription.add(<Topic4>);
```

```
 event.setTopicArrayListForUnsubscribe(arrayListForUnSubscription);

 PiStats.getInstance().subscriptionEvent(event);
```

3. Load: This event is used to track all the different pages/screens a user visits. This would be the most frequently used event in the app. Not only does this event keep track of user browsing history but also combines referral information as well.

```
LoadEvent event = new LoadEvent();

      event.setEventName("PageLoad");
      event.setPropertyId(<PropertyId>);
      event.setLangauageUser(<Language>);
      event.setContentId("contentId");
      event.setReferrerType(<ReferrerType>);
      event.setReferrerOrigin(<ReferrerOrigin>);
      event.setReferrerMedium(<ReferrerMedium>);

    PiStats.getInstance().pageLoadEvent(event);
```

4. Opt-out: In some apps user preference includes switching off the push notification completely. If your app supports such an option, use this event to stop push notifications for the particular user.

```
Event event = new Event();

 event.setPropertyId(<PropertyId>);
     event.setLangauageUser(<Language>);
     event.setRegistrationId(<FCMToken>);
     ArrayList<String> arrayListForTopicUnSubscribe=new ArrayList<>();

 /*Add topics to unsubscribe*/

     arrayListForTopicUnSubscribe.add(<Topic1>);
     arrayListForTopicUnSubscribe.add(<Topic2>);
     arrayListForTopicUnSubscribe.add(<Topic3>);
     event.setTopicArrayListForUnsubscribe(arrayListForTopicUnSubscribe);

PiStats.getInstance().optOutEvent(event);
```

5. Opt-in: Opposite of the "opt-out" event if the user selects the option to receive notifications in the app, call this event to switch the push notifications back on for the user.

```
Event event = new Event();

     event.setPropertyId(<PropertyId>);
     event.setLangauageUser(<Language>);
     event.setRegistrationId(<FCMToken>);
     ArrayList<String> arrayListForTopicSubscribe=new ArrayList<>();

 /*Add topics to subscribe*/

     arrayListForTopicSubscribe.add(<Topic1>);
     arrayListForTopicSubscribe.add(<Topic2>);
     arrayListForTopicSubscribe.add(<Topic3>);
     event.setTopicArrayListForSubscribe(arrayListForTopicSubscribe);

PiStats.getInstance().optInEvent(event);
```

## 3.3 iOS - Objective-C

1. Add the following permissions in the app's plist file:

- Location
- Notifications

2. Add piStats SDK to the X-code project
3. Add Firebase SDK: Please follow the instructons in the link below: https://firebase.google.com/docs/cloud-messaging/ios/client
4. Import piStats in AppDelegate.m

```
import <pistats/PistatsManage.h>
```

5. Initialize piStats in application:didFinishLaunchingWithOptions

```
(BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions

{

 [[PistatsManage getInstance]initWithProperty:PiStatsPropertyid];

}
```

6. Implement observer: Add kPiStatsObserverNotification in application:didFinishLaunchingWithOptions

```
(BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
  {
       [PistatsManage getInstance];
       [[NSNotificationCenter defaultCenter]addObserver:self␣
↪selector:@selector(pistatObserverCall:)
      name:kPiStatsObserverNotification object:nil];
  }
(void)pistatObserverCall:(NSNotification*)notification
  {
       [self registrationCallForPistats];
       [self subscriptionCallForPistats];
  }
```

After piStats SDK has been initialized we can track the following type of events:

1. Registration :

- After we have received the FCM registration token we need to register the device with piStats for push notification.

```
NewEvent *event           = [[NewEvent alloc]init];

     event.Language               = @"Language name",
     event.SubscribeToTopics      = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
     event.FCMToken               = @"Firebase token"
     event.UnsubscribeFromTopics  = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
     event.PropertyId             = PiStatsPropertyid;
```

```
    event.deviceUDID              = @"device UDID";

[[PistatsManage getInstance]registerForDevice:PiStatsPropertyid WithEvent:event];
```

2. Subscription: Subscription event is used to subscribe and unsubscribe users to push notification topics

- For push notifications the subscription call is best placed as a callback from FCM registration

```
(void)tokenRefreshNotification:(NSNotification *)notification
  {
      if(![CurrentFCMPushToken isEqualToString:[userDefaults␣
↪objectForKey:FCMPushToken])
        {
           [[PistatsManage getInstance]registerForDevice:PiStatsPropertyid␣
↪WithEvent:event]; // Registration call for piStats
        }
  }
```

- piStats subscription

```
 NewEvent *event             = [[NewEvent alloc]init];

   event.Language                = @"Language name",
   event.SubscribeToTopics         = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
   event.FCMToken              = @"Firebase token"
   event.UnsubscribeFromTopics     = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
   event.PropertyId              = PiStatsPropertyid;
   event.deviceUDID              = @"device UDID";

[[PistatsManage getInstance]subscribeForNotification:PiStatsPropertyid␣
↪WithEvent:event];
```

- piStats unsubscribe

```
NewEvent *event       = [[NewEvent alloc]init];
    event.Language        = [userDefaults valueForKey:ChannelType],
    event.SubscribeToTopics     = [[NSArray alloc]initWithObjects:currentChannel,␣
↪nil];
    event.UnsubscribeFromTopics = [[NSArray alloc]initWithObjects:lastChannel, nil];
    event.FCMToken        = [userDefaults valueForKey:FCMPushToken],
    event.ProprtyId       = PiStatsPropertyid;
    event.deviceUDID      = [userDefaults valueForKey:UserUDID];

[[PistatsManage getInstance]subscribeForNotification:event];
```

3. Load: This event is used to track all the different pages/screens a user visits. This would be the most frequently used event in the app. Not only does this event keep track of user browsing history but also combines referral information as well.

```
NewEvent *event       = [[NewEvent alloc]init];

    event.EventName       = @"PageLoad";
    event.Language        = @"Language name";
    event.ReferrerOrigin  = @"internal";
    event.ReferrerMedium  = @"page";
    event.ReferrerType    = @"type";
```

```
    event.EventTimestamp  = @"Timestamp" /* ISO date and time formate
    event.PropertyId      = PiStatsPropertyid;
    event.ContentID       = @"ContentID";
    event.deviceUDID      =  @"device UDID";

[[PistatsManage getInstance]loadviewController:self WithEvent:event];
```

4. Opt-out: In some apps user preference includes switching off the push notification completely. If your app supports such an option, use this event to stop push notifications for the particular user.

```
NewEvent *event             = [[NewEvent alloc]init];

      event.Language              = @"Language name",
      event.SubscribeToTopics        = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
      event.FCMToken                  = @"Firebase token"
      event.UnsubscribeFromTopics   = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
      event.PropertyId                = PiStatsPropertyid;
      event.deviceUDID                = @"device UDID";

[[PistatsManage getInstance]deviceSwitchOffForNotification:PiStatsPropertyid␣
↪WithEvent:event];
```

5. Opt-in: Opposite of the "opt-out" event if the user selects the option to receive notifications in the app, call this event to switch the push notifications back on for the user.

```
NewEvent *event             = [[NewEvent alloc]init];

        event.Language              = @"Language name",
        event.SubscribeToTopics        = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
        event.FCMToken                = @"Firebase token"
        event.UnsubscribeFromTopics   = [[NSArray alloc]initWithObjects:@"topic1",
↪@"topic2", nil];
        event.PropertyId            = PiStatsPropertyid;
        event.deviceUDID          = @"device UDID";

[[PistatsManage getInstance]deviceSwitchOnForNotification:PiStatsPropertyid␣
↪WithEvent:event];
```

## 3.4 iOS - Swift

1. Add the following permissions in the app's plist file:

- Location
- Notifications

2. Add piStats SDK to the X-code project and import two files in project bridging header

```
#import <pistats/PistatsManage.h>
#import <pistats/NewEvent.h>
```

3. Add Firebase SDK: Please follow the instructons in the link below: https://firebase.google.com/docs/cloud-messaging/ios/client

4. Import piStats in AppDelegate

```
import pistats
```

5. Initialize piStats in application:didFinishLaunchingWithOptions

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions␣
↪launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool
   {
     PistasManage.getInstance()(property:PiStatsPropertyid)
   }

}
```

6. Implement observer: Add kPiStatsObserverNotification in application:didFinishLaunchingWithOptions

```
 NotificationCenter.default.addObserver(self, selector: Selector(("pistatObserverCall:
↪")), name: NSNotification.Name.piStatsObserver, object: nil)

func pistatObserverCall(_ notification: Notification) {

    self.registrationCallForPistats()
    self.subscribeCallForPistats()
}
```

After piStats SDK has been initialized we can track the following type of events:

1. Registration :

- After we have received the FCM registration token we need to register the device with piStats for push notification.

```
if let fcmTokenDefaultValue = userDefault.value(forKey:pistatsFCMToken)
    {
        fcmTokenDefaultValue = deviceTokenValue as! String
    }
        let notifnEvent = NewEvent()
        notifnEvent.language = currentLanguage()
        notifnEvent.fcmToken = fcmToken  /* Firebase token Mandatory  */
        if let deviceTokenValue = userDefault.value(forKey:pistatsDeviceToken)
        {
            notifnEvent.deviceUDID = deviceTokenValue as! String
        }
        notifnEvent.proprtyId = pistatsPropertyId
        notifnEvent.eventTimestamp  = getTimeandDateInISO()

  (PistatsManage.getInstance() as AnyObject).register(forDevice: notifnEvent)
```

2. Subscription: Subscription event is used to subscribe and unsubscribe users to push notification topics

- For push notifications the subscription call is best placed as a callback from FCM registration

```
func messaging(_ messaging: Messaging, didRefreshRegistrationToken fcmToken: String) {
     var fcmTokenDefaultValue = ""
     if let deviceTokenValue = userDefault.value(forKey:pistatsFCMToken)
     {
         fcmTokenDefaultValue = deviceTokenValue as! String
     }
```

```
    if(fcmToken != fcmTokenDefaultValue)
    {
        let notifnEvent = NewEvent()
        notifnEvent.language = currentLanguage()
        notifnEvent.fcmToken = fcmToken  /* Firebase token Mandatory  */
        if let deviceTokenValue = userDefault.value(forKey:pistatsDeviceToken)
        {
            notifnEvent.deviceUDID = deviceTokenValue as! String
        }
        notifnEvent.proprtyId = pistatsPropertyId
        notifnEvent.eventTimestamp  = getTimeandDateInISO()

        (PistatsManage.getInstance() as AnyObject).register(forDevice: notifnEvent)
        userDefault.set(fcmToken, forKey:pistatsFCMToken)
    }
}
```

- piStats subscription

```
let event = NewEvent()
  event.language = currentLanguage()
  event.subscribeToTopics = subscribeTopics /* Pass array of all topics */
  event.fcmToken = FCMTokenValue as! String
  event.propertyId = pistatsPropertyId
  event.eventTimestamp = getTimeandDateInISO()

(PistatsManage.getInstance()as AnyObject).subscribe(forNotification:event)
```

- piStats unsubscribe

```
let notifnEvent = NewEvent()
  notifnEvent.language = currentLanguage()
  notifnEvent.subscribeToTopics = subscribeTopics /* Pass array of all subscribe␣
↪topics */
  notifnEvent.unsubscribeFromTopics = unsubscribeTopics /* Pass array of all␣
↪unsubscribe topics */
if let FCMTokenValue = userDefault.value(forKey:PistatsFCMToken) /*Pass FCM token */
  {
      notifnEvent.fcmToken = FCMTokenValue as! String
  }
if let deviceTokenValue = userDefault.value(forKey:pistatsDeviceToken)  /*Pass Device␣
↪token */
  {
     notifnEvent.deviceUDID = deviceTokenValue as! String
  }
  notifnEvent.proprtyId  = pistatsPropertyId

(PistatsManage.getInstance() as AnyObject).subscribe(forNotification:notifnEvent)
```

3. Load: This event is used to track all the different pages/screens a user visits. This would be the most frequently used event in the app. Not only does this event keep track of user browsing history but also combines referral information as well.

```
let event = NewEvent()
   event.eventTimestamp = currentFormattedDate()
   event.language = currentLanguage()
   event.proprtyId = pistatsPropertyId
   event.referrerOrigin = RefferOrigin
```

```
   event.referrerMedium = sectionName
   event.contentID = newsID
 (PistatsManage.getInstance() as AnyObject).loadviewController(self, with:event)
```

4. Opt-out: In some apps user preference includes switching off the push notification completely. If your app supports such an option, use this event to stop push notifications for the particular user.

```
var event = NewEvent()
    event.language = userDefaults.value(forKey:   ChannelType),
    event.subscribeToTopics = [currentChannel]
    event.fcmToken = userDefaults.value(forKey: FCMPushToken),
    event.proprtyId = PiStatsPropertyid
    event.deviceUDID = userDefaults.value(forKey: UserUDID)
    event.eventTimestamp = Utility.getTimeandDateInISO()
PistatsManage.getInstance().deviceSwitchOff(forNotification: event)
```

5. Opt-in: Opposite of the "opt-out" event if the user selects the option to receive notifications in the app, call this event to switch the push notifications back on for the user.

```
var event = NewEvent()
    event.language = userDefaults.value(forKey: ChannelType),
    event.subscribeToTopics = [currentChannel]
    event.fcmToken = userDefaults.value(forKey: FCMPushToken),
    event.proprtyId = PiStatsPropertyid
    event.deviceUDID = userDefaults.value(forKey: UserUDID)
    event.eventTimestamp = Utility.getTimeandDateInISO()
PistatsManage.getInstance().deviceSwitchOn(forNotification: event)
```