Gimbal™ SDK for iOS Documentation

V 1.16

October 2014

Gimbal, Inc.

Gimbal, Inc.
11010 Roselle St, Ste 150
San Diego, CA 92121
U.S.A.

Gimbal™ SDK for iOS Documentation
V 1.16
October 2014

Contents

1 Introduction

1.1 Purpose and scope of this document

This document describes how to use the *Gimbal* framework to develop iOS applications that can benefit from *contextual* Services

1.2 Who should use this document

Any developer who wants to develop a User Context aware application

1.3 Conventions

- *Italics* is used to reflect technical information.
- Casing is important for everything in italics and all examples (font size is used to highlight blocks only).
- We assume the reader is familiar with JSON [JSON].

1.4 Revision history

The following revisions have been made to this document.

Revision	**Date**	**Reason for change**
1.0	June 2012	SDK Documentation
1.1	September 2012	Added place attributes to places in Section 5.2.1
1.2	December 2012	Removed Image Recognition from documentation
1.3	March 2013	Adding Geofence control section
1.4	March 2013	Adding section to disable communications on the client
1.5	April 2013	Added Gimbal push notification
1.7.1	May 2013	Added reference to libz.dylib SDK dependency
1.8	May 2013	Adding section for Disabling selective place monitoring
1.9	June 2013	Added documentation for deleting user data
1.10	June 2013	Added section 7.2.5 for searching content by attributes Added ContentAttributes field to ContentDescriptor type
1.12	July 2013	Added section 9 for logging analytics.
1.14	February 2014	Removed Sandbox environment.
1.15	May 2014	Renaming QRS to Gimbal
1.16	October 2014	Company address change

1.5 Acronyms, abbreviations, and definitions

The following terms are used in this document. Some elements are identified by more than one term.

Term	**Definition**
Gimbal	The name of the Gimbal solution
Organization	The organization the 3rd-party developer represents and is developing the application on behalf of.

1.6 References

[Profile] Gimbal Interests Profiling

[Software License Agreement] http://www.manager.gimbal.com/sdk-license/

2 Overview

2.1 Gimbal Components

You are probably using Gimbal in part to push relevant content to users at the correct time and at the correct location. In the overall Gimbal eco-system, you want to understand the roles and responsibilities of its various components:

Gimbal™ Manager – accessed via web browser to:

- Generate the API key for your iOS application
- Provide shared geofences you want monitored in your iOS application
- Push content you want the user to receive based on time, location, and interests

Gimbal™ SDK for iOS – an SDK for iOS composed of several frameworks which includes header files and binaries used to compile your iOS application for either a device or simulator:

- Runs a service in the background that monitors user activity
- Identifies user interests
- Monitors geofences and notifies the client application of geofence events and content events when the application has configured itself as a listener
- Allows application to retrieve user interests

2.2 What constitutes the Gimbal™ SDK

The SDK contains the following components:

- Gimbal SDK frameworks
    - Common.embeddedframework
    - NetworkServices.embeddedframework
    - ContextCore.embeddedframework
    - ContextLocation.embeddedframework
    - ContextProfiling.embeddedframework
- Documentation
- Sample application source code (Xcode project)
- This document (The Gimbal™ SDK for iOS)

2.3 Overview of the API

The frameworks you will find in the SDK let you enhance your applications with user contextual information including:

- Geofence monitoring (defined by the application, the user or by a representative of your organization in the Gimbal Manager)
- Content delivery to the user triggered by geofence events and/or time
- Access user interests

The SDK exposes these interfaces through a series of modules (frameworks), each exposing a *connector*. The following connectors are available:

- *QLContextCoreConnector* (required)
- *QLContextPlacesConnector* (optional)
- *PRContextInterestsConnector* (optional)

The *QLContextCoreConnector* connector is required for basic operations (solution enablement) and additional connectors can be used as needed. To leverage the connector features the appropriate frameworks are required in your application framework dependencies.

As part of the connectors, user interfaces are included and will be showed to users to let them manage the permissions of features they want to enable or disable. At a high level, users can turn on or off Gimbal. At a more fine-grained level they can manage the following permissions:

- Places
- Interests

For instance, users can turn off places so that your application will no longer receive geofences events but can still access user interests.

The table below explains the mapping between features and modules (frameworks) and how they jointly work:

Feature	Required frameworks	Description
Core	• Common • NetworkServices • ContextCore	• Enable the APIs • Check status of APIs • Display the Permissions UI

Geofencing	Core frameworks plus: • ContextLocation	• Listen for geo-fence events • Listen for content • Retrieve geo-fence event history • Retrieve content history • Create, read, update, and delete user defined places • Retrieve private points of interests • Retrieve and listen for geofencing user permission
Interests	Core frameworks plus: • ContextProfiling	• Retrieve user interests • Retrieve and listen for interests user permission

2.4 Pre-requisites

- SDK & Software versions
  - iOS SDK: 5.0 or later
  - Xcode: 4.3.1, other versions may work but have not been tested
  - You must have obtained an iOS developer account from Apple
  - You must have a provisioning profile that includes any devices that you wish to test with
- Hardware
  - SDK works on iPhone 4 or higher (SDK does not support 3gs, etc.)

3 Setting up your application

3.1 Setting up your application in Gimbal Manager

Obtain an application key (API Key) from the Gimbal Manager Apps Manager page (https://manager.gimbal.com) in order to enable your app to work with Gimbal.

Then log in to Gimbal Manager using the username and password for your organization. Choose the 'My Apps' button in the top navigation. Fill in the appropriate information into the form and click generate.

Field	Expected Value
Identifier	On iOS this is your application's bundleIdentifier.

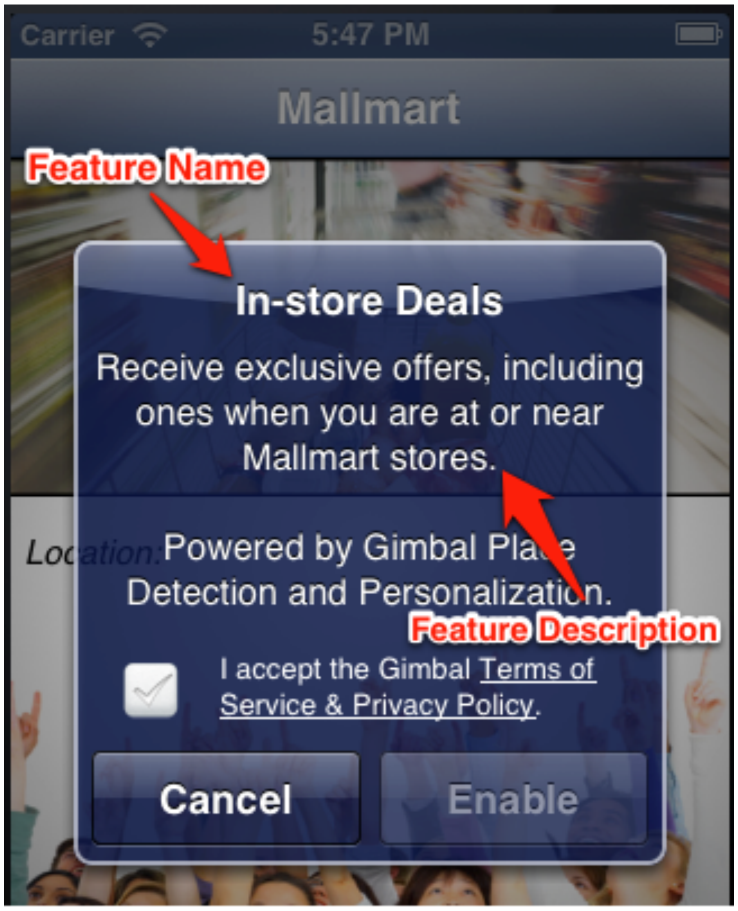
3.2 Configuring Your Application for the SDK

3.2.1 Setting up the UserContext.plist file

The SDK uses a *plist* file in your projects directory, called *UserContext.plist*, to define application specific properties that enable the SDK.

SDK supports the following properties:

Name	**Description**	**Required**
PRODUCTION_API_KEY	Production API key that identifies your application.	Required if APPSTORE_BUILD is YES
FEATURE_NAME	The text you wish to use in your application to describe the value added feature you are providing for your users.	YES
FEATURE_DESCRIPTION	The description you wish to convey to the users about what your application does and any other information you would like the user to know about your application	NO



In-store Deals

Receive exclusive offers, including ones when you are at or near Mallmart stores.

Powered by Gimbal Place Detection and Personalization.

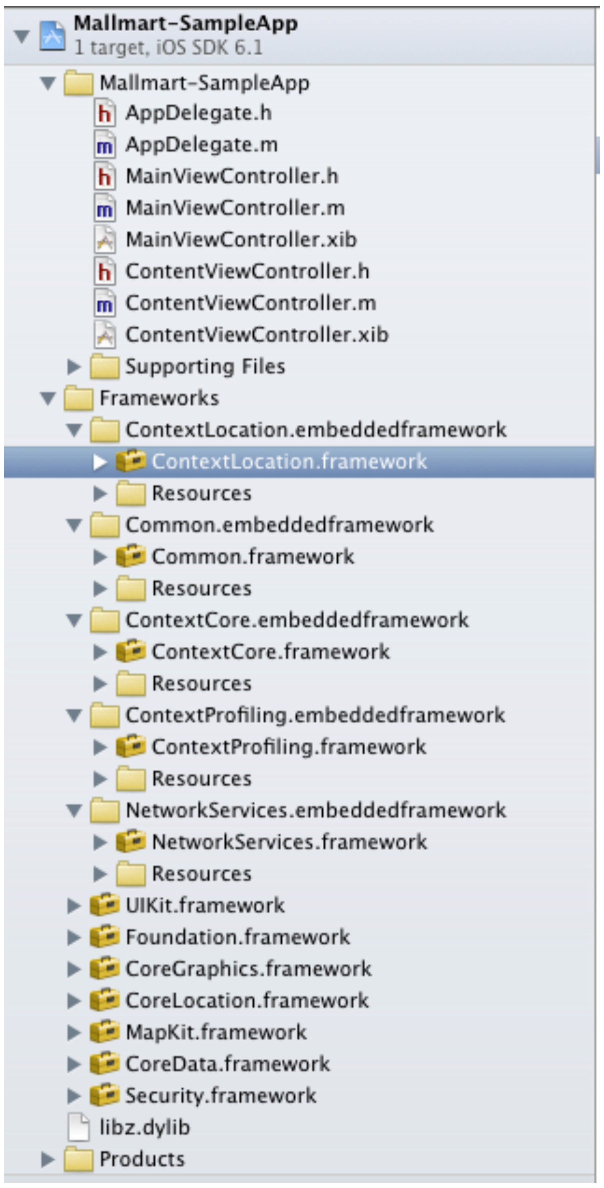
I accept the Gimbal Terms of Service & Privacy Policy.

3.2.2 Importing frameworks into Xcode

The SDK is composed of multiple frameworks, some required and some optional.

- Required
  - Common.embeddedframework
  - NetworkServices.embeddedframework
  - ContextCore.embeddedframework
- Optional
  - ContextLocation.embeddedframework
  - ContextProfiling.embeddedframework

Drag and drop these frameworks from the unzipped SDK folder, into the Frameworks group of your Xcode project as shown in the figure below.



```
Mallmart-SampleApp
1 target, iOS SDK 6.1
  Mallmart-SampleApp
    AppDelegate.h
    AppDelegate.m
    MainViewController.h
    MainViewController.m
    MainViewController.xib
    ContentViewController.h
    ContentViewController.m
    ContentViewController.xib
    Supporting Files
  Frameworks
    ContextLocation.embeddedframework
      ContextLocation.framework
      Resources
    Common.embeddedframework
      Common.framework
      Resources
    ContextCore.embeddedframework
      ContextCore.framework
      Resources
    ContextProfiling.embeddedframework
      ContextProfiling.framework
      Resources
    NetworkServices.embeddedframework
      NetworkServices.framework
      Resources
    UIKit.framework
    Foundation.framework
    CoreGraphics.framework
    CoreLocation.framework
    MapKit.framework
    CoreData.framework
    Security.framework
    libz.dylib
  Products
```

The SDK depends on multiple frameworks that are packaged with the iOS SDK. Here is the list of frameworks that has to be added to your application

- MapKit.framework
- CoreLocation.framework
- Security.framework
- CoreData.framework
- UIKit.framework
- Foundation.framework
- CoreGraphics.framework
- libz.dylib

4 Using the QLContextCoreConnector

The *QLContextCoreConnector* must be enabled prior to using any other features. All calls to the API will return failures with a disabled status message until this step is complete.

4.1 Quick Start

1. Create an instance of the core connector
2. Then call *checkStatusAndOnEnabled:disabled:* on this instance. This will call you back on either the enabled: or the disabled: block
3. If connector is 'disabled', then call *enableFromViewController:success:failure:*. This call will prompt the user with terms of service the first time and show the permission UI screen
4. If connector is 'enabled', then you can start using other connectors (see sections below)

Note that your application must ALWAYS include a button/tab/link in its settings to call *showPermissionsFromViewController:success:failure* so that users can always have the ability to change permissions settings.

4.2 Enable the connector and check its status

The QL*ContextCoreConnector* has two methods used to check the status of and possibly enable the QL*ContextCoreConnector*. The *enableFromViewController:success:failure:* method is used to enable the SDK for use by the end user. The *checkStatusAndOnEnabled:disabled:* method allows your application to ensure that it has been previously enabled.

These calls are asynchronous and use objective-c blocks to return results when they are available.

4.2.1 Check the status

```
[self.contextCoreConnector
      checkStatusAndOnEnabled:
        ^(QLContextConnectorPermissions *contextConnectorPermissions) {
              // do something if enabled
        }
        disabled:^(NSError *error) {
              // enable connector
        }];
```

4.2.2 Enable the connector

```
QLContextCoreConnector *contextCoreConnector = [[QLContextCoreConnector alloc] init];

[contextCoreConnector
  enableFromViewController:self.navigationController
```

```
    success:^{
      // do something when enabled
    }
    failure:^(NSError *error) {
      // Check your credentials and retry
    }];
```

4.3 Next steps

Now that the connector is enabled you can use the places and interests connectors to implement the core functionality of the SDK.

4.4 Deleting a users data and disabling Gimbal

```
[self.contextCoreConnector deleteAllUserDataAndOnSuccess:^{
    NSLog(@”User data deletion SUCCESS”);
} failure:^(NSError *error) {
    NSLog(@”User data deletion FAILURE: %@”, error  );
}];
```

5 Geofence

Your application can receive a geofence event when the user's device enters or exits places you set in the Gimbal Manager (called public places in this section) or defined by your application (private places). The Gimbal Geofence Manager is used to manage public places while the private places are automatically created and managed on the user's device. The user's private places are not available on the Gimbal Manager.

Geofence and Content events happen mostly in the background of your application. So, it's important to initialize the connectors and delegates within your application's *AppDelegate* didFinishLaunchingWithOptions: method or classes that are initialized within this method.

Initializing connectors here allows our SDK to start monitoring for geofence events and deliver content to your application in the background.

Gimbal SDK always runs in the background even when your application is not in the foreground.

5.1 Place monitoring

5.1.1 Initialize connector

```
self.contextPlaceConnector = [[QLContextPlaceConnector alloc] init];
```

5.1.2 Add delegate

Implement the protocol *QLContextPlaceConnectorDelegate* to in order to receive place events, content associated to places, place permission change etc. Add this implementation on *contextPlaceConnector*.

```
self.contextPlaceConnector.delegate = self;
```

5.1.3 Listening for place event

Implement *didGetPlaceEvent* method to listen for place events. Gimbal SDK will call this method when it detects Entry/Exit for a place

```
- (void)didGetPlaceEvent: (QLPlaceEvent *)placeEvent
{
   // do something with the place event
}
```

Note: See section 4.1.1

The following fields are available in the *QLPlaceEvent* passed to the listener:

Field Name	Description

placeType	*QLPlaceTypeOrganization* refers to places created in Context Console and applies to all of your users. A *QLPlaceTypePrivate* is created locally on the phone and only applies to a single user.
eventType	*QLPlaceEventTypeAt* means that the user has arrived at the place. *QLPlaceEventTypeLeft* means that the user has just left the place.
Place	The *QLPlace* object associated with the event
Time	The time of the event in milliseconds since 1970 (see System.currentTimeMillis()).

5.1.4 Checking if place monitoring is available

The Gimbal SDK does not support place monitoring on devices such as the iPhone 3GS due to the hardware not being battery efficient to continually monitor geofences in the background. To determine if the device supports place monitoring, call the *"`isPlaceMonitoringAvailable`"* on the QLContextPlaceConnector to see if the current device supports place monitoring.

5.2 Private Places

A "private place" is a place specific (and private) to the end-user. Your app can create user-defined places. In addition, the Gimbal SDK can automatically create these places from user activities; Gimbal determines the twenty (20) places that a user goes to regularly and/or spends time at.

Each application on a phone that uses Gimbal SDK can create places that are specific to the user of that particular phone. These places are independent of the places created with Gimbal's Geofence Manager. User-defined places are not shared between applications and cannot be managed with Gimbal Geofence Manager. These places can be used to trigger location events.

5.2.1 Creating a new private place

This function lets you create new user-specific places. For example, this could be called by a *MapActivity* that allows the user to select a location and a place radius. The radius must be set to 50 meters or greater.

As of SDK version 0.31, a Place object supports "**place attributes**". A place attribute is a key/value pair that can be used to set custom properties on a place. **Note:** The "id" field must NOT be set when creating a place.

```
- (void)createPlace
{
    QLPlace *place = [[QLPlace alloc] init];
    place.name = @”Home”;
    QLGeoFenceCircle *circle = [[QLGeoFenceCircle alloc] init];
    circle.latitude = 32.893;
    circle.longitude = -117.199;
    circle.radius = 100;
    place.geoFence = circle;

   NSMutableDictionary *placeAttributesDictionary = [[NSMutableDictionary alloc] init];
   [placeAttributesDictionary setValue:@”TYPE” forKey:@”Airport”];
   [placeAttributesDictionary setValue:@”APPLICATION_SPECIFIC_PLACE_ID” forKey:@”101”];

   QLPlaceAttributes *placeAttributes = [[QLPlaceAttributes alloc] initWithPlaceAttributes:placeAttributesDictionary];
   [place setPlaceAttributes:placeAttributes];


    [self.contextPlaceConnector createPlace:place
        success:^(QLPlace *place)
        {
            // do something after place was created successfully
        }
        failure:^(NSError *error)
        {
            // failed with statusCode
        }];
}
```

Depending on the result method that gets called in the callback, you may need to take corrective action.

If *success()* is called, the Place will have an id which will permanently identify the newly created place. If you later need to delete or update the place, this id will be used to identify the correct place.

5.2.2 Getting existing private places

This function will retrieve a list of existing private places.

```
- (void)allPlacesAndOnSuccess
{
    [self.contextPlaceConnector allPlacesAndOnSuccess:^(NSArray *allPrivatePlaces)
    {
        // do something after places were retrieved
    }
    failure:^(NSError *error) {
        // failed with statusCode
    }];
}
```

5.2.3 Updating a private place

The name or location (GeoFence) of an existing place can be changed. The best way to do this is to get the existing place (see above) and change the fields as desired. **Note:** The “id” field cannot be changed.

```
- (void)updatePlace: (QLPlace *)existingPlace
{
  existingPlace.name = @"New home name";

  [self.contextPlaceConnector updatePlace:existingPlace
    success:^(QLPlace *place)
    {
     // do something after place update
    }
    failure:^(NSError *error)
    {
       // failed with statusCode
    }];
}
```

5.2.4 Deleting a private place

An existing place can be deleted. Only the id needs to be provided, not the entire place.

```
- (void)deletePlace: (long long)existingPlaceId

{
  [self.contextPlaceConnector deletePlaceWithId:existingPlaceId
    success:^()
    {
       // do something after place has been deleted
    }
    failure:^(NSError *error)
    {
       // failed with statusCode

    }];

}
```

5.2.5 Retrieving the top 20 private points of interest

```
- (void)allPrivatePointOfInterestAndOnSuccess
{
  [self.contextPlaceConnector allPrivatePointsOfInterestAndOnSuccess:^(NSArray * allPrivatePointsOfInterest)
  {
    // do something after top 20 private points of interest were retrieved
  }
  failure:^(NSError *error) {
    // failed with statusCode
  }];
}
```

5.3 Geofence Filtering

Gimbal SDK allows your application to selectively disable monitoring for certain places. Implement *shouldMonitorPlace* method on *QLContextPlaceConnectorDelegate* to let Gimbal SDK know if you want Gimbal to not monitor a place.

Gimbal SDK calls this method and monitors the place for geofence events if method returns *YES*.

Gimbal SDK decides when to call this method and can call this method multiple times during the day. Make sure the implementation of this method does not have long running operations, as it will affect performance of you application.

5.3.1 Adding delegate method

```
- (BOOL)shouldMonitorPlace: (QLPlace *)place

{
    // Return YES if place needs to be monitored
    // Return NO if place does not need to be monitored

}
```

5.3.2 Remove delegate

You can remove the delegate from the place connector if you no longer wish get notified from place connector.

```
self.contextPlaceConnector.delegate = nil;
```

5.4 Geofence Control

The geofence service can be turned off and on (as long as the user permission allows it) and can be put into foreground only mode or a combined foreground/background mode.

The default is for both foreground and background geofencing to be enabled. To disable geofencing:

```
[self.contextPlaceConnector dontMonitorPlacesWhenAllowed];
[self.contextPlaceConnector dontMonitorPlacesInBackground];
```

To enable geofencing:

```
[self.contextPlaceConnector monitorPlacesWhenAllowed];
[self.contextPlaceConnector monitorPlacesInBackground];
```

In addition, if you know your application will never need background mode, a property can be set in your UserContext.plist file to simplify the user's privacy controls to not include the background option. To disable the background privacy control, add the following property named *BACKGROUND_GEOFENCING_DISABLED* with Boolean value set to *YES*.

6 Interest Sensing

The user’s interests are profiled each day. These interests are defined by a rule set which can take from installed apps and other inputs.

6.1 Quick Start

1. Create an instance of *PRContextInterestsConnector*
2. You can retrieve interests from connector. This will return a *JSON* string with likelihood (confidence) values in the range of 0.0 to 1.0.

6.2 Listening for User Interests

6.2.1 Retrieve the connector

```
self.contextInterestsConnector = [[PRContextInterestsConnector alloc] init];
```

6.2.2 Request the Profile

```
PRProfile *interests = self.contextInterestsConnector.interests;
```

Profile fields:

Field Name	**Field Type**	**Description**
attributes	*NSDictionary*	The attributes of the profile

ProfileAttribute fields:

Field Name	**Field Type**	**Description**
key	*NSString*	Attribute key
attributeCategories	*NSArray* of *PRAttributeCategory*	Attribute categories

AttributeCategory fields:

Field Name	**Field Type**	**Description**
key	*NSString*	Category key

likelihood	*double*	A floating point value between 0 and 1 representing the likelihood that the category applies to the user

6.2.3 Set custom profile attributes

```
PRCustomAttributes *customAttributes = self.contextInterestsConnector.
customAttributes;
if (customAttributes == nil)
       customAttributes = [[PRCustomAttributes alloc] init];
[customAttributes addStringAttribute@”attr-value” forKey:@”attr-name”];
self.contextInterestsConnector.customAttributes = customAttributes;
```

7 Communicate

These are also known as rich media push notifications. They're managed by the Gimbal Communicate Manager and can be targeted to a specific audience by place and/or time.

7.1 Place based communicate

7.1.1 Quick Start

1. Start Listening for any communications coming from the Gimbal Communication services by assigning a delegate (*QLContextPlaceConnectorDelegate*) to *QLContextPlaceConnector*
2. You can retrieve the latest content events with *requestContentHistoryAndOnSucess:success:failure:*
3. Stop Listening by removing the delegate

7.1.1.1 Listening for content

7.1.1.1.1 Initialize connectors

```
self.contextPlaceConnector = [[QLContextPlaceConnector alloc] init];
```

7.1.1.1.2 Start Listening

```
self.contextPlaceConnector.delegate = self;
```

7.1.1.1.3 Implement protocol method

```
- (void)didGetContentDescriptors: (NSArray *)contentDescriptors
{
        // do something with content
}
```

QLContentDescriptor fields:

Field Name	Field Type	Description
title	*NSString*	The title of the content
contentDescription	*NSString*	The description of the content
contentUrl	*NSString*	The content url of the content
campaignId	*NSString*	The campaign id defined by the Gimbal Manager
expires	*NSNumber*	The timestamp when this content expires

displayCount	*NSNumber*	Number of times campaign with 'campaignId' was delivered to your application
placeId	*NSNumber*	The placeId for which content was retrieved
eventTime	*NSNumber*	Latest time of the content with 'campaignId' was delivered to your application
contentAttributes	*QLContentAttributes*	Attributes of the content, defined using Manager API's or Manager UI

7.1.1.1.4 Stop listening

```
self.contextPlaceConnector.delegate = nil;
```

7.1.2 Retrieving content event history

```
[self.contextPlaceConnector
requestContentHistoryAndOnSuccess:^(NSArray *contentHistories)
  {
    // do something with content
  }
  failure:^(NSError *error)
  {
    //failed with error
   }];
```

7.2 Time based communicate

Time based communicate are pushed to iOS devices using Apple push notifications. Your application has to be setup for apple push notifications before these messages can be delivered to your clients. Refer to Section 7.2.1

7.2.1 Setup Gimbal Push

To enable your application to receive push notifications from Gimbal, your application should register its device token received from Apple push service with Gimbal. If registration is successful, your application will be enabled to receive time-based content from Gimbal.

7.2.2 Quick Start

7.2.2.1 Register for Gimbal Push Notification

This step will register your device with Apple push service to enable push notifications. If its successful, the application delegate receives a device token in the application:didRegisterForRemoteNotificationsWithDeviceToken: method; if registration fails it is informed via the application:didFailToRegisterForRemoteNotificationWithError: method.

This method has to be called every time your application finishes launching, because this helps the SDK to keep up-to-date with active device tokens, as device tokens may change.

```
[QLPushNotificationsConnector registerForRemoteNotificationTypes:UIRemoteNotificationTypeAlert |
        UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound];
```

7.2.2.2 Register Device Token

Gimbal, uses device token information, generated by apple push service, to send push notifications. Call this method from application delegate's application:didRegisterForRemoteNotificationsWithDeviceToken: method; This will enabled the device to receive push notifications from Gimbal.

```
[QLPushNotificationsConnector didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
```

7.2.2.3 Listen to Remote Notifications

Gimbal, has to be notified when your application receives push notification from Apple, so that relevant content can be pushed to the devices. Call this method from application delegate with the parameters you received in application:didReceiveRemoteNotification: method; SDK will call you back with relevant content, refer to section 7.2.3 for listening for content.

```
[QLPushNotificationsConnector didReceiveRemoteNotification:userInfo];
```

7.2.2.4 Listen to Application Launch

Call this method, from application delegate with the parameters your received in application:didFinishLaunchingWithOptions: method; to let the SDK know that application was launched.

```
[QLPushNotificationsConnector didFinishLaunchingWithOptions:launchOptions];
```

7.2.2.5 Unregister Gimbal Push Notification

Push notifications can be stopped on the client using the following API. But note that it should be called only if your application no longer wants to support Gimbal push notifications.

```
[QLPushNotificationsConnector unregisterForRemoteNotifications];
```

7.2.3 Listening for Content

7.2.3.1 Initialize connectors

```
self.contentConnector = [[QLContentConnector alloc] init];
```

7.2.3.2 Start Listening

```
self.contentConnector.delegate = self;
```

7.2.3.3 Implement protocol method

```
-(void) didReceiveNotification: (QLContentNotification *)notification
                        appState: (QLNotificationAppState)appState
{
    // do something with notification.
    // You can fetch detailed content information, using contentId. Refer to Section 7.2.4
}
```

QLNotification fields

Field Name	Field Type	Description
message	*NSString*	The message of the content
contentId	*NSString*	The contentId of the content

7.2.4 Fetching Content

Content related to delivered push notifications can be fetched from the server. Basic content is represented by *QLContent* class and consists of following fields:

Field Name	Field Type	Description
identifier	*NSString*	The id of the content
title	*NSString*	The title of the content
contentDescription	*NSString*	The description of the content
contentUrl	*NSString*	The contentUrl of the content
campaignId	*NSString*	The campaignId of the content
expires	*NSDate*	The timestamp when the content expires
contentAttributes	*QLContentAttributes*	Attributes of the content, defined using Manager API's or Manager UI

7.2.4.1 Fetching content by ID

Content related to the notification clicked by the user can be fetched from server, using *contentId*. The *contentId* will be delivered to your application as part of the notification on *contentConnector* delegate.

```
[self.contentConnector contentWithId: identifier
                                success:^(QLContent *content)
                                {
                                    // do something with notification.
                                }
```

```
                                    failure:^(NSError *error)
                                    {
                                        // failed to fetch content.
                                    }];
```

7.2.4.2 Fetching content based on time range

Content related to notifications that user never clicked can be fetched from the server by providing a date range. This call fetches content related to all time based communications that were triggered for the client during the provided time range.

```
[self.contentConnector timeContentsFromStartDate:startDate
                                       toEndDate:endDate
                                         success:^(NSArray *timeContents) {
                                                  // do something with content array
                                         } failure:^(NSError *error) {
                                                  // failed to fetch content
                                         }];
```

The array of content consists of elements of type *QLTimeContent.* In addition to all the fields from *QLContent*, *QLTImeContent* has following additional fields:

Field Name	Field Type	Description
lastTriggerTime	*NSDate*	The timestamp when the last time trigger occurred

7.2.5 Content Search

7.2.5.1 By attribute

Content is fetched from the server based on the Query passed into this method. There are different implementations for QLQuery.

- *QLQueryForAnyAttributes*: Fetches content that matches any of the attributes defined in the query.

```
QLQueryForAnyAttributes *queryForAnyAttributes = [[QLQueryForAnyAttributes alloc] init];
[queryForAnyAttributes whereKey:@"hours-open" containsStringValue:@"10-13"];
[queryForAnyAttributes whereKey:@"hours-open" containsStringValue:@"17-20"];
[queryForAnyAttributes whereKey:@"type" containsStringValue:@"restaurant"];

[self.contentConnector contentsWithQuery:queryForAnyAttributes
                        success:^(NSArray *contents) {
                          // do something with the content array
                        } failure:^(NSError *error) {
                          // failed to search for content
                        }];
```

The above example will fetch content that matches any of the specified key/value pairs. More specifically, the key must be an exact match and the value must be *contained* within the attribute value.

7.3 Disabling Communications

Fetching of content from the server can be disabled on the client if communications are not used. To disable communications on the client add the property COMMUNICATIONS_DISABLED to UserContext.plist with Boolean value set to *YES*.

8 Permissions (Gimbal Privacy)

8.1 Listening for permission change

The user can control the kinds of information that they are willing to share with your application. It is important that your app gracefully fail should the user remove your app's access to Gimbal's functionality.

QLContextCoreConnector, *QLContextPlaceConnector and PRContextInterestsConnector* need permissions that have to be enabled by the users for your application to work. The user can change these permissions at any time.

You can request the current permissions the user has granted your application and listen to changes that the user makes to your app's permissions using the corresponding connector.

8.1.1 Start Listening for subscription permission

Permission that allows your application to use Gimbal services

```
self.contextCoreConnector.permissionsDelegate = self;
```

8.1.2 Implement protocol method

```
- (void)subscriptionPermissionDidChange: (BOOL)subscriptionPermission
{
        // do something when subscription permission changed
}
```

8.1.3 Start Listening for location permission

Permission that allows your application to receive geofence events and location based campaigns.

```
self.contextPlaceConnector.delegate = self;
```

8.1.4 Implement protocol method

```
- (void)placesPermissionDidChange: (BOOL)placesPermission
{
        // do something when location permission changed
}
```

Note: See section 4.1.1

In a nutshell, if *subscriptionPermission* or *placesPermission* is false, your application will not receive geofence events. In addition any campaign events triggered by a change in the user's location will not apply to this particular user until they turn the permission back on. It is in the interest of your application to drive the user to allow this permission for maximum efficiency.

If *enabled* or *profileEnabled* is 'false', this means that any campaign that relies on profile attributes (like Age, Gender, Income, etc.) will not match this user. User Context will assume that this user profile is generic and fits all categories. It is in the interest of your application to encourage the enabling of profile permissions so that user receives very focused/relevant content.

8.1.5 Stop listening

```
self.contextPlaceConnector.delegate = nil;
self.contextCoreConnector.permissionsDelegate  = nil;
```

9 Analytics

9.1 About

Analytic APIs allow developers to log different type of events. Each event requires data to be provided by the developer for logging the events successfully. Logged events are later converted into meaningful reports in the Manager module.

9.2 Types of Analytic Events

There are 3 different types of QLAnalyticEvent. Use one of the following subtypes of QLAnalyticEvent to log the desired event.

9.2.1 QLContentNotifiedEvent, QLContentClickedEvent, and QLContentDisplayedEvent

Use one of these three events to log, based on what you would like to log.

Field Name	**Field Type**	**Description**
time	*NSDate*	The timestamp when the user was notified, clicked, or displayed of the content. Current time is initialized by default.
trigger	*QLPlaceTrigger*	PlaceTrigger lets you set id of the place that triggered notified event. This field is optional.
contentId	*NSString*	Identifier of the content you would like to log. **This field is mandatory.**

9.3 Analytics Connector

Exposes a static method to log analytic events. Errors are reported back to the developer via the error reference object passed in as a part of the API.

Note: If you’re not interested in error object, just pass NULL for error: parameter. Errors will be just logged on the console.

```
QLPlaceTrigger *placeTrigger = [[QLPlaceTrigger alloc] initWithPlaceId:@"place-id"];
QLContentNotifiedEvent *event = [[QLContentNotifiedEvent alloc] initWithContentId: :@"content-id"
                                                   trigger:placeTrigger];
NSError *error = nil;
[QLAnalyticsConnector log:event error:&error];
```

10 Getting events when your application is not running

Gimbal SDK monitors place events in the background, you can listen (with a delegate) for place events and take action, such as putting up a notification when appropriate.

Be aware that if you are not declared as a continuous background application, that you will have very little time to actually take any action during the callback – total time including the time SDK has already used is 10 seconds.

If you are a continuous background application, you can start listening for location changes (and ignore them) during these callbacks and you will typically have 10 minutes of processing time. Using this mechanism to gain additional time will require a warning for your application in the App Store and if your actual event processing takes significant time, the battery drain will be significant.

11 Error codes and messages

To make the development of your application easier, Gimbal application sends back meaningful error codes and error messages. Reference to these can be found in ContextConnectorError documentation.