Gimbal™ SDK for Android Documentation

V 1.18

October 2014[1]

Gimbal, Inc.

1

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

Gimbal™ SDK for Android Documentation
V 1.18
October 2014

Contents

1 Introduction

1.1 Purpose and scope of this document

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

1.2 Who should use this document

Any developer who wants to develop a 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.

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	July 2012	• Deprecated *PrivacyControlListener* • Deprecated *PrivacyControlChangeNotifier* • Added new attribute *app.capabilities* in Section 3.3.1 • Alljoyn.jar no longer needed
1.2	September 2012	• Added place attributes to places in Section 5.5.1
1.3	October 2012	• Added InterestChangeListener in Section 6.4
1.5	November 2012	• Added setCurrentActivity in Section 4.4
1.6	November 2012	• Added setCustomActivityIndicator in Section 7.4
1.7	November 2012	• Updated to reflect project and dependencies changes.
1.8	December 2012	• Removed IR documentation
1.9	February 2013	• Updated references to deprecated code
1.10	March 2013	• Added information about foreground only mode and the associated privacy controls
1.11	March 2013	• Added section to disable communications on the client
1.13	June 2013	• Added documentation for deleting user data

Revision	Date	Reason for change
1.14	June 2013	• Added ContentAttributes to ContentDescriptor class in Section 7.2 • Added documentation for content search by attributes in Section 7.4
1.15	July 2013	• Added ContextAnalyticsConnector in section
1.16	February 2014	• Removed Sandbox environment.
1.17	May 2014	• Renaming Qualcomm Retail Solutions, Inc to Gimbal, Inc
1.18	October 2014	• Company address change.

1.5 Acronyms, abbreviations, and definitions

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

Term	Definition
Gimbal	The name of the Gimbal SDK
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.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 ecosystem, 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 Android application
- Provide shared geofences that you want your application to monitor
- Push content you want the user to receive based on time, location and interests

Gimbal™ SDK for Android – an SDK for Android jar files and libraries used by the Android application on the device:

- 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 jars and libraries
    - Context-Core jar
    - Context-Places jar
- Documentation
    - JavaDoc for the interface library (two sets of javadoc)
    - A sample application source code (Eclipse project)
    - This document (The Gimbal™ SDK for Android)

2.3 Overview of the API

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

- Geofence monitoring (defined by the application/user or by you 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 (jars), each exposing a connector. The following connectors are available:

- *ContextCoreConnector* (required)
- *ContextPlacesConnector* (optional)
- *ContextInterestsConnector* (optional)

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

As part of the connectors, user interfaces are included and will be showed to users to let them manage 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.

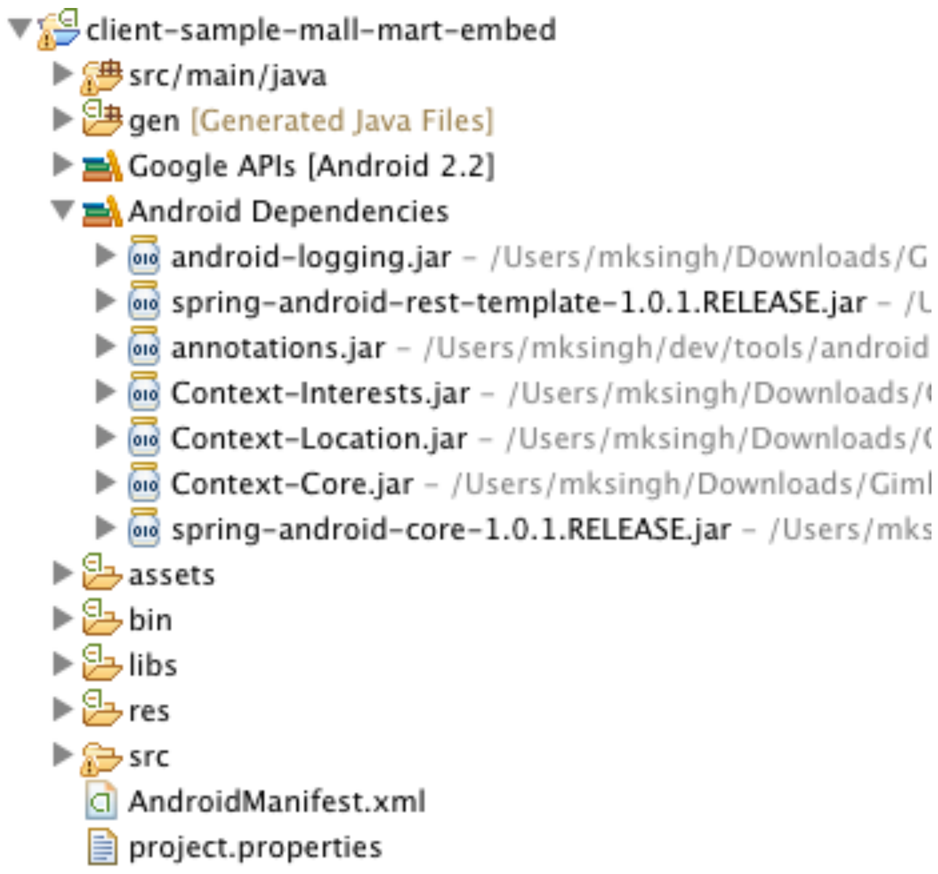
The table below explains the mapping between features and modules (JARs) and how they jointly work:

Feature	Required jars/libraries	Description
Core	• Context-Core.jar	• Enable the APIs • Check status of APIs • Display the Permissions UI • Listen for content • Retrieve content history • Listen for core permission changes
Geofencing	• Context-Core .jar • Context-Location.jar	• Listen for geofence events • Retrieve geofence event history • Create, read, update and delete user-defined places • Retrieve private points of interests • Listen for geofence permission changes
Interests	• Context-Core .jar • Context-Interests.jar	• Retrieve user interests • Listen for interests permission change

2.4 Pre-requisites

- Our sample project ships as an Eclipse project, the Eclipse Indigo (3.7) or higher release is required. Latest eclipse can be found here:
    - http://www.eclipse.org/downloads/
- You must use the ADT plug-in from Android located here:
    - http://developer.android.com/sdk/eclipse-adt.html
- You also need an Android device running Android version 2.2 or higher (we do not support the Android emulator)
- The Gimbal SDK depends on the following Open Source libraries (These jar's are included in the sample Eclipse project in SDK):
    - spring-android-rest-template-1.0.1.RELEASE.jar that you can download here.
        - http://repo.springsource.org/libs-release-local/org/springframework/android/spring-android/1.0.1.RELEASE/spring-android-1.0.1.RELEASE-dist.zip
    - spring-android-core-1.0.1.RELEASE.jar that you can download here.
        - http://repo.springsource.org/libs-release-local/org/springframework/android/spring-android/1.0.1.RELEASE/spring-android-1.0.1.RELEASE-dist.zip

If you are using Eclipse, these libraries are present in the "libs" directory of the project. This should cause the jars to be listed in the "Android Dependencies". Your eclipse environment should look like the following:



```
client-sample-mall-mart-embed
  src/main/java
  gen [Generated Java Files]
  Google APIs [Android 2.2]
  Android Dependencies
    android-logging.jar - /Users/mksingh/Downloads/G
    spring-android-rest-template-1.0.1.RELEASE.jar - /L
    annotations.jar - /Users/mksingh/dev/tools/android
    Context-Interests.jar - /Users/mksingh/Downloads/(
    Context-Location.jar - /Users/mksingh/Downloads/(
    Context-Core.jar - /Users/mksingh/Downloads/Giml
    spring-android-core-1.0.1.RELEASE.jar - /Users/mks
  assets
  bin
  libs
  res
  src
  AndroidManifest.xml
  project.properties
```

3 Setting up your application

3.1 Two steps set-up

To get your application up and running with Gimbal you need to go through the following two steps:

1. Get an application key on Gimbal Manager
    a. You will need to give your application package name
2. Set up some properties for your application

3.2 Get your application key

Obtain an application key (API Key) from the Gimbal Manager Setup 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 'Setup' button in the top navigation. Fill the appropriate information into the form and click generate.

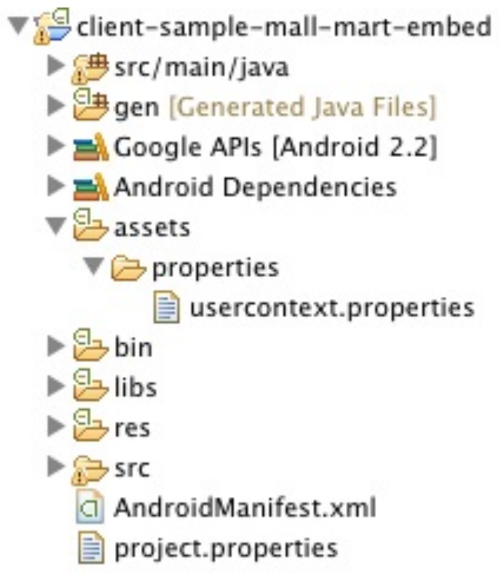
Field	Expected Value
Identifier	On Android this is your application package name, must match what you have configured in your manifest

3.3 Configuring Your Application for the SDK

3.3.1 Setting up the usercontext.properties file

The SDK uses a property file in your assets directory, *usercontext.properties*, to define application-specific properties that enable the SDK.

The file must be located within the Android project in the */assets/properties/* directory to ensure it gets picked up by the SDK as depicted below:



```
▼ client-sample-mall-mart-embed
    ▶ src/main/java
    ▶ gen [Generated Java Files]
    ▶ Google APIs [Android 2.2]
    ▶ Android Dependencies
    ▼ assets
        ▼ properties
            usercontext.properties
    ▶ bin
    ▶ libs
    ▶ res
    ▶ src
      AndroidManifest.xml
      project.properties
```

The following properties are supported:

Name	**Description**	**Required**
app.key	The application key that identifies your application	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
app.capabilities	A comma separated list of the Gimbal capabilities you would like your application to expose. Currently the capabilities are limited to: *Geofence* and *Interests*	No (please, refer to section 3.3.3) Please note that capabilities are case sensitive

3.3.2 Setting up your AndroidManifest.xml

In your AndroidManifest.xml you will need to configure certain elements depending on the functionality you are using.

3.3.2.1 Required Elements

The following elements are required for basic operation.

```

      <service
            android:name="com.qualcommlabs.usercontext.service.GimbalService"
                                                    android:exported="false">
            <intent-filter>
                  <action
                        android:name="<YOUR PACKAGE NAMEHERE>
                                          .service.USER_CONTEXT_SERVICE" />
            </intent-filter>
      </service>

      <receiver
            android:name="com.qualcommlabs.usercontext.service.UserContextServiceStar
            tStopReceiver" android:enabled="true" >
            <intent-filter>
                  <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
            <intent-filter>
                  <action android:name="android.intent.action.ACTION_SHUTDOWN" />
            </intent-filter
      </receiver>

```

3.3.2.2 Location Plugin Permissions

The following are permissions you would need to put in your Android Application Manifest based on the plugin you include in your application.

3.3.2.2.1 Location Plugin Permissions

```
      <!-- REQUIRED -->
      <uses-permission android:name="android.permission.INTERNET" />
      <uses-permission android:name="android.permission.READ_PHONE_STATE" />
      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
      <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
      <uses-permission android:name="android.permission.WAKE_LOCK" />
      <uses-permission android:name="android.permission.BATTERY_STATS" />
      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
      <uses-permission android:name="android.permission.WRITE_SETTINGS" />
      <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETE" />

      <!-- OPTIONAL -->
      <uses-permission android:name="android.permission.VIBRATE" />
```

3.3.2.2.2 Interests Plugin Permissions

```
<!-- REQUIRED -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />

<!-- OPTIONAL -->
<uses-permission android:name="android.permission.VIBRATE" />
```

3.3.3 Enabling optional components and privacy controls

The Gimbal SDK comes with several components that you can choose to use or to omit from your application. To enable these, enumerate them as *app.capabilities* in your *usercontext.properties* file.

For example, if your application needs to use both *ContextPlaceConnector* and the *ContextInterestsConnector*, you want the following property entry:

```
app.capabilities = Geofence, Interests
```

The system will dynamically load these components and will register corresponding privacy controls for each one to let users control them. Please note that these capabilities are case sensitive.

4 Using the ContextCoreConnector

The *ContextCoreConnector* is required to use Gimbal features. Each call to the API will return failures with a disabled status until this step is complete.

4.1 Quick Start

1. Obtain an instance of the core connector with the *ContextCoreConnectorFactory*
2. Then call *isPermissionEnabled()* on this instance to check if connector is enabled
3. If connector permission is not enabled, then call *enable()*. This call will prompt the user with terms of service the first time.
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 *showUpdatePermissionsUI()* so that users can always have the ability to change permissions settings.

4.2 Obtain an instance

First, you need to obtain an instance of *ContextCoreConnector* and enable it. The first time it is enabled, it tries to register with the Gimbal server using your applications package name and API Key:

```
private ContextCoreConnector contextCoreConnector;

@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	contextCoreConnector = ContextCoreConnectorFactory.get(this);
}
```

4.3 Enable the connector and check its status

The *ContextCoreConnector* has two methods used to check the status of, and enable, the *ContextCoreConnector*. The *enable()* method is used to enable the SDK for use by the end user. The *isPermissionEnabled()* method allows your application to ensure that it has been previously enabled.

4.3.1 Check if the permission enabled

```
if (contextCoreConnector.isPermissionEnabled()) {
		startService();
		setupListeners();
}
else {
		ContextCoreStatus status = contextCoreConnector.getStatus();
		toastAndLogError(status.getStatusMessage());
		enableCoreConnector();
}
```

4.3.2 Enable the connector

```
 contextCoreConnector.enable(this, new Callback<Void>() {
		@Override
		public void success(Void responseObject) {
			// do something when successfully enabled
		}
		@Override
		public void failure(int statusCode, String errorMessage) {
			// failed with statusCode
		}
});
```

4.4 Notify the connector with the current activity

To allow Gimbal to optimize location detection and balance battery consumption based on users' current interaction with your application, use the method setCurrentActivity() on *ContextCoreConnector* to notify Gimbal when the current activity changes.

4.4.1 In your onResume() call

```
contextCoreConnector.setCurrentActivity(this);
```

4.4.2 In your onPause() call

```
contextCoreConnector.setCurrentActivity(null);
```

4.5 Next steps

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

4.6 Deleting a users data and disabling Gimbal

```
        contextCoreConnector.deleteAllUserData(new Callback<Void>() {

                                        @Override
                                        public void success(Void responseObject) {
                                            LOG.info("Delete User Data COMPLETE");
                                        }

                                        @Override
                                        public void failure(int statusCode, String errorMessage) {
                                            LOG.info("Delete User Data FAILED");
                                        }

                                    });
```

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). Public places are managed at the Gimbal Geofence Manager while the private places while are automatically created and managed on the user's device. The user's private places are not available on the Gimbal Manager.

5.1 Quick Start

1. Obtain an instance of place connector with *ContextPlaceConnectorFactory*.
2. Start listening with a *new PlaceEventListener()*.

5.2 Check to see if user has enabled use of the connector

The state of the *ContextPlaceConnector* can be determined by calling the method *isPermissionEnabled() and isBackgroundPermissionEnabled()*.

```
        LOG.info("The current contextPlaceConnector enabled state is: "
                        + contextPlaceConnector.isPermissionEnabled());
        LOG.info("The current contextPlaceConnector enabled state is: "
                        + contextPlaceConnector. isBackgroundPermissionEnabled());
```

5.3 Keep Gimbal informed about activity changes

The user can decide to allow or disallow geofencing and in addition, can choose to only allow geofencing when your application is in the foreground. Gimbal relies on each activity in your application to tell it when that activity becomes active (generally onResume()) or inactive (generally onPause()). It is important to make these calls in each of your activities.

```
        @Override
        protected void onResume() {
                super.onResume();
                contextCoreConnector.setCurrentActivity(this);
        }

        @Override
        protected void onPause() {
                super.onPause();
                contextCoreConnector.setCurrentActivity(null);
        }
```

5.4 Listening for geofence events

5.4.1 Obtain an instance

```
contextPlaceConnector = ContextPlaceConnectorFactory.get(this);
```

5.4.2 Start Listening

```
PlaceEventListener placeEventListener = new PlaceEventListener() {
        @Override
        public void placeEvent(PlaceEvent placeEvent) {
                // do something with the place event
        }
};
contextPlaceConnector.addPlaceEventListener(placeEventListener);
```

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

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

5.4.3 Stop listening

```
contextPlaceConnector.removePlaceEventListener(placeEventListener);
```

5.5 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; as of the time of this writing, 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.

5.5.1 Creating a new private place

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

Since 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.

```
public void createPrivatePlace() {
        GeoFenceCircle circle = new GeoFenceCircle();
        circle.setLatitude(32.893);
        circle.setLongitude(-117.199);
        circle.setRadius(100);

        HashMap<String, String> placeAttributes = new HashMap<String, String>();
        placeAttributes.put("TYPE", "Airport");
        placeAttributes.put("APPLICATION_SPECIFIC_PLACE_ID", "101");

        PlaceAttributes placeAttribute = new PlaceAttributes(placeAttributes);
        Place place = new Place();
        place.setName("New user specific place");
        place.setGeoFence(circle);
        place.setPlaceAttributes(new PlaceAttrubutes(placeAttributes));

        contextPlaceConnector.createPlace(place, new Callback<Place>() {
            @Override
            public void success(Place place) {
                // do somethind with place
            }
            @Override
            public void failure(int statusCode, String errorMessage) {
                // 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.

5.5.2 Getting existing private places

This function will retrieve a list of existing private places.

```
	public void getPlaces() {
		contextPlaceConnector.allPlaces(new Callback<List<Place>>() {
			@Override
			public void success(List<Place> place) {
				// do something with place
			}
			@Override
			public void unexpectedFailure(int statusCode,
								String errorMessage){
				// failed with statusCode
			}
		});
	}
```

5.5.3 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.

```
	@Override
	public void updatePlace(Place existingPlace) {
		existingPlace.setName("New place name");
		contextPlaceConnector.updatePlace(place, new Callback<Place>() {
			@Override
			public void success(Place place) {
				// do something with place
			}
			@Override
			public void failure(int statusCode, String errorMessage) {
				// failed with statusCode
			}
		});
	}
```

5.5.4 Deleting a private place

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

```
	public void deletePlace(Long placeId) {
		contextPlaceConnector.deletePlace(placeId, new Callback<Void>() {
			@Override
			public void success(Void place) {
				// place successfully deleted
			}
			@Override
			public void failure(int statusCode, String errorMessage) {
				// failed with errorCode
			}
		});
```

```
    }
```

5.5.5 Retrieving the top 20 private points of interest

```
public void allPrivatePointsOfInterest() {
        contextPlaceConnector.allPrivatePointsOfInterest(new
                                    Callback<List<PrivatePointOfInterest>>() {
              @Override
              public void success(List<PrivatePointOfInterest>) {
                   // place successfully deleted
              }
              @Override
              public void failure(int statusCode, String errorMessage) {
                   // failed with errorCode
              }
        });
}
```

5.6 Geofence Performance

The geofence service has been optimized for good location events with minimal battery consumption. The default setting should be sufficient for most applications.

The operation of the geofence service can be tuned if your application requires different behavior. To change the performance and the battery consumption, call *ContextPlaceConnector. setPerformanceLevel ()* with any of the values defined in the *PlacePerformanceLevel* enum.

To increase the timeliness of place events at the cost of higher battery drain, use *HIGH_PERFORMANCE_MODERATE_BATTERY_CONSUMPTION*. This highest consumption value should only be used for short, well-defined periods of time.

5.6.1 Example setting the place exit timeliness level

```
contextPlaceConnector.setPerformanceLevel(
        PlacePerformanceLevel.OPTIMIZED_PERFORMANCE_LOW_BATTERY_CONSUMPTION);
```

5.7 Geofence Filtering

Gimbal SDK allows your application to selectively disable monitoring for certain places. Implementing ContextPlaceMonitoringFilter, lets your application decide if the place has to be monitored or not. Returning true, tells the SDK to monitor the place for place events.

Gimbal decides when to call on this filter and can be called multiple times during the day.

Make sure the implementation of this method does not have long running operations, as it affects the performance of your application.

5.7.1 Adding Place Monitoring Filter

```
ContextPlaceMonitoringFilter placeMonitoringFilter = new
ContextPlaceMonitoringFilter() {
    @Override
    public boolean shouldMonitor(Place place) {
      return true;
    }
};
contextPlaceConnector.addPlaceMonitorFilter(placeMonitoringFilter);
```

5.7.2 Removing Place Monitoring Filter

```
contextPlaceConnector.removePlaceMonitorFilter(placeMonitoringFilter);
```

5.8 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:

```
contextPlaceConnector.dontMonitorPlacesInBackground();
contextPlaceConnector.dontMonitorPlacesWhenAllowed();
```

To enable geofencing:

```
contextPlaceConnector.monitorPlacesInBackground();
contextPlaceConnector.monitorPlacesWhenAllowed();
```

In addition, if you know your application will never need background mode, a property can be set in your usercontext.properties file to simplify the user's privacy controls to not include the background option. To disable the background privacy control, add this line:

```
background.geofencing.disabled=true
```

6 Interest Sensing

The user's interests are profiled each day (when the device is plugged in at night). These interests are defined by a rule set which can derived from the web browsing history, installed apps, and other inputs. You cannot use Gimbal to obtain a user's browsing history, but you can use Gimbal's Interests Sensing to understand what the user is interested in.

6.1 Quick Start

1. Obtain a connector with *ContextInterestsConnectorFactory*.
2. Request the user's interests with *requestProfile()*. This will return a JSON string with likelihood (confidence) values in the range of 0.0 to 1.0.

6.2 Check to see if user has enabled use of the connector

The state of the *ContextInterestsConnector* can be determined by calling the method *isPermissionEnabled()*.

```
        LOG.info("The current contextInterestsConnector enabled state is: "
                                  + contextInterestsConnector.isPermissionEnabled());
```

6.3 Retrieving User Interests

6.3.1 Retrieve the connector

```
contextInterestsConnector = ContextInterestsConnectorFactory.get(this);
```

6.3.2 Request the Profile

```
contextInterestsConnector.requestProfile(new Callback<Profile>() {
            @Override
            public void success(Profile profile) {
                // do something with profile
            }
            @Override
            public void failure(int statusCode, String errorMessage) {
                // failed with errorCode
            }
        });
```

Profile fields:

Field Name	Field Type	Description
attributes	*Map<String, ProfileAttributes>*	The attributes of the profile

ProfileAttribute fields:

Field Name	Field Type	Description
Key	*String*	Attribute key
attributeCategories	*List<AttributeCategory>*	Attribute categories

AttributeCategory fields:

Field Name	Field Type	Description
Key	*String*	Category key
likelihood	*double*	A floating point value between 0 and 1 representing the likelihood that the category applies to the user

6.4 Listening for User Interests

InterestChangeListener is notified when the profile is generated. They do not necessarily mean that profile has changed since it was notified previously.

6.4.1 Start Listening

```
InterestChangeListener interestChangeListener = new InterestChangeListener() {
        @Override
        public void interestChanged(Profile profile) {
                // do something with the profile
        }
};
contextInterestConnector.addInterestChangeListener(interestChangeListener);
```

6.4.2 Stop Listening

```
contextInterestConnector.removeInterestChangeListener(interestChangeListener);
```

6.5 Set custom profile attributes

```
CustomAttributes customAttributes =
contextInterestConnector.getCustomAttributes();
If (customAttributes == null)
        customAttributes = new CustomAttributes();
customAttributes.addStringValue(“attr-name”, “attr-value”);
contextInterestConnector.setCustomAttributes(customAttributes);
```

7 Communicate

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

7.1 Quick Start

1. Start Listening for any communications coming from the Gimbal Communication services with a new *ContentListener()*.
2. You can retrieve the latest content events with *requestContentHistory()*.
3. Stop Listening with *removeContentListener()*.

7.2 Listening for content

7.2.1 Start Listening

```
ContentListener contentListener = new ContentListener() {
        @Override
        public void contentEvent(ContentEvent contentEvent) {
                // do something with content
        }
};

contextCoreConnector.addContentListener(contentListener);
```

Note that you retrieve content by attaching the content listener to the **core connector**. In the future, this will evolve and will no longer be attached to the core connector.
Content can be received upon a geofence event, as well as upon time.

ContentEvent fields:

Field Name	**Description**
type	String representing the even type, i.e. Geofence, Time Trigger, etc.
content	List of content descriptor objects
time	EPOCH timestamp

ContentDescriptor fields:

Field Name	Field Type	Description
title	String	The title of the content
contentDescription	String	The description of the content
contentUrl	String	The content url of the content
campaignId	String	The campaign id defined by the Gimbal Manager
expires	Long	The timestamp when this content expires
contentAttributes	ContentAttributes	Attributes of the content, defined using Manager API's or Manager UI

7.2.2 Stop listening

```
contextCoreConnector.removeContentListener(contentListener);
```

7.3 Retrieving latest content events

```
contextCoreConnector.requestContentHistory(
        new Callback<List<ContentDescriptorHistory>>() {
            @Override
            public void success(List<ContentDescriptorHistory>
              contentDescriptors) {
                // do something with content
            }
            @Override
            public void failure(int statusCode, String errorMessage) {
                // failed with statusCode
            }
    });
```

The *ContentDescriptorHistory* object is similar to a *ContentEvent* as it contains a *ContentDescriptor*. But unlike the *ContentEvent* class, it contains only one *ContentDescriptor* with a timestamp on when it was last delivered to the user and how many times in total.

7.4 Content Search

7.4.1 By attributes

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

- *QueryForAnyAttributes*: Fetches contents that have any of the attributes defined in the query.

```
ContextContentConnector contextContentConnector =
ContextContentConnectorFactory.get(context);

QueryForAnyAttributes query = new QueryForAnyAttributes();
query.whereAttributeContains("hours-open", "10-13");
query. whereAttributeContains("hours-open", "17-20");
query. whereAttributeContains("type ", "restaurant");

contextContentConnector.searchContentWithQuery(query, new
Callback<List<ContentDescriptor>>() {

      @Override
      public void success(List<ContentDescriptor> responseObject) {
              // do something with content
      }

      @Override
      public void failure(int statusCode, String errorMessage) {
              // do something with content
      }
});
```

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

7.5 Disabling Communications

Fetching of content from the server can be disabled on the client if communications are not used. Add the following property in usercontext.properties to disable communications on the client

```
communications.disabled=true
```

8 Analytic

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.

8.1 Types of Analytic Events

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

8.1.1 ContentNotifiedEvent, ContentClickedEvent, and ContentDisplayedEvent

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

Field Name	Field Type	Description
time	*Long*	The timestamp when the user was notified, clicked, or displayed of the content. Current time is initialized by default.
trigger	*Trigger*	Initialize your trigger with one of the 2 predefined Triggers (PlaceTrigger or TimeTrigger) if you would like to have Gimbal generate reports on them in the future.
contentId	*String*	Identifier of the content you would like to log. **This field is mandatory.**

8.2 Calling ContextAnalyticsConnector to Log Analytic Events

To log analytic events using the ContextAnalyticsConnector, the sample code to call the log method is bellow; follow by the descriptions how to use it.

```
ContentClickedEvent analyticEvent = new ContentClickedEvent();
analyticEvent.setContentId("TEST_CONTENT_ID");
PlaceTrigger trigger = new PlaceTrigger();
trigger.setId("1234-asdf-1234");
analyticEvent.setTrigger(trigger);
try { ContextAnalyticsConnector.log(this, analyticEvent); }
catch (GimbalException e) { e.printStackTrace(); }
```

9 Getting events when your application is not running

Your app can receive geofence events and content events even when your app is in the background by following the steps below. For instance, you could use this to put up notifications to the user when the events are received.

9.1 Quick Start

1. Create an Android service in your application
2. Add listener for content/events to gimbal connectors

9.2 Creating background service

1. Make sure that service is started as STICKY (override onStartCommand method to return Service.START_STICKY)
2. In Service.onCreate() method get instance for Gimbal connectors and add listeners for the desired events/content
3. Add an entry in *AndroidManifest.xml* for your background service
4. Create a Broadcast receiver that listens for ACTION_BOOT_COMPLETE intent
5. Start your service when receiver receives boot complete intent
6. Add entry in *AndroidManifest.xml* for your broadcast receiver

9.3 Launching your application with a content URL scheme

As described in section 7.2.1, a *ContentDescriptor* contains a URL. This is a very flexible way to define content on the Campaign Management system. What immediately comes to mind is that you can refer to some web location of the content you want to display to your user.

Another interesting mechanism is to use Android schemes so that the URL is used to launch your application in the foreground when the user clicks on it.

In order to do so, add an intent filter defining your scheme to the activity you would like to launch. The following configuration will allow you to launch *MyActivity* using the following URL on the device mallmart://'.

```
<activity
     android:label="@string/app_name"
```

```
            android:name="com.company.MyActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <actionandroid:name="android.intent.action.MAIN"/>
                <categoryandroid:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
            <intent-filter>
                <dataandroid:scheme="mallmart"/>
                <actionandroid:name="android.intent.action.VIEW"/>
                <categoryandroid:name="android.intent.category.DEFAULT"/>
                <categoryandroid:name="android.intent.category.BROWSABLE"/>
            </intent-filter>
        </activity>
```

In the Context Console, create a campaign and choose URL for the creative type like below (read the section titled “Edit Push Notification” in the Context Console User Guide to see how to do this).

10 Privacy Controls

User privacy settings are an important part of the SDK. All Connectors in the SDK implement the *ConnectorPermissionChangeNotifier* interface. Accordingly, client code can listen to changes in a user's privacy settings by adding a *ConnectorPermissionChangeListener* to any of the Connectors. The following sections describe how to listen for changes to the user's privacy control settings.

10.1 Adding a ConnectorPermissionChangeListener to a ConnectorPermissionChangeNotifier

The *ConnectorPermissionChangeNotifier* interface implements two interface methods: *addConnectorPermissionChangeListener* and *removeConnectorPermissionChangeListener*. These methods allow the developer to call code to register listeners against the ContextCoreConnector, the *ContextPlaceConnector,* and the *ContextInterestsConnector*. The following code fragment demonstrates how to add a *ConnectorPermissionChangeListener* to the *ContextCoreConnector.*

```
ConnectorPermissionChangeListener coreConnectorPermissionChangeListener =
        new ConnectorPermissionChangeListener () {
        @Override
        public void permissionChanged(Boolean enabled) {
                Log.i(TAG, "Core permission change ");
        }
};
contextCoreConnector.addConnectorPermissionChangeListener
                                                (corePermissionChangeListener);
```

Adding instances of *ConnectorPermissionChangeListener* to the *ContextPlaceConnector* and the *ContextInterestConnector* follows the same pattern.

10.2 Removing a ConnectorPermissionChangeListener from a ConnectorPermissionChangeNotifier

To stop listening to user's privacy settings changes, simply remove the listener from the Connector.

```
contextCoreConnector.removeConnectorPermissionChangeListener
                                (coreConnectorPermissionChangeListener);
```

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 ContextConnector java documentation.

12 Developer Tips and Tricks

12.1 Refreshing Organizations and Places

When changes are done to the Organization such as adding/updating/deleting a place, they are pushed to all the clients only during the night. If you want to see those changes on your application immediately turn ON/OFF Location Permission to refresh the places.