Gimbal™ SDK for Android Documentation

V 1.1

July 2012

Qualcomm Labs

Qualcomm Labs, Inc.
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Gimbal™ SDK for Android Documentation
V 1.1
July 2012

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

1.5 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 Qualcomm Labs 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
- Upload target images and content for image recognition in the application

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
- Enables image recognition for applications to overlay content on recognized image targets.

2.2 What constitutes the Gimbal™ SDK

The SDK contains the following components:

- Gimbal jars and libraries
    - Context-Core jar
    - Context-Places jar
    - Context-Interests jar
    - Context-IR jar
    - Context-libs Eclipse project that wraps the above libraries into a project your application project should depend on
    - Image recognition APK library (Eclipse project)
- 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
- Content overlay triggered by Image Recognition

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)
- *ContextImageRecognitionConnector* (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 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
Interests	• Context-Core .jar • Context-Interests.jar	• Retrieve user interests
Image Recognition	• Context-Core .jar • Context-IR.jar • Image-recognition-apklib (eclipse project)	• Retrieve image recognition targets • Launch the image recognition UI that identifies targets and overlay's content configured in the Gimbal Manager

2.4 Pre-requisites

- Our sample project ships as an Eclipse project, the Eclipse Indigo (3.7) release is required:
  - http://www.eclipse.org/downloads/

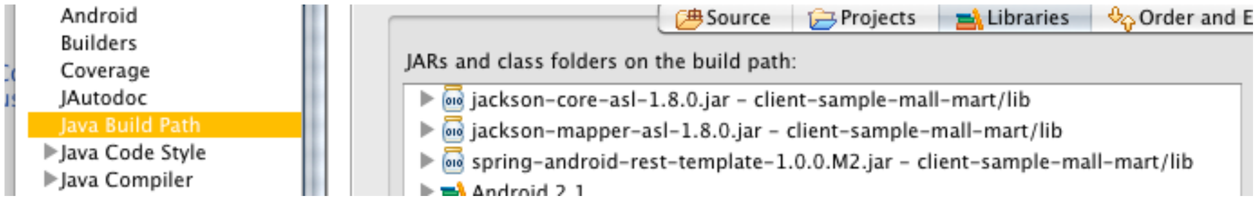
- You must use the ADT plug-in from Android located at the following URL:
  - http://developer.android.com/sdk/eclipse-adt.html

- 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:
  - jackson-core-asl-1.8.0.jar that you can download here: http://repository.codehaus.org/org/codehaus/jackson/jackson-core-asl/1.8.0/jackson-core-asl-1.8.0.jar
  - jackson-mapper.asl-1.8.0.jar that you can download here: http://repository.codehaus.org/org/codehaus/jackson/jackson-mapper-asl/1.8.0/jackson-mapper-asl-1.8.0.jar
  - spring-android-rest-template-1.0.0.M2.jar that you can download here: http://maven.springframework.org.s3.amazonaws.com/milestone/org/springframework/android/spring-android-rest-template/1.0.0.M2/spring-android-rest-template-1.0.0.M2.jar

- If you are using Image Recognition
  - qcar.jar and libQCAR.so for ARM and ARM v7a – two libraries (https://ar.qualcomm.at/qdevnet/sdk log in with the Vuforia user account, download the sdk and extract the jar and the so libraries). The supported QCAR libraries version is 1.5.9. You can raise a support request if 1.5.9 is no longer available on the above-mentioned url.

If you are using Eclipse, these libraries should be referenced as jar files or external jar files in the Java Build Path section of the project properties.

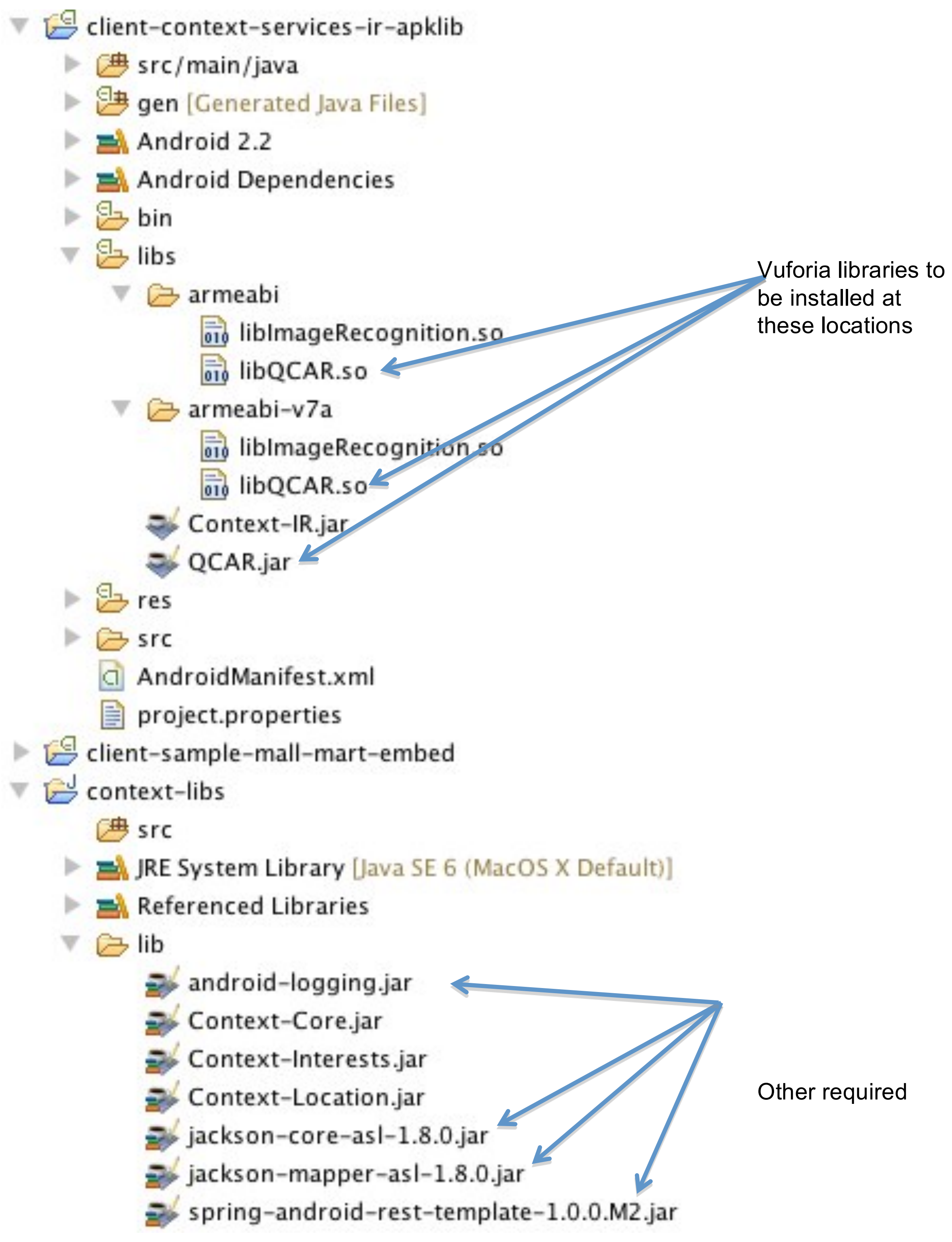


JARs and class folders on the build path:

jackson-core-asl-1.8.0.jar – client-sample-mall-mart/lib
jackson-mapper-asl-1.8.0.jar – client-sample-mall-mart/lib
spring-android-rest-template-1.0.0.M2.jar – client-sample-mall-mart/lib
Android 2.1

The API jar file(s) should also be referenced in this section of your project properties.

Your eclipse environment should look like the following:



```
▼ client-context-services-ir-apklib
    ▶ src/main/java
    ▶ gen [Generated Java Files]
    ▶ Android 2.2
    ▶ Android Dependencies
    ▶ bin
    ▼ libs
        ▼ armeabi
              libImageRecognition.so
              libQCAR.so
        ▼ armeabi-v7a
              libImageRecognition.so
              libQCAR.so
          Context-IR.jar
          QCAR.jar
    ▶ res
    ▶ src
      AndroidManifest.xml
      project.properties
▶ client-sample-mall-mart-embed
▼ context-libs
      src
    ▶ JRE System Library [Java SE 6 (MacOS X Default)]
    ▶ Referenced Libraries
    ▼ lib
          android-logging.jar
          Context-Core.jar
          Context-Interests.jar
          Context-Location.jar
          jackson-core-asl-1.8.0.jar
          jackson-mapper-asl-1.8.0.jar
          spring-android-rest-template-1.0.0.M2.jar
```

3 Setting up your application

3.1 Three steps set-up

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

1. Get an application key on Gimbal Manager
    a. For development purpose, you will just need to give your application package name
    b. When you go live, you will also need to get your certificate MD5 fingerprint
2. Set up some properties for your application
3. Use the right libraries for features you want to use

3.2 Get your application key

Obtain an application key (API Key) from the Gimbal Manager Apps Manager page in order to enable your app to work with Gimbal.

For development purposes, you can obtain it from: https://sandbox.gimbal.com.

For production[1] purposes (your final release), you obtain an application key from https://manager.gimbal.com.

When you use production (that is, when your application is final and will be pushed to the Android market place) you will need to provide Gimbal Manager with the fingerprint of the certificate you use to sign your application. This is described below.

When you are using Sandbox for development purposes, you don't need a fingerprint; instead, you can use your development certificate (android development certificate) to generate your APK.

Then log into the appropriate Gimbal Manager instance at the URLs listed above using the username and password for your organization. Choose the 'My Apps' 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

[1] Note: developers must adhere to the terms and conditions outlined in the "SDK License Agreement". Developers will need to view and click through this agreement upon their first login the developer portal at http://www.gimbal.com .

Fingerprint (for Production only)	The MD5 fingerprint of the certificate you used to sign your application

3.2.1 How to Obtain Your Application's MD5 Fingerprint

In order to obtain your application key, you need to provide an MD5 fingerprint of the certificate that you will use to sign your application. Before you visit the Gimbal Apps Manager page (https://manager.gimbal.com/apps), use the standard Java keytool program (from <JAVA_HOME>/jre6/bin/) to generate the fingerprint of your certificate.

First, determine which key you will use to sign your application at release and determine the path to the keystore that contains it.

Next, run Keytool with the -list option, against the target keystore and key alias. The table below lists the options you should use.

Keytool Option	Description
-list	Print an MD5 fingerprint of a certificate.
-keystore <keystore-path and name>.keystore	The name of the keystore containing the target key.
-storepass <password>	A password for the keystore. As a security precaution, do not include this option in your command line unless you are working at a secure computer. If not supplied, Keytool prompts you to enter the password. In this way, your password is not stored in your shell history.
-alias <alias_name>	The alias for the key for which to generate the MD5 certificate fingerprint.
-keypass <password>	The password for the key. As a security precaution, do not include this option in your command line unless you are working at a secure computer. If not supplied, Keytool prompts you to enter the password. In this way, your password is not stored in your shell history.

Here's an example of a Keytool command that generates an MD5 certificate fingerprint for the key `alias_name` in the keystore `my-release-key.keystore`:

```
$ keytool -list -alias alias_name -keystore my-release-key.keystore
```

Keytool will prompt you to enter passwords for the keystore and key. As output of the command, Keytool prints the fingerprint to the shell. For example:

```
Certificate fingerprint (MD5): 94:1E:43:49:87:73:BB:E6:A6:88:D7:20:F1:8E:B5:98
```

Once you have the fingerprint, you can begin the registration process at the Gimbal Manager (see the section entitled “Apps Manager” in the Gimbal Manager).

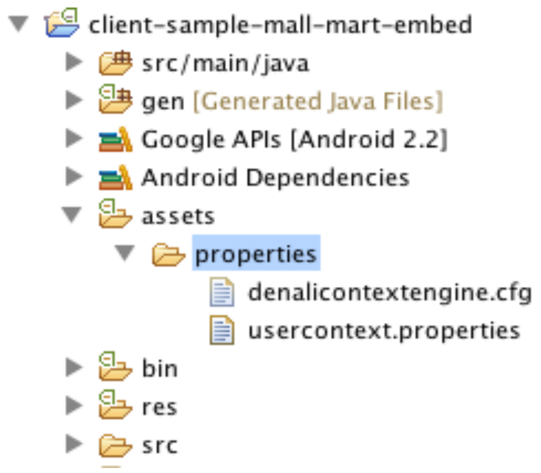
For more information, please read the Android documentation: http://developer.android.com/tools/publishing/app-signing.html.

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:



The following properties are supported:

Name	Description	Required
app.key	The application key that identifies your application	Yes
env	The server environment your application is using. Defaults to ‘SANDBOX’ for developer testing. ‘PROD’ for production.	No (Defaults to SANDBOX) Must be changed to PROD before submitting to the market
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)

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.UserContextService" >
                    <intent-filter>
                        <action android:name="<YOUR PACKAGE NAME
        HERE>.service.USER_CONTEXT_SERVICE" />
                    </intent-filter>
                </service>

                <receiver

        android:name="com.qualcommlabs.usercontext.service.UserContextServiceStartStopR
        eceiver"
                    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>

                <service

        android:name="com.qualcomm.denali.contextEngineService.DenaliContextEngineServi
        ce"
                    android:enabled="true" />
```

3.3.2.2 Optional Elements

The following elements must be configured if you are using the Image Recognition feature.

```
<activity
            android:name="com.qualcommlabs.usercontext.ir.ImageRecognition"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
            <intent-filter>
                <action android:name="<YOUR PACKAGE NAME HERE>.IR_ACTIVITY" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
</activity>

<service
android:name="com.qualcommlabs.context.ir.services.ImageRecognitionService" >
           <intent-filter>
               <action android:name="<YOUR PACKAGE NAME HERE>.IR_SERVICE" />
           </intent-filter>
</service>
```

3.3.2.3 Permissions

```
<uses-feature android:name="android.hardware.camera" />

<uses-permission android:name="android.permission.CAMERA" />
<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.BLUETOOTH" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

Note that the “camera” permission is required only if you use the Image Recognition Context-IR.jar.

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.

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 *checkStatus()* on this instance. This will call you back either *enabled()* or *disabled()*
3. If connector is 'disabled', then call *enable()*. 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 *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 *checkStatus()* method allows your application to ensure that it has been previously enabled.

4.3.1 Check the status

```
contextCoreConnector.checkStatus(new StatusCallback() {
            @Override
            public void enabled(ContextConnectorPermissions
contextConnectorPermissions) {
                // do something when the SDK is enabled
            }
           @Override
            public void disabled(int statusCode, String message) {
                // do something when the SDK is disabled
            }
});
```

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 Next steps

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

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 manged 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()*.
3. You can retrieve place events at any time with *requestLatestPlaceEvents()*.

5.2 Listening for geofence events

5.2.1 Obtain an instance

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

5.2.2 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.
placeId	This field is an internal field and may not mean anything to your application. It is important when updating existing places created locally on the phone rather than at the Context Console (and to help with debugging).
placeName	The name as assigned in the Context Console for organization places. The name as assigned when created (or updated) locally for personal places.
time	The time of the event in milliseconds since 1970 (see *System.currentTimeMillis()*).

5.2.3 Stop listening

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

5.3 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.

5.3.1 Creating a new private place

This function will create a new user-specific place. This could be called by a *MapActivity* that allows the user to select a location and a place radius. The radius should be set to 50 meters or greater. **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);

                Place place = new Place();
                place.setName("New user specific place");
                place.setGeoFence(circle);

                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.

5.3.2 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
                    }
                });
        }
```

5.3.3 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
                }
           });
    }
```

5.3.4 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
                }
           });
    }
```

5.3.5 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
                }
                });
        }
```

5.4 Retrieving latest geofence events

This call lets you retrieve all geofence events (public and private places) that occurred in the last 24 hours or what Gimbal was capable of keeping in its cache.

```
        contextPlaceConnector.requestLatestPlaceEvents(new Callback<
        List<PlaceEvent>>() {
                @Override
                public void success(List<PlaceEvent> placeEvent) {
                    // do something with place events
                }

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

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 *ContextInterestsConnectorFactor*.
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 Listening for User Interests

6.2.1 Retrieve the connector

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

6.2.2 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

7 Image Recognition

Gimbal's Image Recognition (IR) module allows developers to add image recognition capabilities to their applications. End users can point their device's camera at images to view and engage with interesting information related to the image in the camera view.

The IR capability is located in a supplementary project packaged as a zip file. The zip file contains all the resources needed to implement an IR experience in an application.

The SDK sample application provided by Qualcomm Labs contains a working example of the IR capability.

7.1 Quick Start

At a high level, the enabling of the IR experience in an application requires the following steps.

1. Enable the *ContextCoreConnector* (see section 4 above).
2. Optionally initiate a download of your organization's image recognition target datasets from the Gimbal Manager web application. For more information about target definition sets, refer to section titled "Image Recognition" of the "Gimbal Manager User Guide".
3. Once the target dataset has been successfully retrieved, show the IR camera viewfinder via the *ContextImageRecognitionConnector*.

7.2 Image Recognition application initialization

The IR SDK was designed to leverage functionality from the other Gimbal SDK components discussed in this document. Specifically, any application that leverages the IR SDK must use the same application initialization process outlined in section 4.

Once the *ContextCoreConnector* has been initialized and enabled, the *ContextImageRecognitionConnector* can be used to deliver the IR experience to the user. Typically, the initialization process is chained together as follows in the next several sections.

7.3 Image Recognition core concepts and code examples

7.3.1 Import the appropriate classes.

To use the ImageRecognition API of the SDK, include the appropriate imports. For IR, only packages related to the *ContextImageRecognitionConnector* and *ContextCoreConnector* are needed. The import statements below import the needed classes.

```
import com.qualcommlabs.usercontext.ContextCoreConnector;
import com.qualcommlabs.usercontext.ContextCoreConnectorFactory;
…
```

```
import com.qualcommlabs.context.ir.sdk.ContextImageRecognitionConnector;
import com.qualcommlabs.context.ir.sdk.ContextImageRecognitionConnectorFactory;
…
import com.qualcommlabs.usercontext.Callback;
Define instance variables to reference instances of ContextCoreConnector and
ContextImageRecognitionConnector.
```

As mentioned above, the *ContextCoreConnector* must be defined as an instance variable. The *ContextImageRecognitionConnector* is also defined as an instance variable. Typically, these objects are defined in a class that extends Android's Activity class.

```
private ContextCoreConnector contextCoreConnector;
private ContextImageRecognitionConnector imageRecognitionConnector;
```

In the activity's onCreate() method, these class members are initialized with the use of factory objects.

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

The *ContextImageRecognitionConnector* requires the *ContextCoreConnector* to be initialized and enabled before the IR functionality can be invoked. The following code provides the proper initialization.

```
protected void enableContextConnector() {
contextCoreConnector.enable(this, new Callback<Void>() {
@Override
public void success(Void responseObject) {
contextConnectorEnabled();
}

@Override
public void failure(int statusCode, String errorMessage) {
mallMartPresenter.onContextConnectorEnableFailure(errorMessage);
}
});
}
```

If successful, the *ContextCoreConnector* will call the *success()* method on Callback object. This callback method initiates the download of the applications IR data files.

```
private void contextConnectorEnabled() {
. . .
retrieveImageRecognitionTargets();
```

```
        . . .
        }

        private void retrieveImageRecognitionTargets() {
        imageRecognitionConnector.retrieveTargetBundle(MallMartActivity.this, new
        Callback<Void>() {

        @Override
        public void success(Void responseObject) {
        showImageRecognitionUI();
        Log.d(TAG, "Successfully retrieved targets");
        }

        @Override
        public void failure(int statusCode, String errorMessage) {
        Log.e(TAG, errorMessage);
        }
        });
        }
```

Once the target dataset is retrieved from the server, the Image Recognition UI can be displayed to the user.

```
            private void showImageRecognitionUI() {
                contextImageRecognitionConnector.showImageRecognitionUI(this, false,
        new Callback<String>() {

                    @Override
                    public void success(String message) {
                        Log.i(this.getClass().getName(), String.format("IR Successfully
        launched: %s", message.toString()));
                    }

                    @Override
                    public void failure(int statusCode, String errorMessage) {
                        throw new RuntimeException(errorMessage);
                    }

                });
            }
```

8 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.

8.1 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()*.

8.2 Listening for content

8.2.1 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
iconUrl	String	The icon URL for 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

8.2.2 Stop listening

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

8.3 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.

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. Derive your background service from *UserContextService*.
2. Replace the entry in *AndroidManifest.xml* for *UserContextService* with your background service.

9.2 Extend the UserContextService

Extend the *com.qualcommlabs.usercontext.service.UserContextS*ervice with your own service that implements the appropriate listener for the events you are interested in. In the event callback method implement the code to submit the appropriate system notification.

See the *com.company.CompanyService* in the sample application to view an example of how to do this.

9.3 Setting up your AndroidManifest.xml file with your service

Replace the *UserContextService* in the manifest with your custom service that extends *UserContextService*. Be sure to replace the highlighted text with your service name and package name respectively.

```
<service android:name="com.company.CompanyService">
        <intent-filter>
                <action
android:name="com.company.service.USER_CONTEXT_SERVICE" />
        </intent-filter>
</service>
```

9.4 Launching your application with a content URL scheme

As described in section 8.2.1, 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: *addPermissionChangeListener* and *removePermissionChangeListener*. 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 corePermissionChangeListener = new
 ConnectorPermissionChangeListener () {

     @Override
     public void permissionChanged(Boolean enabled) {
         Log.i(TAG, "Core permission change ");
     }

};

. . .
contextCoreConnector.addPermissionChangeListener(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.removePermissionChangeListener(corePermissionChangeListener);
```

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.