Google Awareness API Plugin

For the most recent documentation please always visit the online documentation!

Documentation: https://github.com/NinevaStudios/unity-google-awareness-api-docs/wiki
Support: info@ninevastudios.com
Facebook: https://www.facebook.com/nineva

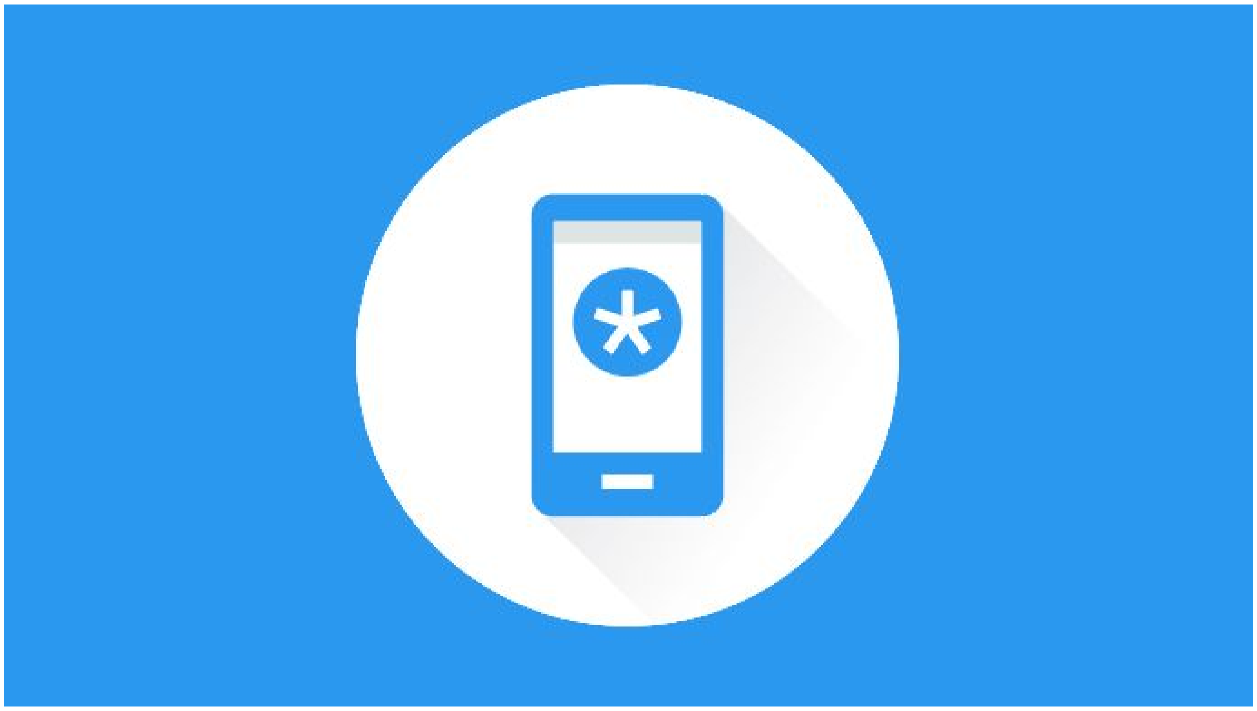
The Awareness API unifies 7 location and context signals in a single API, enabling you to create powerful context-based features with minimal impact on system resources. Combine optimally processed context signals in new ways that were not previously possible, while letting the API manage system resources so your app doesn't have to.

Table of contents

Plugin limitations

- Fences API receives callbacks ONLY when app is in foreground
- Pricing might be applicable to certain APIs (e.g. like https://developers.google.com/places/web-service/usage-and-billing) when free limits are exceeded

Snapshot API



You can use the Snapshot API to get information about the user's current environment.

Using the Snapshot API, you can access a variety of context signals:

- Detected user activity, such as walking or driving
- Nearby beacons that you have registered
- Headphone state (plugged in or not)
- Location, including latitude and longitude.
- Place where the user is currently located.
- Weather conditions in the user's current location.

Required permissions

Before using the API please make sure that you have the required permission declared in your `AndroidManifest.xml` and that user granted the permission at runtime.

Method	Required Android Permission
SnapshotClient.GetDetectedActivity()	com.google.android.gms.permission.ACTIVITY_RECOGNITION
SnapshotClient.GetBeaconState()	android.permission.ACCESS_FINE_LOCATION
SnapshotClient.GetHeadphoneState()	none
SnapshotClient.GetLocation()	android.permission.ACCESS_FINE_LOCATION
SnapshotClient.GetPlaces()	android.permission.ACCESS_FINE_LOCATION
SnapshotClient.GetWeather()	android.permission.ACCESS_FINE_LOCATION

Example XML with all possible required permissions and API keys (should be in `Plugins/Android`folder):

```xml
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.ninevastudios.awarenessdemo"
        xmlns:tools="http://schemas.android.com/tools"
        android:installLocation="preferExternal" android:versionName="1.0"
android:versionCode="1">
   <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="25"/>
   <supports-screens android:smallScreens="true" android:normalScreens="true"
android:largeScreens="true"
                    android:xlargeScreens="true" android:anyDensity="true"/>
```

```
    <application android:theme="@style/UnityThemeSelector"
android:icon="@drawable/app_icon"
                android:label="@string/app_name" android:debuggable="true">
        <activity android:name="com.unity3d.player.UnityPlayerActivity"
android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
            <meta-data android:name="unityplayer.UnityActivity" android:value="true"/>
        </activity>
        <!-- Put your key in the value! -->
        <meta-data
                android:name="com.google.android.geo.API_KEY"
                android:value="YOUR_API_KEY_HERE"/>
        <meta-data
                android:name="com.google.android.awareness.API_KEY"
                android:value="YOUR_API_KEY_HERE"/>
        <meta-data
                android:name="com.google.android.nearby.messages.API_KEY"
                android:value="YOUR_API_KEY_HERE" />
    </application>

    <!-- Weather snapshots depend on this permission -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <!-- Activity recognition requires its own permission -->
    <uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

    <uses-feature android:glEsVersion="0x00020000"/>
</manifest>
```

Detected user activity

You can get the user's current activity by calling `SnapshotClient.GetDetectedActivity()`, which returns an `ActivityRecognitionResult` containing information about the user's most recent current activities.

The `SnapshotClient.GetDetectedActivity()` method requires the `com.google.android.gms.permission.ACTIVITY_RECOGNITION` permission. Add this permission to `AndroidManifest.xml`.

Example code:

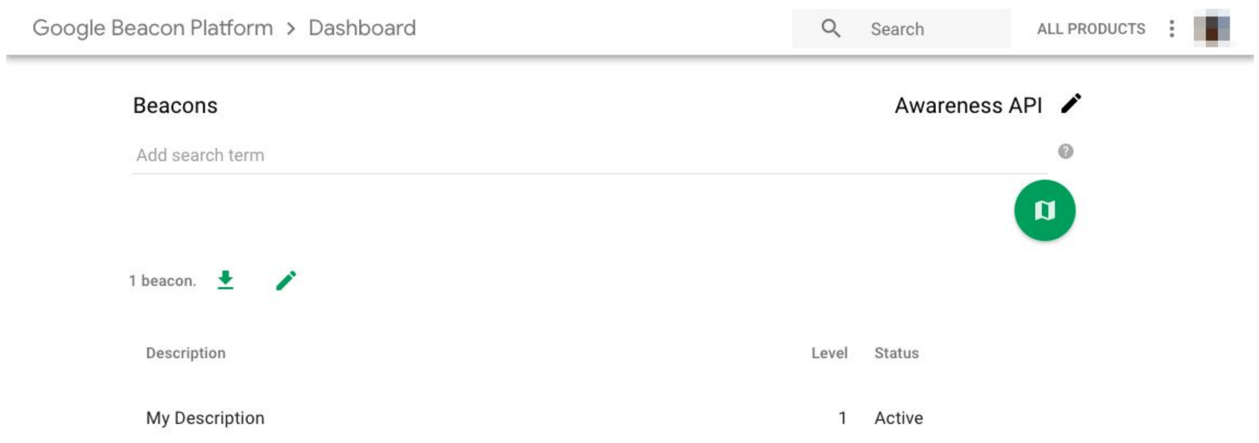
```
SnapshotClient.GetDetectedActivity(result =>
{
    Debug.Log("Still confidence: " +
result.GetActivityConfidence(DetectedActivity.ActivityType.Still));
    Debug.Log("Running confidence: " +
result.GetActivityConfidence(DetectedActivity.ActivityType.Running));
    LogSuccess(result);
}, err =>
{
    Debug.LogError(err);
});
```

Nearby beacons that you have registered.

Read the below info carefully! It doesn't query arbitrary beacons, you have to register them in your Google API Dashboard.

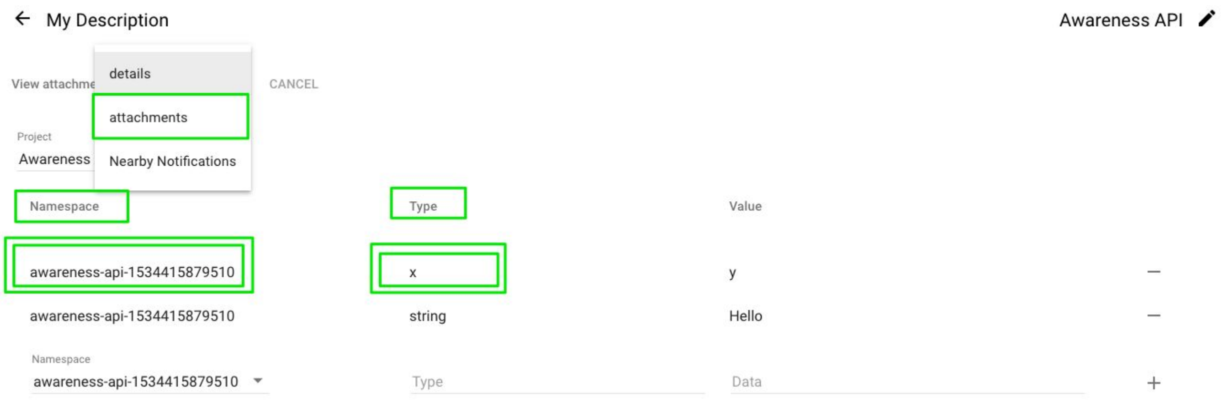
To use this API you first need to register your beacons with Google. To do so, please follow instructions on Google Developers in Use the Beacon Tools app section.

After you add the beacons with Beacon Tools Android app you should be able to see your beacons and manage them on your Beacon Dashboard



Description	Level	Status
My Description	1	Active

After that, go to Attachments and configure attachment so you can get it's namespace and type:



Namespace	Type	Value
awareness-api-1534415879510	x	y
awareness-api-1534415879510	string	Hello

Only after that you would be able to successfully query the beacons providing the namespace and type:

Example:

```
var theNamespace = "awareness-api-1534415879510";
var beaconTypes = new List<BeaconState.TypeFilter>
```

```
{
    BeaconState.TypeFilter.With(theNamespace, "string"),
    BeaconState.TypeFilter.With("com.google.nearby", "en"),
    BeaconState.TypeFilter.With(theNamespace, "x")
};
SnapshotClient.GetBeaconState(beaconTypes, state =>
{
    if (state.BeaconInfos.Count == 0)
    {
        Debug.Log("No beacons found");
    }
    else
    {
        Debug.Log(state);
    }
}, LogFailure);
```

Headphone state

You can find out if the headphones are connected or not.

Example code:

```
SnapshotClient.GetHeadphoneState(state => LogSuccess(state), LogFailure);
```

Location

Gets the device's current location (lat/lng).

```
SnapshotClient.GetLocation(location => LogSuccess(location), LogFailure);
```

Example response:

```
[Location: Latitude=49.8277755, Longitude=23.9916274, HasAccuracy=True,
Accuracy=16.903, Timestamp=1537187578720, HasSpeed=True, Speed=0,
HasBearing=False, Bearing=0, IsFromMockProvider=False]
```

Place where the user is currently located.

You can get the list of places with the user most probable location.

```
SnapshotClient.GetPlaces(places => places.ForEach(LogSuccess), LogFailure);
```

Example response:

```
Place: Id: ChIJ9dSnv37nOkcR7OZcMazVjis, Address: Heroiv UPA St, 77, L'viv, L'vivs'ka
oblast, Ukraine, 79000, Attrubutions: , Name: PrimeCode - реклама в інтернеті,
PhoneNumber: +380 68 614 4830, Locale: , PlaceTypes: PointOfInterest,Establishment,
PriceLevel: -1, Rating: 5, Location: lat/lng: (49.827879,23.9915659), Viewport:
[LatLngBounds SW: lat/lng: (0,0), NE: lat/lng: (0,0)], WebsiteUrl:
http://www.prime-code.net/
```

Weather conditions in the user's current location.

Gets the current weather conditions (temperature, feels-like temperature, dewpoint, humidity, etc.) at the current device location.

```
SnapshotClient.GetWeather(LogSuccess, LogFailure);
```

Example response:

```
[Conditions: Clear, Humidity: 52, DewPoint: 11.11111, Temperature: 21.11111, Feels
LikeTemperature: 21.11111]
```

Fences API

Limitations

Receiving callbacks when any fence is triggered is possible only if the app is running in the foreground!!!

Note: The Places and Weather context types are not supported for use with fences. Use the Snapshot API to get values for these types.

Overview

In the Awareness API, the concept of "fences" is taken from geofencing, in which a geographic region, or "geofence", is defined, and an app receives callbacks when a user enters or leaves this region. The Fence API expands on the concept of geofencing to include many other context conditions in addition to geographical proximity. An app receives callbacks whenever the context state transitions. For example, if your app defines a fence for headphones, it will get callbacks when the headphones are plugged in, and when they are unplugged.

Please read the official description of Fences API as an introduction

Using the Fence API, you can define fences based on context signals such as:

- The user's current location (lat/lng).
- The user's current activity (walking, driving, etc.).
- Device-specific conditions, such as whether the headphones are plugged in.
- Proximity to nearby beacons.

The Fence API lets you create fences by combining multiple context signals using `AND`, `OR`, and `NOT` boolean operators. Your app then receives callbacks whenever the fence conditions are met. Some examples of possible fences include:

User plugs in headphones and starts walking. User enters a 100-meter geofence before 5 p.m. on a weekday. User enters range of a specific BLE beacon. The following example shows defining a fence that activates whenever the user is walking:

```
AwarenessFence walkingFence =
DetectedActivityFence.During(DetectedActivityFence.ActivityType.Walking);
```

For examples see `FenceApiExamples.cs` file in `Example` folder.

Create a fence

```
static AwarenessFence CreateExercisingWithHeadphonesFence()
{
    // DetectedActivityFence will fire when it detects the user performing the specified
    // activity.  In this case it's walking.
    var walkingFence =
DetectedActivityFence.During(DetectedActivityFence.ActivityType.Walking);
```

```
    // There are lots of cases where it's handy for the device to know if headphones have
been
    // plugged in or unplugged.  For instance, if a music app detected your headphones fell
out
    // when you were in a library, it'd be pretty considerate of the app to pause itself before
    // the user got in trouble.
    var headphoneFence = HeadphoneFence.During(HeadphoneState.PluggedIn);

    // Combines multiple fences into a compound fence.  While the first two fences trigger
    // individually, this fence will only trigger its callback when all of its member fences
    // hit a true state.
    var notWalkingWithHeadphones =
AwarenessFence.And(AwarenessFence.Not(walkingFence), headphoneFence);

    // We can even nest compound fences.  Using both "and" and "or" compound fences, this
    // compound fence will determine when the user has headphones in and is engaging in at
least
    // one form of exercise.
    // The below breaks down to "(headphones plugged in) AND (walking OR running OR
bicycling)"
    var exercisingWithHeadphonesFence = AwarenessFence.And(
        headphoneFence,
        AwarenessFence.Or(
            walkingFence,
            DetectedActivityFence.During(DetectedActivityFence.ActivityType.Running),
            DetectedActivityFence.During(DetectedActivityFence.ActivityType.OnBicycle)));

    return exercisingWithHeadphonesFence;
}
```

Register a fence

```
FenceClient.UpdateFences(new FenceUpdateRequest.Builder()
    .AddFence(ExercisingWithHeadphonesKey, CreateExercisingWithHeadphonesFence())
    .AddFence(AllHeadphonesKey, CreateHeadphonesFence())
    .AddFence(AllLocationKey, CreateLocationFence())
    .AddFence(AllBeaconFence, CreateBeaconFence())
    .AddFence(AroundSunriseOrSunsetKey, CreateSunriseOrSunsetFence())
    .AddFence(AnyTimeIntervalKey, CreateAnyTimeIntervalFence())
    .AddFence(WholeDayKey, CreateWholeDayFence())
    .AddFence(NextHourKey, TimeFence.InInterval(currentTimeMillis, currentTimeMillis +
HourInMillis))
    .Build(), () => { LogSuccess("Fences successfully updated"); }, LogFailure);
```

Get fence callbacks

To receive the callbacks for the fences that you registered you need to subscribe to `FenceClient.OnFenceTriggered` event:

```
FenceClient.OnFenceTriggered += result =>
{
    text.text = result.ToString();
    Debug.Log(result);
};
```

Note that it will be triggered only if the app is in the foreground.

Query for fence state

You can also query the state of currently registered fences:

```
FenceClient.QueryFences(FenceQueryRequest.All(), response =>
{
    // This callback will be executed with all fences that are currently active
    var sb = new StringBuilder();
    sb.Append("Active fences: ");
    foreach (var fenceState in response.FenceStateDictionary)
    {
        sb.AppendFormat("{0} : {1}\n", fenceState.Key, fenceState.Value);
    }
    LogSuccess(sb);
}, LogFailure);
```