Skyhook Context Accelerator (Android)

Table of Contents

Skyhook Context Accelerator (Android)

Prerequisites

Supported on Android 2.2 (Froyo), 2.3.x (Gingerbread), 4.0.x (Ice Cream Sandwich), 4.1.x-4.3 (Jelly Bean), 4.4 (KitKat), 5.0-5.1 (Lollipop), 6.0 (Marshmallow) and 7.0-7.1 (Nougat) including forked platforms such as the Kindle Fire.

Installation

1. Open the zip file containing the Context Accelerator SDK.
2. The accelerator library files are found in the `lib` directory of the SDK.

    - If you are using Android Studio:

        1. Add all `*.jar` files to the `libs` directory (or any other of your choice) of your project
        2. Add depenencies to your `build.gradle`:

            ```
            dependencies {
                compile files('libs/accelerator.jar',
                              'libs/commons-codec-1.3.jar',
                              'libs/guava-r09.jar',
                              'libs/jsr305.jar',
                              'libs/s2-geometry-java.jar',
                              'libs/sqlite4java.jar')
            }
            ```
        3. Copy the following directores from the SDK to the `src/main/jniLibs` directory of your project:

            ```
            `lib/armeabi`
            `lib/armeabi-v7a`
            `lib/arm64-v8a`
            `lib/x86`
            `lib/x86_64`
            ```
    - If you are using Eclipse/ADT:

        - Add all of the contents of this directory to the `libs` directory of your project.
3. Add the accelerator service to your manifest.

    ```
    <application
        ...

        <service
            android:name="com.skyhookwireless.accelerator.AcceleratorService"
            android:exported="false" />
    </application>
    ```
4. Add the required permissions to your manifest.

    ```
    <!-- used to communicate with Skyhook's servers -->
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- used to initiate Wi-Fi scans -->
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    ```

```
    <!-- used to obtain information about the Wi-Fi environment -->
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <!-- used to obtain Wi-Fi or cellular based locations -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <!-- used to access GPS location for hybrid location functionality -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <!-- used to keep processor awake when receiving background updates -->
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <!-- used to check network connection type to optimize performance -->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Using the Android Emulator

The Context Accelerator SDK will not be able to determine location using Wi-Fi or cellular beacons from the emulator because it is unable to scan for those signals. Because of that, its functionality will be limited on the emulator. In order to verify your integration of the SDK using the emulator, you may want to use the `requestIPLocation()` method call. The full functionality will work only on an Android device.

Initializing

Import the SDK

Import the accelerator client and any other accelerator components where needed.

```
import com.skyhookwireless.accelerator.AcceleratorClient;
```

Instantiate the accelerator client

Connect to the accelerator service, and wait for the connection to complete successfully before calling any of the other accelerator methods.

```
public class YourActivity
    extends ...
    implements ...
               ConnectionCallbacks,
               OnConnectionFailedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        accelerator = new AcceleratorClient(this, yourKey, this, this);
        accelerator.connect();
    }

    @Override
    protected void onDestroy() {
        ...

        accelerator.disconnect();
    }

    @Override
    public void onConnected() {
        // It is now safe to call any of the Skyhook persona or IP
        // location accelerator methods. Before calling any of the
        // campaign monitoring methods, a pending intent first needs to
        // be registered for campaign monitoring, and before calling the
        // isMonitoringAllCampaigns() or getMonitoredCampaigns() methods,
        // that registration must be complete.

        handle successful connection...
    }

    @Override
    public void onDisconnected() {
        handle disconnection...
    }

    @Override
    public void onConnectionFailed(int errorCode) {
        handle connection failure...
    }

    private AcceleratorClient accelerator;
}
```

Take care of the location permissions

With the new permissions model introduced in Android M, the Context Accelerator requires location permission to be granted before calling most of its methods. Depending on how the Context Accelerator is used in the application, developer can decide when to request the permission and if an explanation needs to be displayed for the user.

Skyhook Personas

You can request a refresh of the Skyhook persona of the current user by calling `refreshPersona()`.

Once the `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service, you can refresh the persona. The `onPersonaDetermined()` method is called when the refresh persona operation completes successfully. Otherwise the `onPersonaError()` method will be called instead.

```
accelerator.refreshPersona(new OnPersonaResultListener() {

    @Override
    public void onPersonaDetermined(Persona persona) {
        process persona...
    }

    @Override
    public void onPersonaError(int statusCode) {
        handle failure...
    }
});
```

Latest Persona

You can access the latest persona at any time by calling the `fetchLastPersona()` method after `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service. Note that `fetchLastPersona()` uses the same `OnPersonaResultListener` interface as `refreshPersona()`. Persona will be passed to the `onPersonaDetermined()` method if it is available, otherwise `onPersonaError()` will be called.

```
accelerator.fetchLastPersona(new OnPersonaResultListener() {

    @Override
    public void onPersonaDetermined(Persona persona) {
        process persona...
    }

    @Override
    public void onPersonaError(int statusCode) {
        handle failure...
    }
});
```

Note that for apps using Google Play Services (v4.0 or later) the latest persona may be erased after a change in the Google Ad settings on the device. The `onPersonaError` method will be called with the `ERROR_PERSONA_NOT_SET` status code. You may want to request a persona refresh in that case.

The `Persona` object has the following demographic properties:

- age
- gender
- ethnicity
- income
- education

Each of these properties is a `List` composed of `Demographic` objects, listed in a decreasing order of probability.

For example, to list all the values for the current ethnicity persona:

```
for (Demographic demographic : persona.ethnicity) {
    Log.d(TAG, "value:"+demographic.value
               +" probability:"+demographic.probability
               +" variance:"+demographic.variance);
}
```

The result of the above code would print this:

```
value:white probability:0.65 variance: 0.1
value:other probability:0.15 variance: 0.1
value:black probability:0.12 variance: 0.1
value:asian probability:0.08 variance: 0.1
```

Note that since the values are ordered by probability, you can simply grab the first element (index 0) from the list to get the most probable value for the current user.

Behaviors

In addition to the five demographic objects, `Persona` also includes a `behaviors` field.

```
for (Behavior behavior : persona.behaviors) {
    Log.d(TAG, "id:"+behavior.id
               +" name:"+behavior.name);
}
```

The result of the above code would print something like this:

```
ID:12341318394918 name:auto intenders
ID:1234131839491234 name:auto enthusiasts
```

Campaign Monitoring

Before starting campaign monitoring you must register your pending intent so that you can be notified when the user enters or exits campaign venues when the app is in the background.

```
public class YourActivity
    extends ...
    implements ...
               ConnectionCallbacks,
               OnConnectionFailedListener,
               OnRegisterForCampaignMonitoringResultListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        accelerator = new AcceleratorClient(this, yourKey, this, this);
        accelerator.connect();
    }

    @Override
    public void onConnected() {
        ...

        Intent intent = new Intent(this, YourAcceleratorIntentService.class);
        PendingIntent pendingIntent =
            PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
        accelerator.registerForCampaignMonitoring(pendingIntent, this);
    }

    @Override
    public void onRegisterForCampaignMonitoringResult(int statusCode,
                                                      PendingIntent pendingIntent) {

        // The isMonitoringAllCampaigns and getMonitoredCampaigns methods
        // can also be called here if desired.
        accelerator.startMonitoringForAllCampaigns(new OnStartCampaignMonitoringResultListener()
            @Override
            public void onStartCampaignMonitoringResult(int statusCode, String campaignName) {
                // On success, calls to isMonitoringAllCampaigns or
                // getMonitoredCampaigns will now reflect that this campaign
                // was started.
            }
        });
    }
}
```

Note that `registerForCampaignMonitoring` can be called *without* establishing a connection to the accelerator service. This is useful in a special case when monitoring is resumed from a `RECEIVE_BOOT_COMPLETED` receiver (see below).

Once you start monitoring for campaign alerts, accelerator will send an alert using the provided pending intent when a campaign venue is entered or exited. An easy way to handle these alerts is to create an intent service to receive them.

```
public class YourAcceleratorIntentService
    extends IntentService {

    public YourAcceleratorIntentService() {
        super("YourAcceleratorIntentService");
    }

    protected void onHandleIntent(Intent intent) {

        if (AcceleratorClient.hasError(intent)) {
            int errorCode = AcceleratorClient.getErrorCode(intent);
            handle error...
        } else {
            CampaignVenue venue = AcceleratorClient.getTriggeringCampaignVenue(intent);
            if (venue != null) {
                if (AcceleratorClient.getCampaignVenueTransition(intent)
                        == CampaignVenue.CAMPAIGN_VENUE_TRANSITION_ENTER) {
                    process enter transition...
                } else {
                    process exit transition...
                }
            }
        }
    }
}
```

Add that intent service to your manifest.

```
<application
    ...

    <service
        android:name=".YourAcceleratorIntentService"
        android:exported="false" />
</application>
```

For most apps the `startMonitoringForAllCampaigns()` method is all that is needed for starting monitoring, but if more control is required, then your app can instead start monitoring for individual campaigns (assuming that the current class implements the `OnStartCampaignMonitoringResultListener` interface).

```
accelerator.startMonitoringForCampaign("YourCampaignName1", this);
accelerator.startMonitoringForCampaign("YourCampaignName2", this);
accelerator.startMonitoringForCampaign("YourCampaignName3", this);
```

When monitoring for individual campaigns, you can stop monitoring ones that are no longer needed (assuming that the current class implements the `OnStopCampaignMonitoringResultListener` interface).

```
accelerator.stopMonitoringForCampaign("YourCampaignName2", this);
```

Note that if all campaigns are being monitored, then calling `stopMonitoringForCampaign()` has no effect. It will not stop the specified campaign from being monitored.

Or you can stop monitoring for all campaigns, whether individual campaigns or all of them are being monitored (assuming that the current class implements the `OnStopCampaignMonitoringResultListener` interface).

```
accelerator.stopMonitoringForAllCampaigns(this);
```

Your app can fetch the list of recent campaign visits by calling the `fetchRecentCampaignVisits()` method once the `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service.

```
accelerator.fetchRecentCampaignVisits(100, new CampaignVisitsListener() {

    @Override
    public void onCampaignVisitsFetched(List<CampaignVisit> visits) {
        // handle recent campaign visits...
    }

    @Override
    public void onCampaignVisitsError(int errorCode) {
        // handle fetch recent campaign visits error...
    }
});
```

To resume campaign monitoring after a device reboot, add this to your manifest to listen for the BOOT_COMPLETED action.

```
<application
    ...

    <receiver android:name=".YourBootReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED">
            </action>
        </intent-filter>
    </receiver>
</application>
```

Add the appropriate permission to your manifest.

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Then when that action is received, resume campaign monitoring by calling `registerForCampaignMonitoring()`, but **do not call** `connect` from the `BroadcastReceiver`.

```
public class YourBootReceiver
    extends BroadcastReceiver
    implements ...
               ConnectionCallbacks,
               OnConnectionFailedListener,
               OnRegisterForCampaignMonitoringResultListener {

    @Override
    public void onReceive(Context context, Intent intent) {

        Intent myIntent = new Intent(context, YourAcceleratorIntentService.class);
        PendingIntent pendingIntent =
            PendingIntent.getService(context, 0, myIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        AcceleratorClient accelerator = new AcceleratorClient(context, yourKey, this, this);
        accelerator.registerForCampaignMonitoring(pendingIntent, this);
    }

    @Override
    public void onConnected() {
    }

    @Override
    public void onDisconnected() {
    }

    @Override
    public void onConnectionFailed(int errorCode) {
    }

    @Override
    public void onRegisterForCampaignMonitoringResult(int statusCode,
                                                      PendingIntent pendingIntent) {
    }
}
```

Note that a temporary instance of `AcceleratorClient` is created in the example above for the sole purpose of resuming monitoring. This instance will only exist in the context of the `BroadcastReceiver` and its `onReceive` method, so it should not be used for making any other accelerator calls.

IP Location

The Context Accelerator SDK can also be used to give you on-demand IP locations using the requesting remote IP once the `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service.

```
accelerator.requestIPLocation(new IPLocationListener() {

    @Override
    public void onIPLocationDetermined(Location location) {

        // Note that location will never be null
        double latitude = location.getLatitude();
        double longitude = location.getLongitude();
        long systemTime = location.getTime();

        // Note that extras will never be null
        Bundle extras = location.getExtras();

        // This extra is always present
        long elapsedRealtimeMillis = extras.getLong("elapsedRealtimeMillis");

        process latitude, longitude, systemTime, and elapsedRealtimeMillis...

        if (location.hasAccuracy()) {
            float accuracy = location.getAccuracy();
            process accuracy...
        }

        if (location.hasAltitude()) {
            double altitude = location.getAltitude();
            process altitude...
        }
        IPLocationType type = (IPLocationType) extras.get("type");
        if (type != null && type != IPLocationType.UNKNOWN) {
            process IP type...
        }

        StreetAddress streetAddress = (StreetAddress) extras.get("streetAddress");
        if (streetAddress != null) {
            String postalCode = streetAddress.postalCode;
            if (postalCode != null) {
                process postal code...
            }
        }
    }

    @Override
    public void onIPLocationError(int errorCode) {
        handle IP location failure...
    }
});
```

If successful, the `onIPLocationDetermined()` method will be called with an Android `Location` object which contains some extras that are specific to IP locations. All IP locations generated by accelerator are guaranteed to have a valid latitude, longitude, and timestamp (both UTC time and elapsed real-time in milliseconds, not nanoseconds, since boot). All other properties are optional.

The IP location properties are as follows:

IPLocation

Required properties:

Method Name	Property Type	Definition
getLatitude()	double	latitude, in degrees
getLongitude()	double	longitude, in degrees
getTime()	long	UTC time of this fix, in milliseconds since January 1, 1970

Required extras:

Extra Name	Property Type	Definition
elapsedRealtimeMillis	long	time of this fix in milliseconds, in elapsed real-time since system boot

Optional properties:

Method Name	Property Type	Definition
getAccuracy()	float	horizontal accuracy if available, in meters
getAltitude()	double	altitude if available, in meters above sea level

Optional extras:

Extra Name	Property Type	Definition
type	IPLocationType	type of IP address if available

streetAddress	StreetAddress	street address if available, currently only postalCode is populated

IPLocationType

Value	Definiton
FIXED	fixed IP address
MOBILE	mobile IP address
UNKNOWN	unable to resolve type of IP

Venue Information

The Context Accelerator SDK provides a collection of methods for obtaining venue information related to your campaigns and location.

Nearby Monitored Venues

The `fetchNearbyMonitoredVenues` method allows the client to obtain the unique identifiers of nearby venues that are part of actively monitored campaigns. This method can be used in conjunction with the `fetchVenueInfo` method to obtain more detailed venue information. This method can be called once the `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service. The client should be registered for campaign monitoring prior to calling this method.

```
accelerator.fetchNearbyMonitoredVenues(100, new NearbyMonitoredVenuesListener() {

    @Override
    public void onNearbyMonitoredVenuesFetched(List<NearbyCampaignVenue> venues) {
        // handle nearby venues...
    }

    @Override
    public void onNearbyMonitoredVenuesError(int errorCode) {
        // handle fetch venue info error...
    }
});
```

Venue Information by unique identifer

The `fetchVenueInfo` method allows the client to obtain more detailed venue information using the unqiue venue identifiers from the `CampaignVenue` and `NearbyCampaignVenue` objects. This method can be called once the `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service. The client should be registered for campaign monitoring prior to calling this method.

```
@Override
public void onNearbyMonitoredVenuesFetched(List<NearbyCampaignVenue> venues) {
    // Fetch venue information for nearby venues

    List<Long> ids = new ArrayList<Long>(venues.size());
    for (NearbyCampaignVenue venue : venues)
        ids.add(venue.venueId);

    accelerator.fetchVenueInfo(ids, new VenueInfoListener() {
        @Override
        public void onVenueInfoFetched(List<VenueInfo> venues) {
            // handle venue information...
        }

        @Override
        public void onVenueInfoError(int errorCode) {
            // handle fetch venue info error...
        }
    });
    ...
```

Venue Information at the current location

The `fetchVenueInfoAtLocation()` method allows the client to request the venue information at the current user location. This method can be called once the `AcceleratorClient` has been instantiated and `onConnected` has been called after connecting to the accelerator service.

```
accelerator.fetchVenueInfoAtLocation(new VenueInfoListener() {
    @Override
    public void onVenueInfoFetched(List<VenueInfo> venues) {
        // handle venue information...
    }

    @Override
    public void onVenueInfoError(int errorCode) {
        // handle fetch venue info error...
    }
});
```

Privacy Considerations

For apps using Google Play Services version 4.0 or later, the Context Accelerator SDK will collect certain usage information in order to improve the quality of Skyhook's positioning and context products. For a detailed overview of the data that the Skyhook Context Accelerator SDK will collect and use, please read our Privacy Policy, available at http://www.skyhookwireless.com/privacy-policy/skyhook. Please note that for apps not using Google Play Services 4.0 or later, user-based personas are not currently supported and the Accelerator SDK will only collect and provide anonymous location information.

To leverage user-based persona information, the Accelerator SDK needs an anonymous identifier. By default it will use Google Ad Id. You can override the identifier by assigning a custom identifier using Accelerator's setUserId method:

```
AcceleratorClient accelerator;
...
accelerator.setUserId("some-valid-and-unique-user-id");
```

In addition to Skyhook's default privacy protections, developers integrating the Context Accelerator SDK may also allow application users to opt out of usage data collection. Note that the application may receive degraded context information if data collection is disabled.

To disable usage data collection:

```
accelerator.setOptedIn(false);
```

You can check the data collection status anytime with:

```
accelerator.fetchOptedIn(new OnOptedInResultListener() {
    @Override
    void onOptedIn(boolean optedIn) {
        ...
    }
});
```

or re-enable it with:

```
accelerator.setOptedIn(true);
```

Note that by default Accelerator SDK follows Google Ads settings: data collection will be enabled only if Google Play Services are installed on the device, the application uses the Google Play Services library and the `Opt out of interest-based ads` option is unchecked in Google Settings.

If your application is not using the Google Play Services library but the application and/or content provider has received an EXPLICIT opt in from the user to leverage a unique device identifier for Geofence logging and personification, developers must set the user ID and set the opt-in to `true` to enable data collection:

```
accelerator.setUserId("some-valid-and-unique-user-id");
accelerator.setOptedIn(true);
```

If either of these conditions are not met, the default behavior of the SDK is to treat the user as opted out and disable data collection on the Server.

Online Documentation

For future updates and extra information please refer to the online version of the documentation here:

http://developer.skyhookwireless.com