HERE Android SDK

Developer's Guide

Starter Edition Version 3.11

Important Information
Notices

Topics:

This section contains document notices.

Legal Notices

Trademark Acknowledgements

Disclaimer

Document Information

Product	
Name:	HERE Android SDK
Version:	Starter Edition Version 3.11

Document	
Name:	HERE Android SDK Developer's Guide
ID:	0751ba2c-3800-40fc-9a31-7ccd4d344ea0
Status:	FINAL
Date:	2019-May-13, 13:07 (GMT)

Service Support

If you need assistance with this or any other HERE product, select one of the following options.

- If you have a HERE representative, contact them when you have questions/issues.
- If you manage your applications and accounts through *developer.here.com*, log into your account and check the pages on the SLA report or API Health. If this does not clarify the issue, then check *stackoverflow.com/questions/tagged/here-api*.
- If you have an evaluation plan, check *stackoverflow.com/questions/tagged/here-api*.
- If you have questions about billing or your account, *Contact Us*.
- If you have purchased your plan/product from a HERE reseller, contact your reseller.

Contents

Chapter 1
Overview

Topics:

The articles that follow introduce the HERE Android SDK, explain essential concepts and describe the common use cases it supports.

What is the HERE Android SDK?

The HERE Android SDK provides a set of programming interfaces that enable developers to build an immersive, geographically-aware Android applications by leveraging a powerful and flexible mapping platform. Through this SDK, developers can add rich location features such as routing, interactive maps, and global place search to their applications.

Feature List

The main features offered by the HERE Android SDK are listed below.

Note: The HERE Android SDK is designed for standalone Android APK development. If you plan to use the HERE SDK for platform-embedded app development (apps that ship with the device ROM), see *Service Support* on page 4 to contact us for more details .

Mapping:

- Dynamically download maps for more than 190 countries in over 60 languages
- Map styles: normal street map, satellite map, transit map, and more
- Touch gestures (including pan, flick, and pinch zoom)
- Overlay objects on the map such as polylines, polygons, icons, and routes
- Overlay custom raster tiles on the map (for example, to display heat maps)
- Ability to render raster tiles and map objects interleaved within different map layers

Search:

- Search through a broad set of geographical content across the globe (including streets, address points, and categorized places)
- Search for a specific place or explore by categories
- Access rich details for a Point of Interest from third-party content sources (including images, ratings, reviews, and editorials)
- Perform geocoding and reverse geocoding lookups

Directions:

- Online Car and Pedestrian Route Directions
- Specify preferred route type (fastest or shortest) and route options (such as avoiding toll roads, motorways, and parks)
- Alternate routes

Legal Requirements

In addition to the applicable *terms and conditions*, under which you have licensed the SDK, the following shall apply.

Components of the HERE SDK collect certain information from your application. Such information includes access credentials (`App_Id`, and `App_Code` – see also *Authenticating Applications* on page 20) and the types of features utilized by your application when used by end users. The information does not identify an individual end user. However, your application's privacy policy must disclose to the end users that you have licensed products and services from HERE, and that such information is collected from your application as it is being used by end users, and that HERE collects and processes such information from the application.

Chapter 2
Quick Start

Topics:

This section provides information to help you start using the HERE Android SDK.

Obtain the SDK

To obtain HERE Android SDK, follow these steps:

1. Visit *developer.here.com* and click on **Get Started for Free**.
2. Register or sign in to your HERE Account.
3. Agree to HERE Terms and Conditions.
4. On the **Platform Activation** screen find the section for **HERE Android SDK Starter Edition** and click on **Generate App ID and App Code** to obtain your app credentials and the SDK download link.
5. Follow the link to download HERE Android SDK as a Zip package.

After you have downloaded and extracted the Zip package, follow one of the next tutorials to begin using the SDK.

Run the Sample Application

This tutorial contains instructions on how to run the basic sample application to render a map on an Android device. This tutorial assumes that you are using the *Android Studio* development environment and a supported Android device. For more details, see *System Requirements* on page 20.

Development tasks for this basic application include:

- Open the sample project in Android Studio.
- Import the necessary resources into the project.
- Add HERE Credentials.

Open the Sample Project in Android Studio

The next task before running the sample HERE SDK project is to locate the project folder and open it in Android Studio as follows:

1. In the **Welcome to Android Studio** dialogue box select **Open an existing Android Studio project**.
2. In the **Open File or Project** dialogue box select the `BasicMapSolution` folder from your file system and click **OK**. The `BasicMapSolution` folder is located at `HERE-sdk.zip/HERE-sdk/tutorial/`.
3. After completing these steps the main Android Studio project window should appear with `"Error: Failed to resolve: :HERE-sdk:"` in the **Messages** pane.

Import the HERE SDK Android Archive

The HERE Android SDK library is shipped as an Android Archive (`.AAR`) file. You can import this library by doing the following:

1. On the **View** menu click **Tool Windows** > **Project**.
2. A few tabs are available in this tool window. Select the **Project** tab to show a file system view of the application structure.
3. Right-click on the `app` folder and select **New** > **Directory** to create a new folder. Use `libs` as the new folder name.
4. In your operating system's file system navigate to the extracted HERE SDK directory. Copy the `HERE-sdk.aar` file and paste it into the newly created `libs` directory.
5. Open the `build.gradle` file under the `app` folder and ensure the following entries are present:

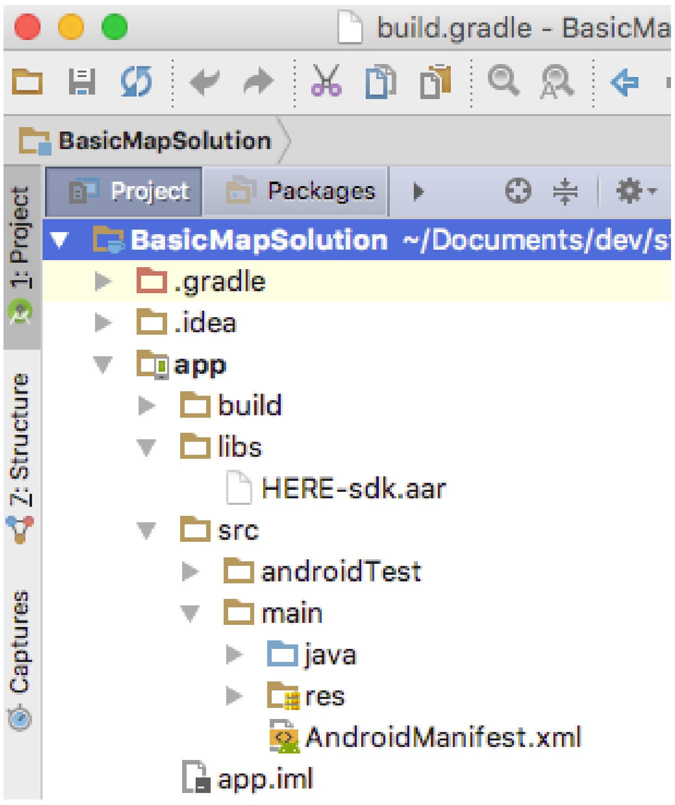
```
repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    implementation(name:'HERE-sdk', ext:'aar')
    // Depending on your specific project configuration you may have other entries here.
}
```

6. Optional: To enable quick Javadoc reference within your Android Studio environment, scroll down to the **External Libraries** section, right-click on `HERE-sdk`, and then select **Library Properties**. Click the + button and locate `HERE-sdk-javadoc.jar` from HERE SDK package.

Note: You can also import `HERE-sdk.aar` by using the menu, selecting **File** > **Project Structure...**, and clicking the "+" button. If you use this method, ensure that `HERE-sdk` is listed properly under the `app` Module Dependencies.

Figure 1: Location of the .AAR file



Add Credentials

This sample application is not shipped with a set of required HERE SDK credentials. Typically, before developing a new HERE SDK application, you need to acquire a set of credentials by registering your application on *http://developer.here.com*. Each application requires a unique set of credentials. When you register your app, the registered bundle identifier must match the package name in your project.

Since this sample app has the package name `com.here.android.tutorial` by default, you can generate a set of app credentials on *http://developer.here.com* using the same package name and add them to the app.

You can add credentials to your app by opening the `BasicMapSolution/app/src/main/AndroidManifest.xml` file and modifying the following `<meta-data>` tags:

- `<meta-data android:name="com.here.android.maps.appid" android:value="{YOUR_APP_ID}"/>`
- `<meta-data android:name="com.here.android.maps.apptoken" android:value="{YOUR_APP_CODE}"/>`

Run the Project

You can run your simple application by pressing the key combination **Shift + F10** (or **Ctrl + R** on macOS) from within Android Studio. The application renders a map retrieved from HERE servers. When you are running your application on a device, make sure data connection is enabled.

Note: For detailed instructions on how to create a new HERE SDK app, see *Create a Simple App Using HERE SDK* on page 13.

Create a Simple App Using HERE SDK

This tutorial provides instructions on how to create a simple application that uses HERE Android SDK to render a map on an Android device.

This tutorial assumes that you are using *Android Studio* development environment. Development tasks for this basic application include:

- Acquire HERE credentials for accessing map services
- Create a new Android Studio project
- Add necessary resources, permissions, and a map fragment to the project
- Modify `AndroidManifest.xml`
- Initialize the map fragment to create a map instance and associate this map with the map fragment for rendering on the client device

Acquire HERE SDK Credentials

Typically, before developing a new HERE SDK application, you need to acquire a set of credentials by registering your application on *http://developer.here.com*. Each application requires a unique set of credentials. When you register your app, the registered bundle identifier must match the package name in your project.

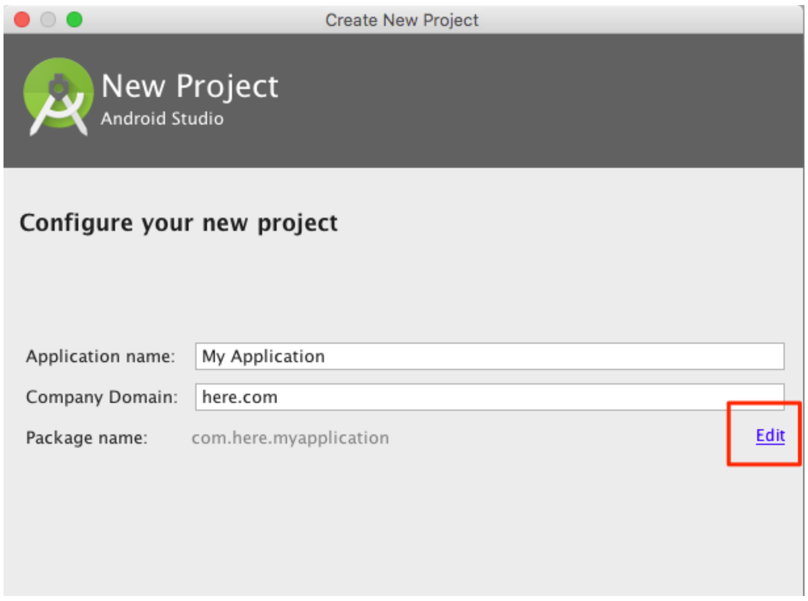
Create a New Android Studio Project

The second stage of developing an application using HERE SDK is to create a new project in Android Studio as follows:

1. From the Welcome to Android Studio dialogue box select **Start a new Android Studio project** to open the Create New Project dialog.
2. In the New Android Application dialog, under Application name, specify an appropriate application name. The remainder of this tutorial uses `BasicMapSolution` as the application name.
3. Under Company Domain specify an appropriate domain.

4. Edit the package name by clicking the **Edit** link. The remainder of this tutorial uses `com.here.android.tutorial` as the package name.

Figure 2: Edit the Package Name


Create New Project

New Project
Android Studio

Configure your new project

Application name: My Application
Company Domain: here.com
Package name: com.here.myapplication

5. Under Project Location specify an appropriate project location in the file system.
6. Click **Next**.
7. Select the form factors supported by your application. For the purpose of this tutorial, check Phone and Tablet.
8. Under Minimum SDK select the lowest version of the Android SDK you wish to support. For this sample application use Android 4.1.x "Jelly Bean".
9. Click **Next**.
10. You may be prompted to agree to a License Agreement. Click Accept and then **Next** to install SDK components. After the installation is complete, click **Next** again.
11. In the "Add an activity to Mobile" dialog box select Empty Activity and click **Next**.
12. In the "Configure Activity" dialog box specify an appropriate activity name in Activity Name. This tutorial uses the name `BasicMapActivity`.
13. Under Layout Name specify an appropriate layout name (this tutorial uses `activity_main`).
14. Click **Finish**.

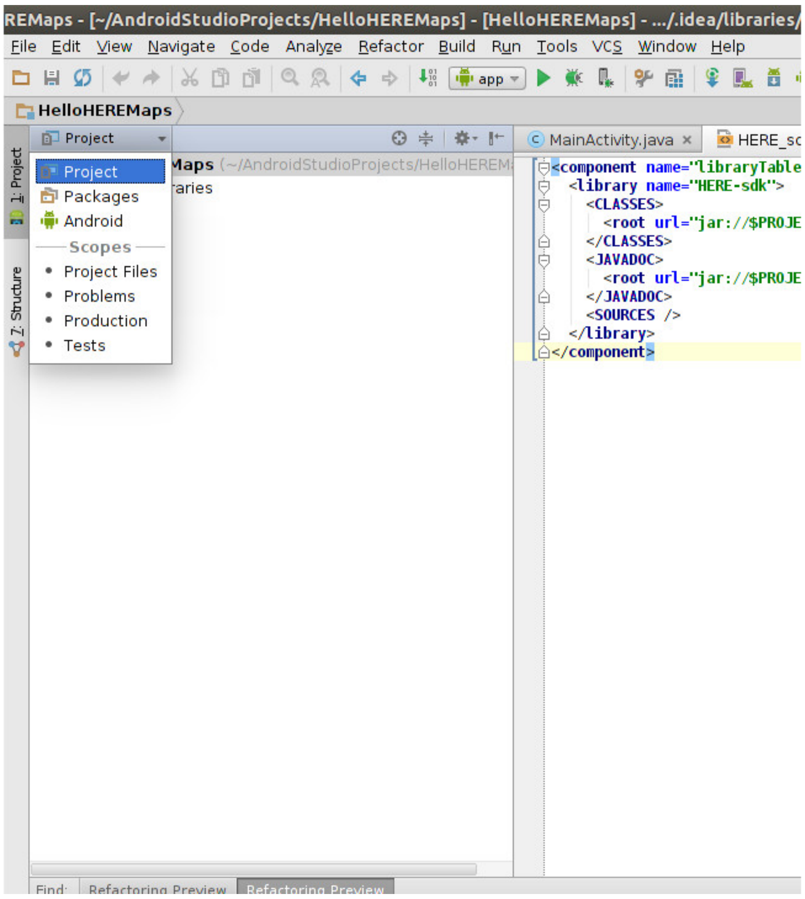
Result: Andriod Studio creates the structure for your project and opens the development environment.

A few views are available in the Android Studio development environment. The Android view shows a flattened view of the application structure, and the Project view shows a flattened view of the project structure including Gradle-related files.

The Android view provides quick access to key source files of your Android application. Selecting the `activity_main.xml` file in Android view opens the file in the Layout Editor and allows you to drag-and-drop widgets into your layout.

The following image shows how to switch between Android and Project view.

Figure 3: Switching Views in Android Studio


REMaps - [~/AndroidStudioProjects/HelloHEREMaps] - [HelloHEREMaps] - .../.idea/libraries/
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
HelloHEREMaps
Project
MainActivity.java
HERE_sc
Project
Packages
Android
Scopes
Project Files
Problems
Production
Tests
Maps (~/AndroidStudioProjects/HelloHEREM
raries
1: Project
7: Structure
Find:
Refactoring Preview
Refactoring Preview

```
<component name="libraryTable
  <library name="HERE-sdk">
    <CLASSES>
      <root url="jar://$PROJE
    </CLASSES>
    <JAVADOC>
      <root url="jar://$PROJE
    </JAVADOC>
    <SOURCES />
  </library>
</component>
```

Import HERE SDK Android Archive

The HERE Android SDK library is shipped as an Android Archive (.AAR) file. You can import this library by doing the following:

1. On the **View** menu click **Tool Windows** > **Project**.
2. A few tabs are available in this tool window. Select the **Project** tab to show a file system view of the application structure.
3. In your operating system's file system navigate to the extracted HERE SDK directory. Copy the `HERE-sdk.aar` file and paste it into the `app/libs/` directory under your application.
4. Optional: To enable quick Javadoc reference within your Android Studio environment, scroll down to the **External Libraries** section, right-click on `HERE-sdk`, and then select **Library Properties**. Click the + button and locate `HERE-sdk-javadoc.jar` from HERE SDK package.

Modify `build.gradle`

After importing the .AAR file modify `build.gradle` to add the file to your list of dependencies.

1. From the Project view pane locate the `build.gradle` file under the `app` folder and open it for editing.
2. In `build.gradle` add the following line into the `android { ... }` section:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

3. Next, add the following into the `dependencies { ... }` section:

   ```
   implementation(name:'HERE-sdk', ext:'aar')
   ```

4. Optional: Add the JTS Topology Suite (version 1.15.0 or later) library, which is used for rendering complex polygons.

   You can add this library by adding the following line into the `dependencies { ... }` section of the `build.gradle` file:

   ```
   implementation 'org.locationtech.jts:jts-core:1.15.0'
   ```

5. Optional: If you plan on extending this application with HERE Places or Routing functionality, add the GSON library to your project. You can add this library by adding the following line into the `dependencies { ... }` section:

   ```
   implementation 'com.google.code.gson:gson:2.8.0'
   ```

Modify AndroidManifest.xml and Add HERE Credentials

1. Add the HERE credentials to `AndroidManifest.xml`. For instructions on how to edit this file, see *Authenticating Applications* on page 20.
2. Modify the opening `<application>` by adding the `android:hardwareAccelerated="true"` attribute.

   ```
   <application android:icon="@drawable/icon"
   android:label="@string/app_name" android:hardwareAccelerated="true">
   ```

3. Add the following markup before the `<application></application>` tags:

   ```
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
   ```

   Note: If your app uses Android API level 23 (Android 6.0) or above, you must also add code to request for permissions during runtime. You can find more information in the *Request for Permissions* section.

4. Optional: To enable quick javadoc reference within your Android Studio environment, click on `.idea/libraries/HERE_sdk.xml` to edit it, and then add the following after </CLASSES> and before <SOURCES />:

   ```
   <JAVADOC>
    <root url="jar://$PROJECT_DIR$/app/libs/HERE-sdk-javadoc.jar!/" />
   </JAVADOC>
   ```

Result: Your project is able to make use of APIs from HERE SDK.

Edit activity_main.xml

Along with permissions and credentials you must add an Android `<fragment />` tag to set up the map fragment that your application activity is associated with. In this section we add a text label (generated as part of the default new application) and a map as follows:

1. From the Android View, under the `res/layout/` folder of your project, double-click the `activity_main.xml` file to open it for editing.
2. Ensure that the XML file has `<LinearLayout></LinearLayout>` as its root element. Depending on your version of Android Studio this may be a `RelativeLayout` instead. If that is the case, replace the contents of the file with the following:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World"
        tools:context=".BasicMapActivity" />

</LinearLayout>
```

3. Add the following markup beneath the `<TextView/>` tag:

```
<!-- Map Fragment embedded with the map object -->
<fragment
 class="com.here.android.mpa.mapping.SupportMapFragment"
 android:id="@+id/mapfragment"
 android:layout_width="match_parent"
 android:layout_height="match_parent"/>
```

Result: When `SupportMapFragment` is initialized, your application's `BasicMapActivity` contains a `SupportMapFragment` UI element (with the `mapfragment` ID) that owns a `Map` object.

Initializing the Map Fragment

When you have defined the basic layout of the application and acquired necessary permissions, the final step is to initialize the instance of the `SupportMapFragment` class, thus creating and associating a `Map` with the `SupportMapFragment` declared in the `activity_main.xml` file:

- From the Android View double-click the `BasicMapActivity.java` file under the `java` folder to open it for editing.
- Revise the `import` statements and functional logic of `BasicMapActivity` to look like the following:

```
package com.here.android.tutorial;

import android.app.Activity;
import android.os.Bundle;

import com.here.android.mpa.common.GeoCoordinate;
import com.here.android.mpa.common.OnEngineInitListener;
import com.here.android.mpa.mapping.Map;
import com.here.android.mpa.mapping.SupportMapFragment;

public class BasicMapActivity extends Activity {

 // map embedded in the map fragment
 private Map map = null;

 // map fragment embedded in this activity
 private SupportMapFragment mapFragment = null;

 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  initialize();
 }

 private void initialize() {
  setContentView(R.layout.activity_main);

  // Search for the map fragment to finish setup by calling init().
  mapFragment = (SupportMapFragment) getFragmentManager().findFragmentById(R.id.mapfragment);

  mapFragment.init(new OnEngineInitListener() {
   @Override
   public void onEngineInitializationCompleted(OnEngineInitListener.Error error) {
    if (error == OnEngineInitListener.Error.NONE) {
     // retrieve a reference of the map from the map fragment
     map = mapFragment.getMap();
     // Set the map center to the Vancouver region (no animation)
     map.setCenter(new GeoCoordinate(49.196261, -123.004773, 0.0),
       Map.Animation.NONE);
     // Set the zoom level to the average between min and max
     map.setZoomLevel((map.getMaxZoomLevel() + map.getMinZoomLevel()) / 2);
    } else {
     System.out.println("ERROR: Cannot initialize Map Fragment");
    }
   }
  });
 }
}
```

Request for Permissions

If your app supports Android 6.0 or above, it needs to ask users to grant certain permissions at runtime. For more information about this requirement, see *Requesting Android Permissions* on page 53.

Run the Applciation

You can run your simple application by pressing the key combination **Shift + F10** (or **Ctrl + R** on Macs) from within Android Studio. The application renders a map retrieved from the HERE servers. When you are running your application on a device, make sure data connection is enabled.

See the `BasicMapSolution` folder for a complete example. You need to add your own `App_Id` and `App_Code` for this example to work.

Chapter 3
User Guide

Topics:

The articles in this section provide a guide to using HERE Android SDK.

System Requirements

HERE Android SDK is designed and tested with Android phones and tablets in mind. SDK performance will vary between devices, since it is primarily determined by CPU, GPU, and display resolution. Currently, Nexus 5 is a suitable reference for a device which delivers acceptable SDK performance. If your target device is not a phone or tablet, see *Service Support* on page 4 to contact us for more details .

- Minimum supported API Level is 16 which corresponds Android 4.1.x "Jelly Bean". (`android:minSdkVersion`)
- Apps should be developed using Android Studio 2.3.2 or above
- 32MB of memory (RAM) available for foreground applications
- A minimum of 10MB per application should be made available for storing HERE SDK libraries
- A minimum of 32MB should be made available for storing map data
- Data connectivity (Wi-Fi or Cellular) is required to download map data

Note: HERE Android SDK does not support x86 Android and x86_64 devices. x86 and x86_64 emulators are supported. It is recommended that app testing is performed on ARM and ARM64 devices.

Authenticating Applications

Developers using HERE SDK with their app are required to register for a set of HERE credentials and to specify these credentials (`App_Id` and `App_Code`) in their app's Android manifest XML file. Failure to do so results in blocked access to certain features and degradation in the quality of other services.

To obtain these credentials visit the developer portal at *https://developer.here.com/plans* and register for a license. Once your project is created, you can generate these credentials on your Project Details page. If you already have a plan, you can also retrieve these credentials from your Project Details page.

Note: Credentials are unique to your account and your application's package namespace. Avoid reusing credentials across multiple applications.

Adding Credentials to the Manifest

You can add your HERE credentials as `<meta-data/>` attributes to `AndroidManifest.xml` file as follows:

1. In your development environment double-click your project's `AndroidManifest.xml` file and ensure that you are viewing the file in text editor mode.
2. Within the `<application></application>` block of tags add the following markup:

```
<meta-data android:name="com.here.android.maps.appid" android:value="YOUR_APP_ID"/>
<meta-data android:name="com.here.android.maps.apptoken" android:value="YOUR_APP_CODE"/>
```

3. Replace the `{YOUR_APP_ID}` and `{YOUR_APP_CODE}` strings with appropriate credentials for your application.

Examples on GitHub

You can find more HERE SDK sample projects on GitHub: *https://www.github.com/heremaps*

Maps

The core feature of HERE Android SDK is Maps. The key concepts covered in this section include adding a map to an Android application, changing the location displayed by the map and its various properties. The classes covered include `SupportMapFragment` and `Map`. `SupportMapFragment` and `Map` are parts of a Model-View-Controller (MVC) pattern where the Model is the `Map`, and the View is the `SupportMapFragment`. The `SupportMapFragment` is a standard Android Fragment derived component. You can create a controller class to coordinate all interactions using custom logic.

The first step to integrate a map into an application is to insert a `SupportMapFragment` to the view layout of the application. This is accomplished by adding `com.here.android.mpa.mapping.SupportMapFragment` to the *Android XML* layout file as follows.

```
<!-- Example fragment. This can be integrated and annotated
   like any other android Fragment or View widget -->
<fragment
 class="com.here.android.mpa.mapping.SupportMapFragment"
 android:id="@+id/mapfragment"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"/>
```

The `SupportMapFragment` class handles all user interactions such as panning, tapping or pinching, as well as other standard HERE SDK touch gestures documented in *MapGestures*.

Initializing SupportMapFragment

After the `SupportMapFragment` is added to the layout, the fragment must be initialized. The `SupportMapFragment` initialization is processed asynchronously. During initialization the `MapEngine` is initialized to create an instance of Map that is associated with the `SupportMapFragment`. The `SupportMapFragment.init(OnEngineInitListener)` method takes in an `OnEngineInitListener` input parameter to signal the caller when initialization is completed and if it was successful. The `SupportMapFragment` also initializes a *MapEngine* instance and creates a `Map` object associated with the `SupportMapFragment`. The following code illustrates the basic initialization flow when an `Activity` is created.

```
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 // Search for the Map Fragment
 final SupportMapFragment mapFragment = (SupportMapFragment)
   getFragmentManager().findFragmentById(R.id.mapfragment);
 // initialize the Map Fragment and
```

```
  // retrieve the map that is associated to the fragment
  mapFragment.init(new OnEngineInitListener() {
   @Override
   public void onEngineInitializationCompleted(
     OnEngineInitListener.Error error) {
    if (error == OnEngineInitListener.Error.NONE) {
     // now the map is ready to be used
     Map map = mapFragment.getMap();
     // ...
    } else {
     System.out.println("ERROR: Cannot initialize SupportMapFragment");
    }
   }
  });
 }
```

Note: For performance reasons `com.here.android.mpa.mapping.SupportMapFragment` has set `Fragment.setRetainInstance(boolean)` to true, and therefore `onCreate(Bundle)` is not called again when `Activity` is re-created (for example, after a zoom-level change).

Working with Map

Once the `SupportMapFragment` is initialized, you get the `Map` associated with the `SupportMapFragment` through `SupportMapFragment.getMap()`. The `Map` class represents the virtual model of the world in a digital format. Key attributes of the `Map` include the center `GeoCoordinate` and zoom level. For example, the following code snippet illustrates how to use `Map.setCenter(GeoCoordinate, Map.Animation)` to render the `Map` at Vancouver, Canada.

```
// map fragment has been successfully initialized
...

// now the map is ready to be used
Map map = mapFragment.getMap();

// Set the map center to Vancouver, Canada.
map.setCenter(new GeoCoordinate(49.196261,
 -123.004773), Map.Animation.NONE);
...
```

In the preceding code:

- The `GeoCoordinate` for the map center is created by a call to the `new GeoCoordinate(double, double)` constructor.
- When setting the center of a map, there is an option either to animate the change or to suppress animation by passing the constant `Map.Animation.NONE` as the relevant parameter.

Map Center and Zoom

Here are examples of setting and getting `Map` attributes:

Map Center

The center of the `Map` is a `GeoCoordinate` location that your Map is focused on. You can move a `Map` by redefining its center `GeoCoordinate`:

```
// Move the map to London
map.setCenter(new GeoCoordinate(51.51,-0.11),
 Map.Animation.NONE);
```

```
// Get the current center of the Map
GeoCoordinate coordinate = map.getCenter();
```

Zoom Level

The size of geographical area displayed by `Map` can be controlled by changing the zoom level. The zoom level ranges from `getMinZoomLevel()` to `getMaxZoomLevel()`, with the minimum value displaying the entire world. The following code sets the zoom level to median.

```
// Get the maximum, minimum zoom level
double maxZoom = map.getMaxZoomLevel();
double minZoom = map.getMinZoomLevel();

// Set the zoom level to the median (10)
map.setZoomLevel((maxZoom + minZoom)/2);

// Get the zoom level back
double zoom = map.getZoomLevel();
```

Animations

The map supports three types of animations when changing attributes:

- `Map.Animation.NONE`
- `Map.Animation.LINEAR`

```

// Move to Vancouver using linear animation
map.setCenter(new GeoCoordinate(49.0,-123.0),
 Map.Animation.LINEAR);
```

Note: If the map changes size or the app comes to the foreground while `Map.Animation.LINEAR` is being used in a `Map` attribute setter method, then the animation aborts, and the transition appears to fail. To avoid this behavior, use the `Map.Animation.NONE` animation type or wait until the map is stable before performing the transition operation.

MapState and Listening for Map Transform Events

`MapState` is an object that is a composite of the zoom level and center map attributes. Your application can listen for updates to the `MapState` by using an `OnTransformListener`.

Map transform events are triggered by any operation that causes the `MapState` to change. These operations include user interaction (for example, map gestures) as well as programmatic calls to the `Map` (for example, `map.setCenter(GeoCoordinate, MapAnimation)`). The `onMapTransformStart()` method is called before `MapState` begins to change, while the `onMapTransformEnd(MapState)` method is called after the `MapState` returns to a steady value. Because of this there can be a significant amount of time between the two callbacks in cases such as animated map movement events and continuous user interaction.

The following code is an example of registering an `OnTransformListener` to listen for map transform events:

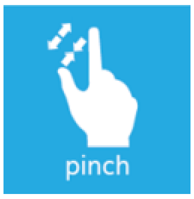
```
map.addTransformListener(new OnTransformListener() {
    @Override
    public void onMapTransformStart() {
        // map transform is about to start
    }
    @Override
    public void onMapTransformEnd(MapState mapState) {
        // map transform has come to an end
    }
});
```

If you need to update UI widgets as the `MapState` changes between these two callbacks, the recommended approach is to trigger a `Runnable` when `onMapTransformStart()` is called, which periodically checks (at no more than 30 frames per second) the current map state via `map.getMapState()` and updates the UI widgets accordingly. This `Runnable` can then be canceled upon a call to `onMapTransformEnd(MapState)`. An Android `Handler` can be used to trigger these `Runnable` objects.

Note: Do not update UI widgets in `MapRenderListener.onPostDraw(boolean, long)` as this method is frequently called.

Map Gestures

The `MapGesture` interface encapsulates all user interactions and touch gestures supported by HERE Android SDK. The `MapGesture` associated with a particular fragment can be retrieved from `SupportMapFragment.getMapGesture()`. The default behavior of the map for each gesture type may be used as-is, supplemented, or replaced entirely. The following table is a summary of the available gestures and their default behavior.

	To zoom the map in a fixed amount, tap the screen twice with one finger.
2 finger tap	**To zoom out a fixed amount**, tap the screen with two fingers.
	To move the map, press and hold one finger to the screen and move it in any direction.
	To pan the map with momentum, press and swipe one finger on the screen. The map continues to move in the same direction and gradually slows to a stop.
	**To zoom in or out continuously**, press and hold two fingers to the screen and increase or decrease the distance between them.

Tap the screen with one finger. This gesture does not have a predefined map action.

Press and hold one finger to the screen. This gesture does not have a predefined map action.

The OnGestureListener Interface

The `OnGestureListener` interface represents a listener to provide notification upon completion of a `Map` gesture event such as a single tap on a map.

For example, you can create a new `OnGestureListener()` as illustrated below.

```
// Map gesture listener
private class MyOnGestureListener implements OnGestureListener {

 @Override
 public void onPanStart() {
 }

 @Override
 public void onPanEnd() {
 }

 @Override
 public void onMultiFingerManipulationStart() {
 }

 @Override
 public void onMultiFingerManipulationEnd() {
 }

 @Override
 public boolean onMapObjectsSelected(List<ViewObject> objects) {
     return false;
 }

 @Override
 public boolean onTapEvent(PointF p) {
     return false;
 }

 @Override
 public boolean onDoubleTapEvent(PointF p) {
     return false;
 }

 @Override
 public void onPinchLocked() {
 }

 @Override
 public boolean onPinchZoomEvent(float scaleFactor, PointF p) {
     return false;
 }

 @Override
 public void onRotateLocked() {
 }

 @Override
 public boolean onRotateEvent(float rotateAngle) {
     return false;
 }

 @Override
 public boolean onTiltEvent(float angle) {
     return false;
 }

 @Override
 public boolean onLongPressEvent(PointF p) {
     return false;
 }

 @Override
 public void onLongPressRelease() {
 }

 @Override
 public boolean onTwoFingerTapEvent(PointF p) {
     return false;
 }
}
```

Note: The `OnGestureListener` methods that mention "rotate" and "tilt", such as `onRotateEvent(float)`, are not supported. They are only defined here to maintain compatibility with the Premium Edition of HERE SDK.

To add the listener to your map, include a call to `addOnGestureListener(OnGestureListener)` after the map fragment has been successfully initialized as follows:

```
...
mapFragment.init(new OnEngineInitListener() {
 @Override
 public void onEngineInitializationCompleted(OnEngineInitListener.Error error) {
  if (error == OnEngineInitListener.Error.NONE) {
   // map fragment has been successfully initialized
   mapFragment.getMapGesture().addOnGestureListener(new MyOnGestureListener());
  }
 }
});
...
```

Note: After you add an `OnGestureListener` to an application, remember to call `removeOnGestureListener(OnGestureListener)` when you no longer need to listen for map events to free up application resources.

The default implementation of a `OnGestureListener` does not affect any of the standard HERE SDK touch gestures. Each method within the `MyOnGestureListener` class returns a value of `false` which stipulates that the application should not override the underlying action a device performs upon detecting a particular gesture.

If you want to customize an action your application performs upon detection of a particular gesture, you must include appropriate commands within a relevant method of the `MyOnGestureListener` class and return a value of `true` to override the default action as illustrated below with revisions to `onTwoFingerTapEvent(PointF)` method.

```
@Override
public boolean onTwoFingerTapEvent(PointF p) {
 // Reset the map view
 double level = map.getMinZoomLevel() + map.getMaxZoomLevel() / 2;
 map.setCenter(new GeoCoordinate(49.196261, -123.004773),
  Map.Animation.NONE);
 map.setZoomLevel(level);
 return true;
}
```

Note: Since the `onTapEvent(PointF)` event is always triggered before the `onMapObjectsSelected(List<ViewObject>)` event, you can leverage this behavior to implement your own object selection logic. While implementing object selection it is recommended that you use both `Map.getSelectedObject(PointF)` and `Map.getSelectedObject(ViewRect)` and combine the results so that the user's tap input is interpreted over a larger area rather than only a single point.

After the revision the basic application responds to each two-finger tap gesture by returning to its initial view (the view displayed upon application launch). Other touch gestures continue to trigger standard HERE SDK actions.

Map Schemes

HERE Android SDK provides a variety of map skins for your application to choose from, these skins are otherwise known as *map schemes*.

`Map.Scheme` defines visualization types that the HERE map service supports. There is a variety of map schemes available that can be used based on the specific use case:

- Normal Day
- Terrain Day
- Satellite Day
- Hybrid Day
- Normal Day Grey

You can set a desired scheme by making a call to `Map.setMapScheme(String)` method.

Examples of Map Scheme

All available schemes are defined as constant strings in the `Map.Scheme` class. The following are examples of string values that you can use to set the map scheme in your application:

Figure 4: Scheme.NORMAL_DAY

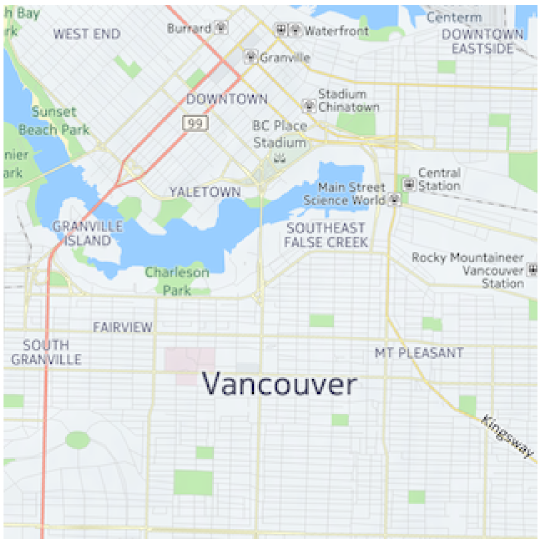


Figure 5: Scheme.SATELLITE_DAY

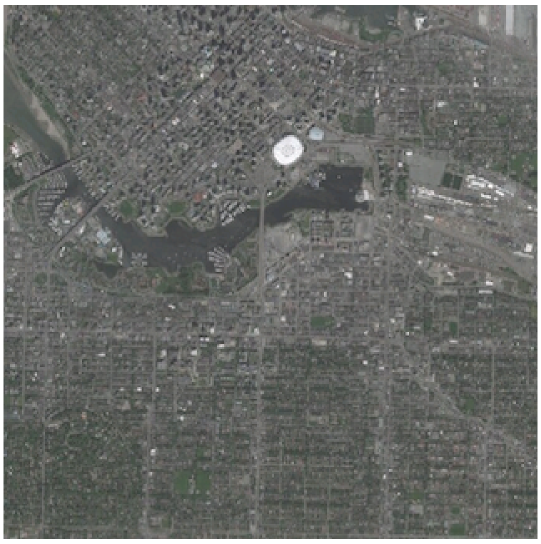


Figure 6: Scheme.HYBRID_DAY

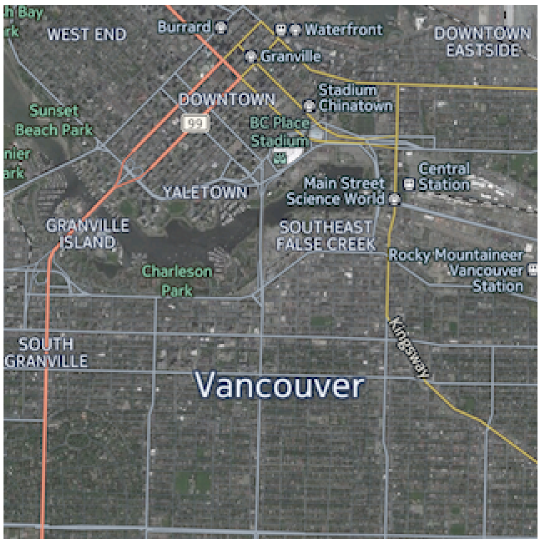
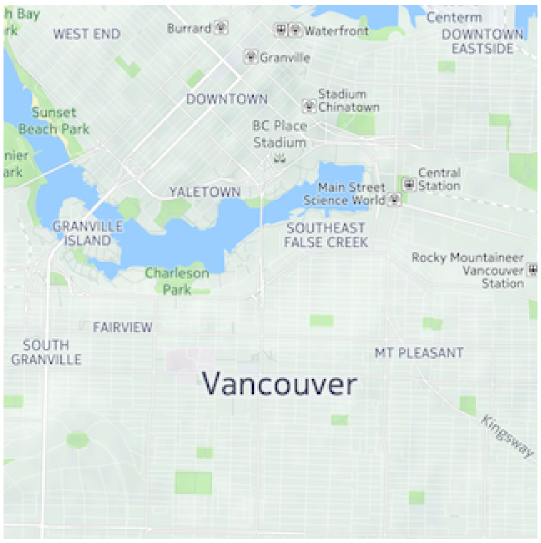


Figure 7: Scheme.TERRAIN_DAY



Setting a Map Scheme

The following example demonstrates how to retrieve available map schemes and change the current map scheme:

```
// Array containing string values of all available map schemes
List<String> schemes = map.getMapSchemes();
// Assume to select the 2nd map scheme in the available list
map.setMapScheme(schemes.get(1));
```

Listening to MapScheme Change Events

Applications can listen to map scheme change events via `Map.OnSchemeChangedListener`:

```
map.addSchemeChangedListener(new OnSchemeChangedListener() {
 @Override
 public void onMapSchemeChanged(String mapScheme) {
  // react to map scheme change here
 }
});
```

MapEngine Class

`MapEngine` is a singleton class used to manage active mapping resources used in applications developed with HERE SDK. `MapEngine` must be initialized before `Map` and map related objects such as `MapMarker` and `Places`, can be instantiated and retrieved from the API. A runtime exception occurs if `MapEngine` is not properly initialized before map related objects are used.

Initialization

`MapEngine` must be initialized before it can be used. It should be done in the main thread. `MapEngine` is automatically initialized for your application by using `SupportMapFragment`. *SupportMapFragment* is a fragment class applications can use as a UI module in an activity for map display. However, if your application does not use `SupportMapFragment` classes, then the application should initialize the `MapEngine` directly before using any HERE APIs. You can do this by calling `MapEngine.init(ApplicationContext, OnEngineInitListener)` as shown below:

```
MapEngine mapEngine = MapEngine.getInstance();
ApplicationContext appContext = new ApplicationContext(context);
mapEngine.init(appContext, new OnEngineInitListener() {
 @Override
 public void onEngineInitializationCompleted(Error error) {
  if (error == OnEngineInitListener.Error.NONE) {
   // Post initialization code goes here
  } else {
   // handle factory initialization failure
  }
 }
});
```

If map engine initialization is in progress or has failed, calling any other HERE SDK APIs fails because invalid objects cannot be created. To avoid this problem, check for `MapEngine.isInitialized()` in your app lifecycle callbacks. For example, the following example avoids problems with using the `PositionManager` before an instance can be properly created:

```
public void onDestroy()
{
 // Set initComplete using MapEngine.isInitialized()
 if (initComplete) {
  PositioningManager.getInstance().removeListener(this);
 }
 super.onDestroy();
}
```

For examples of typical scenarios using the SupportMapFragment that automatically initializes the MapEngine, see *Maps* on page 21.

Objects and Interaction

You can select ViewObject objects by using a single tap gesture. To enable this in your code, create an OnGestureListener object and pass it to SupportMapFragment.getMapGesture().addOnGestureListener(OnGestureListener). When a single tap occurs, the listener receives the onTapEvent(PointF) callback, and if that event is not handled, then the listener receives the onMapObjectsSelected(List<ViewObject>) callback. The application can then define what to do with the selected ViewObject.

Types of ViewObject objects that are selectable are defined within the ViewObject.Type enumeration includes the following:

- USER_OBJECT - an object the application adds to a map with a MapObject base class (MapPolygon for example).
- UNKNOWN_OBJECT - a selectable map object that is neither USER_OBJECT

The ViewObject Abstract Class

The ViewObject abstract class represents the base implementation for all objects selectable on a MapView or SupportMapFragment. The SupportMapFragment features user-selectable objects.

Sub-classes of the ViewObject class include MapObject .

MapObject and Geo Objects

MapObject represents an abstract class for all map related objects that can be added on a Map. The subclasses of this abstract class include the following:

- MapContainer
- MapCircle
- MapPolyline
- MapPolygon
- MapRoute
- MapMarker

These objects can be created by calling the appropriate constructor methods. In some cases a geo object is required in the constructor. Geo objects (for example, GeoPolyline and GeoPolygon) are geographical data representations that act as models to MapObjects, which act as views. Unlike map objects, geo objects cannot be added directly to a Map. For more information on geo objects and creating map objects see the API Reference.

The following code snippet demonstrates how to create a MapPolyline and a GeoPolyline object:

```
List<GeoCoordinate> testPoints = new ArrayList<GeoCoordinate>();
testPoints.add(new GeoCoordinate(49.163, -123.137766, 10));
testPoints.add(new GeoCoordinate(59.163, -123.137766, 10));
testPoints.add(new GeoCoordinate(60.163, -123.137766, 10));
GeoPolyline polyline = new GeoPolyline(testPoints);
MapPolyline mapPolyline = new MapPolyline(polyline);
```

To add a `MapObject` to the map, use `Map.addMapObject(MapObject)` or `Map.addMapObjects(List<MapObject>)`.

MapContainer

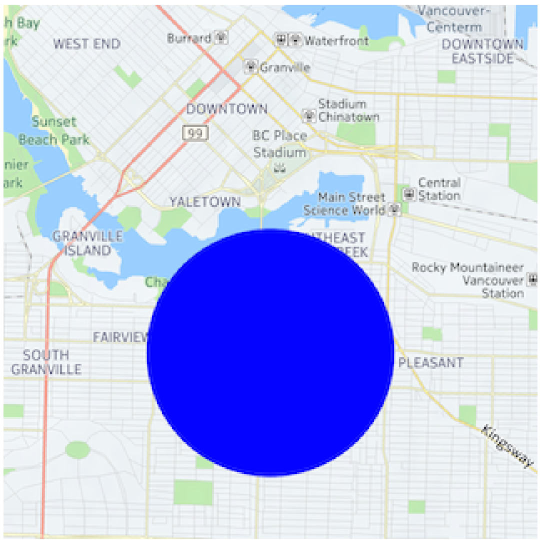
You can use `MapContainer` as a container for other `MapObject` instances. Map containers determine the stacking order of objects displayed on a map. To add a map object, call the `MapContainer.addMapObject(MapObject)` method.

Note: `MapRoute` and `MapContainer` cannot be added to a `MapContainer`.

MapCircle

A `MapCircle` represents a type of `MapObject` in the shape of a circle with an assigned radius distance and a `GeoCoordinate` center. It can be created by calling the constructor `MapCircle(double radius, GeoCoordinate center)`.

Figure 8: A MapCircle object

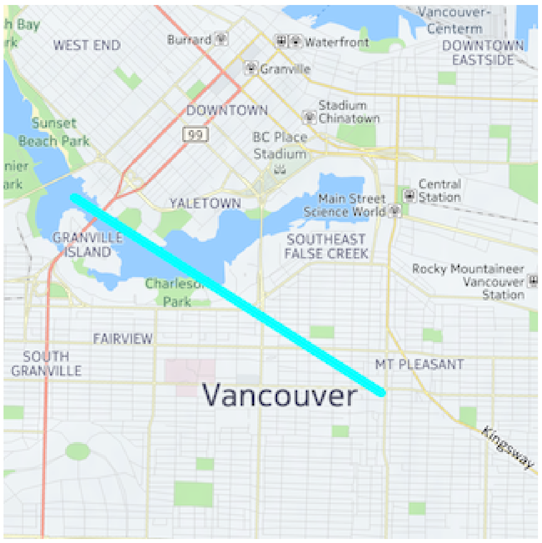


MapPolyline

A `MapPolyline` is a `MapObject` in the shape of a polyline with anchor points at any number of `GeoCoordinate` points. It can be created via a `GeoPolyline` object, which can be created by calling the `GeoPolyline(List<GeoCoordinate> points)` constructor.

Note: A `MapPolyline` or `MapPolygon` can only contain up to 65536 vertices.

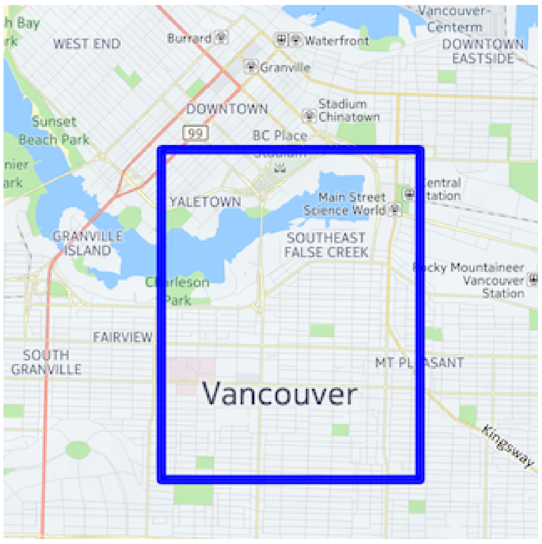
Figure 9: A MapPolyline object



MapPolygon

A `MapPolygon` is a `MapObject` in the shape of a polygon. In contrast to a `MapPolyline` it is assumed that the last coordinate in the line path is connected to the first coordinate thereby constructing an enclosed geometry. A `MapPolygon` may have separate border and fill colors. To create a MapPolygon, use the constructor `MapPolygon(GeoPolygon polygon)`. A `GeoPolygon` can be created by calling `GeoPolygon(List<GeoCoordinate> points)`.

Figure 10: A MapPolygon object



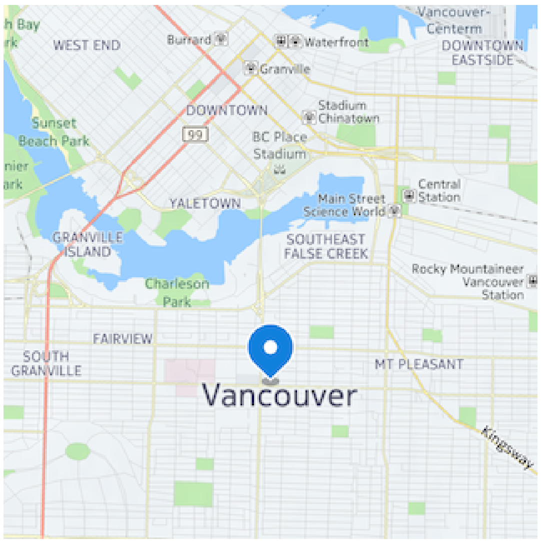
MapRoute

A `MapRoute` is a `MapObject` that displays a calculated route on a map. For more information on `MapRoute` see *Routing*.

MapMarker

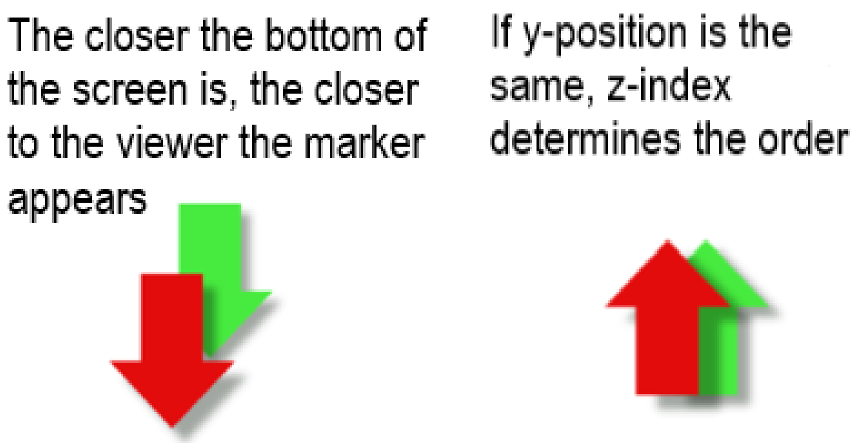
A `MapMarker` is a `MapObject` that displays an icon at a geographical position on a map. You can create a `MapMarker` with your own custom icon by calling `MapMarker(GeoCoordinate, Image)`.

Figure 11: A MapMarker object



`MapMarker` instances are always placed on top of other map objects. Refer to the diagram below for more information about z-index ordering for multiple map markers.

Figure 12: MapMarker order


The closer the bottom of the screen is, the closer to the viewer the marker appears
If y-position is the same, z-index determines the order

You can set `MapMarker` to be draggable by using the `MapMarker.setDraggable(true)` method. To listen to drag events, such as marker position changes, use `MapMarker.OnDragListener`.

User Interactions with MapObject

This section provides an example of handling `MapObject` tap events. In the following code:

- `addMapObject()` adds the object on the Map.
- `List<ViewObject>` holds the objects that have been selected in this tap event. By looping through this list of objects your code can find the `MapObject` that should respond to this tap event.

Note: The `onMapObjectsSelected(List)` callback is triggered after the `onTapEvent(PointF)` callback. For more information on this refer to *Map Gestures* on page 24.

```
// Create a custom marker image
com.here.android.mpa.common.Image myImage =
 new com.here.android.mpa.common.Image();

try {
 myImage.setImageResource(R.drawable.my_png);
} catch (IOException e) {
 finish();
}

// Create the MapMarker
MapMarker myMapMarker =
 new MapMarker(new GeoCoordinate(LAT, LNG), myImage);

map.addMapObject(myMapMarker);

...

// Create a gesture listener and add it to the SupportMapFragment
MapGesture.OnGestureListener listener =
 new MapGesture.OnGestureListener.OnGestureListenerAdapter() {
  @Override
  public boolean onMapObjectsSelected(List<ViewObject> objects) {
   for (ViewObject viewObj : objects) {
    if (viewObj.getBaseType() == ViewObject.Type.USER_OBJECT) {
     if (((MapObject)viewObj).getType() == MapObject.Type.MARKER) {
      // At this point we have the originally added
      // map marker, so we can do something with it
      // (like change the visibility, or more
      // marker-specific actions)
      ((MapObject)viewObj).setVisible(false);
     }
    }
   }
   // return false to allow the map to handle this callback also
   return false;
  }
  ...
 };
```

Custom Raster Tiles

You can use HERE Android SDK to enhance maps with custom raster tiles. Custom raster tiles are tile images that you can add to a map to customize it with enhanced information. For example, you may wish to use this feature to add heat maps over a map of New York City. You can store custom raster tile images locally or on a remote server for users to access when they navigate in a map. If the application is set to display custom raster tiles, then tiles are displayed when users view a designated geographical area at a specified zoom level or range of zoom levels.

Dividing a Map and Tile Lookup

To use your own custom raster tile images, you need to have a scheme for dividing your map according to the zoom level and map coordinates and then provide map tiles according to this scheme. Your application must then use this scheme in the implementation of one of the following classes:

- `MapRasterTileSource` - Implement this class if you plan to fetch local tile images, create dynamic images, or if you would like to provide your own method of retrieving images from a remote server.
- `UrlMapRasterTileSourceBase` - This is a convenience child class of `MapRasterTileSource`. Implement this if you plan to fetch tile images from a remote server using a URL over HTTP.

Note: Raster tiles can be one of the following supported image types:

- PNG
- JPEG
- BMP

Once a tile source has been implemented you can toggle its display by adding or removing it to the map using `Map.addRasterTileSource(MapRasterTileSource)` or `Map.removeRasterTileSource(MapRasterTileSource)`.

The MapRasterTileSource Abstract Class

`MapRasterTileSource` is the common way for you to define your raster tile source. If your application uses local tile images or remote images that require custom server authentication, then you should implement this class by defining `hasTile()` and `getTileWithError()` methods. For example:

```
public class MyTileSource extends MapRasterTileSource {

 @Override
 public boolean hasTile(int x, int y, int zoomLevel) {
  return true;
 }

 @Override
 public TileResult getTileWithError(int x, int y, int zoomLevel) {
  byte[] myImageData = null;
  // perform tile retrieval logic such as server authentication
  // also translate the x, y, and zoomlevel to address an image

  TileResult result = new TileResult(Error.NONE, myImageData);
  return result;
 }
}
```

Note: Ensure that `getTileWithError()` returns within a reasonable amount of time. If your operation takes a longer period of time, launch an asynchronous operation and return the `TileResult.Error.NOT_READY` error code while the operation is in progress.

The UrlMapRasterTileSourceBase Abstract Class

`UrlMapRasterTileSourceBase` is a child abstract class of `MapRasterTileSource` that you can use if you plan to fetch tile images from a remote server using image URLs. The following is a sample implementation of `UrlMapRasterTileSourceBase`. In this example, we use the `MapRasterTileSource.MapTileSystemHelper.tileXYToQuadKey()` method to address our map tiles. This helper method assumes that we are using a quadtree/quadkey scheme where the map is divided into a quadtree (a tree data structure where each node has exactly four children) with 20 levels. Each level of this map quadtree has $(2^x)^2$ tiles where *x* represents the floor function value of the current zoom level. So for level 0 there is 1 x 1 = 1 tile, level 1 has 2 x 2 = 4 tiles, level 2 has 4 x 4 = 16 tiles, and level 3.7 has 8 x 8 = 64 tiles—since the floor value of 3.7 is 3.

For more information about the quadkey/quadtree division scheme see the `tileXYToQuadKey()` API reference.

```
public class LiveMapRasterTileSource extends UrlMapRasterTileSourceBase {

 private final static String URL_FORMAT =
   "http://1.communitymaptiles.example.org/tilehub/live/map/png/%s";

 public LiveMapRasterTileSource() {
  // We don't want the tiles visible between these zoom levels
  hideAtZoomRange(12, 20);
  // Do not cache tiles
  setCachingEnabled(false);
 }

 // Implementation of UrlMapRasterTileSourceBase
 public String getUrl(int x, int y, int zoomLevel) {
  String url = null;

  // Utility to map the x, y coordinates easily into an equivalent
  // quadkey at a specified zoomLevel
  String quadKey =
    MapTileSystemHelper.tileXYToQuadKey(x, y, zoomLevel);

  try {
   // Append the quadkey to the URL template to get a real URL
   url = String.format(URL_FORMAT, quadKey);
  } catch (Exception ex) {
   ex.printStackTrace();
  }

  return url;
 }
}
```

The example above generates a quadkey from the x, y coordinates and the zoom level and appends it to the URL. However, this is server-specific and the method of converting x, y and zoom level to a URL can be done in many ways. Also, it is worth noting that tiles can be cached with `setCachingEnabled(true)`.

Caching Tiles

Tiles can be cached to the disk by calling the following:

```
// Give the tile source a custom prefix so it can be cached on the disk
MapRasterTileSource.setCachePrefix(String cache)

// Give each raster tile file an expiration time in seconds
MapRasterTileSource.setCacheExpiration( int seconds )
```

If no expiration time is set, then the raster tiles remain on the device. We recommend that both a cache prefix and an expiration time be set.

Positioning

Basic Positioning

HERE Android SDK provides the following interfaces for users to retrieve location updates and to display their current location on a map:

- `PositioningManager`
- `OnPositionChangedListener`
- `PositionIndicator`

Note: Android permission `android.permission.ACCESS_FINE_LOCATION` is required when your app calls `PositioningManager.start(LocationMethod)`. Otherwise, the method returns `false`. In addition, to ensure that the app receives location updates, the user needs to have the Location permission enabled (toggled to "on") during runtime.

PositioningManager Class

A `PositioningManager` class provides information related to the device geographical location like the current position and the average speed. Applications can register to receive position updates using one of the positioning mechanisms described in the `LocationMethod`:

- `GPS` - positioning using the real GPS available on the device
- `GPS_NETWORK` - positioning is provided using a wireless network or the real GPS available on the device
- `NETWORK` - positioning using a wireless network

The current status of a particular location method is represented by the `LocationStatus` value returned from the `PositioningManager.getLocationStatus(LocationMethod)` method.

`PositioningManager` can be accessed by calling `PositioningManager.getInstance()`. An application can start receiving real time positioning updates by calling `PositioningManager.start(LocationMethod)` with one of the location methods listed above and can stop positioning updates by calling `PositioningManager.stop()`. While position updates are being received, an application can retrieve the current position of the client device via `PositioningManager.getPosition()` method.

OnPositionChangedListener Interface

In addition to the `PositioningManager`'s `getPosition()` method applications can subscribe to position update notifications from the `PositioningManager` through the `PositioningManager.OnPositionChangedListener` interface. To add or remove `OnPositionChangedListener`, applications can use the following methods:

```
PositioningManager.addListener(WeakReference<OnPositionChangedListener>)
```

```
PositioningManager.removeListener(OnPositionChangedListener)
```

The positioning manager enhances your application with the current position of the user's device. Registration of the positioning listener should be performed after `SupportMapFragment`, `MapView`, or `MapEngine` is initialized as described in the following code snippet.

```
// Define positioning listener
private OnPositionChangedListener positionListener = new
```

```
 OnPositionChangedListener() {

  public void onPositionUpdated(LocationMethod method,
    GeoPosition position, boolean isMapMatched) {
   // set the center only when the app is in the foreground
   // to reduce CPU consumption
   if (!paused) {
    map.setCenter(position.getCoordinate(),
      Map.Animation.NONE);
   }
  }

  public void onPositionFixChanged(LocationMethod method,
    LocationStatus status) {
  }
};

// Register positioning listener
PositioningManager.getInstance().addListener(
 new WeakReference<OnPositionChangedListener>(positionListener));
...
```

In order to avoid unnecessary position updates while the activity is in the background, you need to start or stop the `PositioningManager` within your activity's `onResume()` and `onPause()` methods.

```
// Set this to PositioningManager.getInstance() upon Engine Initialization
private PositioningManager posManager;
...

// Resume positioning listener on wake up
public void onResume() {
 super.onResume();
 paused = false;
 if (posManager != null) {
  posManager.start(
    PositioningManager.LocationMethod.GPS_NETWORK);
 }
}

// To pause positioning listener
public void onPause() {
 if (posManager != null) {
  posManager.stop();
 }
 super.onPause();
 paused = true;
}

// To remove the positioning listener
public void onDestroy() {
 if (posManager != null) {
  // Cleanup
  posManager.removeListener(
    positionListener);
 }
 map = null;
 super.onDestroy();
}
```

PositionIndicator Class

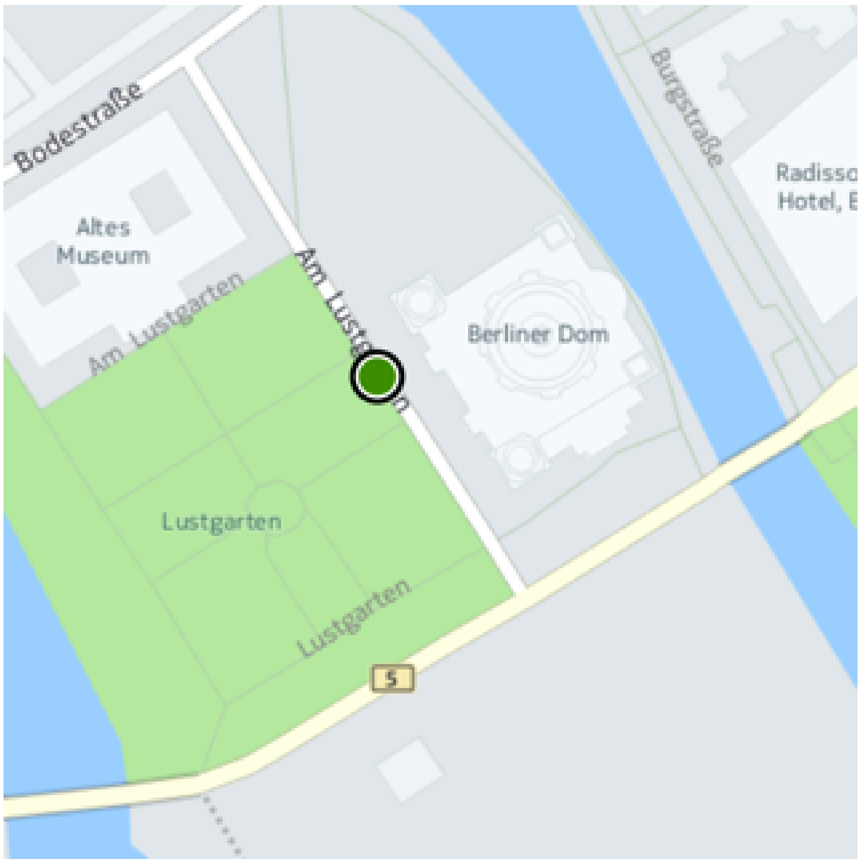
`PositionIndicator` is a special map marker object that allows the current client device position to be shown on a map. Every HERE SDK `Map` object has an integrated position indicator set to invisible by default. The indicator can be retrieved and set to visible by calling `Map.getPositionIndicator()` and `PositionIndicator.setVisible()` as follows:

```
// map fragment has been successfully initialized
// now the map is ready to be used
map = mapFragment.getMap();

// display position indicator
map.getPositionIndicator().setVisible(true);
```

By default the position indicator is rendered as a marker surrounded by a circle, the diameter of which illustrates the accuracy of the indicated position. You can change this marker by calling `PositionIndicator.setMarker(Image)`.

Figure 13: A PositionIndicator



Note: For the position indicator to stay in the center of the map and illustrate real-time updates of the device position, it is necessary to update the map center whenever a new location update is received.

Note: `PositionIndicator` only works if the application has started the `PositioningManager`.

Directions

This section provides an overview of the Directions feature in HERE SDK. The Directions feature allows developers to define and display routes between a start and a destination point within their application. It supports many navigation options such as toll road preference and transport type.

Route Calculation for Walking or Driving

HERE Android SDK supports route calculation with multiple waypoints optimized for walking or driving.

A route describes a path between at least two waypoints, the starting point and the destination, with optional intermediate waypoints in between. Applications can provide route information to users in two ways:

- A line rendered on a map that displays a connecting path between all waypoints
- Turn-by-turn directions in text format

Note:

- To use this feature, your application must include the **Gson** library (release 2.2.4 or a compatible version) on its class path.
- To support routes that are greater than 5000 km, add the `largeHeap` attribute in the `<Application>` section of `AndroidManifest.xml`. For more information see *this article* on managing your app memory.

Route Calculation Classes

This section introduces the following classes that are used for route calculations:

- `RouteManager`
- `RoutePlan`
- `RouteOptions`

The `RouteManager` class is responsible for calculating a route. An application can initiate a route calculation by calling `RouteManager.calculateRoute(RoutePlan, Listener)` method providing options and waypoints through `RoutePlan` and receive progress updates through the `RouteManager.Listener` instance.

`RoutePlan` is a waypoint container that is used for route calculation. A `RoutePlan` object is comprised of a list of waypoint objects and optional `RouteOptions`. If `RouteOptions` are not specified, default values are used. You can add an individual waypoint to a `RoutePlan` by using a `GeoCoordinate`.

The `RouteOptions` class is a model of parameters required to calculate one route. It encapsulates "building block" parameters for a route such as:

- The desired number of routes
- The direction of travel that the route should start in
- The routing type such as fastest travel time or shortest distance
- The mode of transportation, e.g. by walking or driving
- The departure time
- The allowed types of route segments such as dirt roads, highways, or ferries

Note: Routing type describes different optimizations that can be applied during the route calculation:

- `Fastest` - route calculation from start to destination optimized by travel time. In some cases the route returned by the fastest mode may not be the route with the shortest possible travel time. For example, it may favor a route that remains on a highway even if a shorter travel time can be achieved by taking a detour or shortcut through a side road.
- `Shortest` - route calculation from start to destination disregarding any speed information. In this mode the distance of the route is minimized while keeping the route sensible. This includes,

for example, penalizing turns. Because of that the resulting route will not necessarily be the one with minimal distance.
- `Balanced` - route calculation from start to destination optimized based on combination of travel time and distance.

HERE SDK supports alternate routes between two waypoints. The alternate route feature allows more than one route to be returned after a route calculation. You can use the `RouteOptions` class to set the desired number of routes, HERE SDK then returns different routes according to this limit. Note that the first element of the returned list of `RouteResult` is the main route, and the rest of the returned routes are not listed in any specific order.

RouteResult and Route

The `RouteResult` class represents a route calculation result. Applications can retrieve a `Route` object and the corresponding set of violated routing conditions. *Violated routing options* are the conditions that a routing result was unable to adhere to. For example, after specifying a route calculation that avoids tolls and ferries, you may get a `RouteResult` that contains a `Route` object along with `RouteResult.ViolatedOption.AVOID_TOLL_ROADS`. This indicates that although a route was found, this route goes through at least one toll road — violating a condition of your route request.

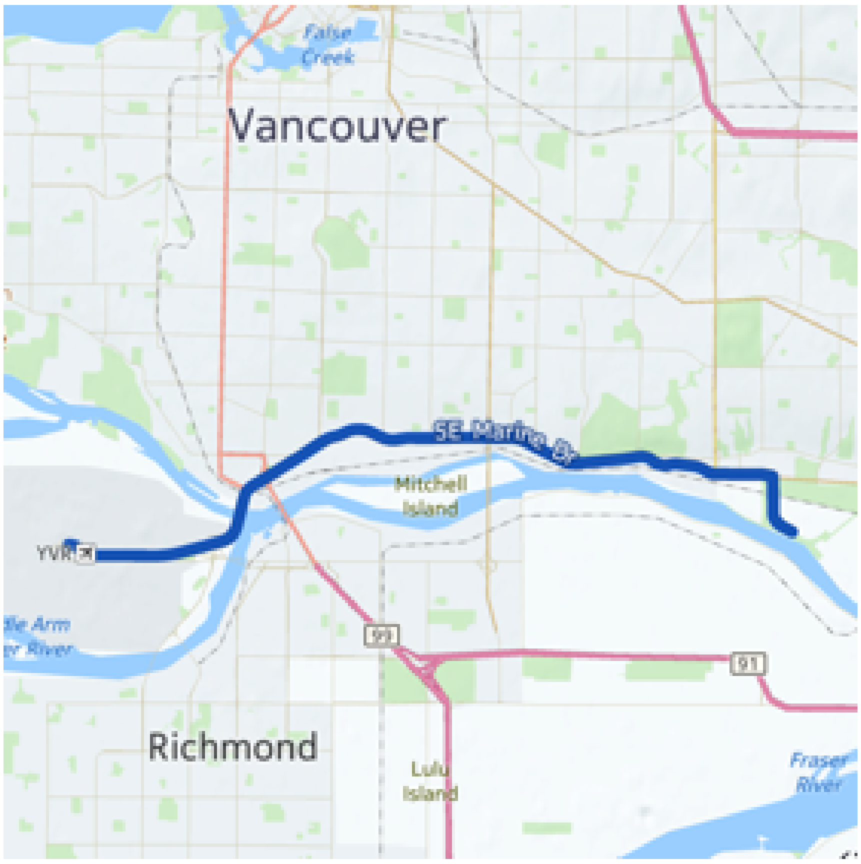
The `Route` class is a distinct calculated path connecting two or more waypoints consisting of a list of maneuvers and route links. By using `RouteManager.calculateRoute(RoutePlan, Listener)` to trigger a route calculation your application can use the `RouteManager.Listener` to monitor the calculation and trigger callback methods. These callback methods have parameters that include a list of the calculated `RouteResult` objects. Using these `RouteResult` objects you can call `getRoute()` to retrieve the routes.

The MapRoute Class

The `MapRoute` class is a type of `MapObject` that displays a calculated route on a map. Typically, an application creates a `MapRoute` after a route calculation passing the relevant `Route` object as a parameter to the `MapRoute(Route)` constructor before adding the MapRoute to the map by calling `Map.addMapObject(MapRoute)`.

For example, if you want to render a route that connects two waypoints (start and destination), you can add the following application logic:

Figure 14: Calculate Route



1. Declare a `RouteManager` instance.

```
// Declare the rm variable (the RouteManager)
RouteManager rm = new RouteManager();
```

2. Create a `RoutePlan` and add two `GeoCoordinate` waypoints.

```
// Create the RoutePlan and add two waypoints
RoutePlan routePlan = new RoutePlan();
routePlan.addWaypoint(new GeoCoordinate(49.1966286, -123.0053635));
routePlan.addWaypoint(new GeoCoordinate(49.1947289, -123.1762924));
```

3. Create a new `RouteOptions` object, set its `Type` and `TransportMode` values by calling appropriate `RouteOptions` methods, and then add it to `RoutePlan`.

```
// Create the RouteOptions and set its transport mode & routing type
RouteOptions routeOptions = new RouteOptions();
routeOptions.setTransportMode(RouteOptions.TransportMode.CAR);
routeOptions.setRouteType(RouteOptions.Type.FASTEST);

routePlan.setRouteOptions(routeOptions);
```

4. To make sure route calculation can handle returning a `Route` object that in turn can be used to create a `MapRoute` instance for rendering on the map, add an inner class by implementing `RouteManager.Listener` in the appropriate activity class.

```

private class RouteListener implements RouteManager.Listener {
  // Method defined in Listener
  public void onProgress(int percentage) {
   // Display a message indicating calculation progress
  }

  // Method defined in Listener
  public void onCalculateRouteFinished(RouteManager.Error error, List<RouteResult> routeResult) {
   // If the route was calculated successfully
   if (error == RouteManager.Error.NONE) {
    // Render the route on the map
    MapRoute mapRoute = new MapRoute(routeResult.get(0).getRoute());
    map.addMapObject(mapRoute);
   }
   else {
    // Display a message indicating route calculation failure
   }
  }
}
```

5. After adding the inner listener class (named `RouteListener` for this example) calculate the route by calling `RouteManager.calculateRoute(RoutePlan, Listener)` (making use of an instantiated `RouteListener` object).

```
// Calculate the route
rm.calculateRoute(routePlan, new RouteListener());
```

Note: Routes with more than 100 waypoints may require significant calculation time.

Routing-related Enumerations

Route calculations make use of HERE SDK enumerations that include:

- The `Route.TrafficPenaltyMode` enum - represents values describing how the `RouteManager` should handle traffic events in a route calculation such as `DISABLED`, `AVOID_LONG_TERM_CLOSURES`, or `OPTIMAL`
- The `RouteOptions.Type` enum - represents values describing different routing types such as `FASTEST`, `SHORTEST`, or `ECONOMIC`
- The `RouteOptions.TransportMode` enum - represents values describing different transport modes such as `CAR` or `PEDESTRIAN`
- The `RouteManager.Error` enum - represents values describing possible route calculation errors, such as `NONE` or `VIOLATES_OPTIONS`
- The `RouteResult.ViolatedOption` enum - represents values describing possible route option violations such as `AVOID_HIGHWAYS` or `AVOID_FERRIES`

Places

This section provides an overview of the Places feature in HERE SDK. The Places feature enables developers to build rich, location-aware applications by adding point of interest search, discovery, interaction, and information retrieval.

For example, when an application submits a place discovery request using this API, the application receives a response that contains a list of links to places resources (among other information). By accessing one of the linked Place resources the application can get detailed information about that place including ratings, images, reviews, editorials, and owner content. This detailed place response also contains references to other related places allowing the application users to discover other places relevant or related to their original search.

Geocoding and Reverse Geocoding

Geocoding and reverse geocoding APIs from HERE Android SDK allow application developers to offer search functionality for requesting location information and structured addresses. Geocoding APIs resolve to a `GeoCoordinate` from a text query while reverse geocoding APIs resolve to a geographic data such as `Address` from a `GeoCoordinate`. `Address` provides textual address information which includes house number, street name, city, country, and district. It encompasses everything about an address or a point on the map. `Location` represents a physical point on the map where additional attributes can be retrieved. These additional attributes include a unique identifier, label, `Address`, `GeoCoordinate` positions, and `GeoBoundingBox` for the `Location`.

The GeocodeRequest Class

`GeocodeRequest` represents an extended `Request`. `GeocodeRequest` can be created using a one-box (free-formatted text) search using a combination of a text query string and geographical area arguments. The following shows the method used to create a one-box request:

```
GeocodeRequest request = new GeocodeRequest(String).setSearchArea(GeoCoordinate, int)
```

The preceding method returns a `GeocodeRequest` object. To begin the search, call `GeocodeRequest.execute()`. This method requires a `ResultListener` as an argument. When the search is completed, `ResultListener.onCompleted()` method is called with a result status and a list of found locations.

After a request is invoked, it can be canceled via `GeocodeRequest.cancel()` method which returns `true` if the request was cancelled successfully. For `GeocodeRequest` a list of `Location` objects are expected at the completion of the request.

The following code example demonstrates how to perform a `GeocodeRequest`:

```
// Implementation of ResultListener
class GeocodeListener implements ResultListener<List<GeocodeResult>> {
 @Override
 public void onCompleted(List<GeocodeResult> data, ErrorCode error) {
  if (error != ErrorCode.NONE) {
   // Handle error
   ...
  } else {
   // Process result data
   ...
  }
 }
}

// Instantiate a GeoCoordinate object
GeoCoordinate vancouver = new GeoCoordinate( 49.2849,- 123.1252);

// Example code for creating a OneBox Request
ResultListener<List<GeocodeResult>> listener = new GeocodeListener();
GeocodeRequest request = new GeocodeRequest("Granville").setSearchArea(vancouver, 5000);
if (request.execute(listener) != ErrorCode.NONE) {
 // Handle request error
 ...
}
```

The ReverseGeocodeRequest Class

The `ReverseGeocodeRequest` class represents an extended `Request` used to retrieve `Address` data. The request is created using a `GeoCoordinate` as shown below:

```
new ReverseGeocodeRequest(GeoCoordinate)
```

The above method returns a `ReverseGeocodeRequest` object. To invoke the request, you can then call `execute()` method of the returned `ReverseGeocodeRequest` object and pass in a `ResultListener` to retrieve the information about the completion of the request. Once a request is invoked, the request can be canceled via `ReverseGeocodeRequest.cancel()` method. `cancel()` returns `true` if the request was canceled successfully. For `ReverseGeocodeRequest` a structured `Address` is expected on the completion of the request.

The following is an example of creating `ReverseGeocodeRequest` using `new ReverseGeocodeRequest(GeoCoordinate)`:

```
// Implementation of ResultListener
class ReverseGeocodeListener implements ResultListener<Location> {
 @Override
 public void onCompleted(Location data, ErrorCode error) {
  if (error != ErrorCode.NONE) {
   // Handle error
   ...
  } else {
   // Process result data
   ...
  }
 }
}

// Instantiate a GeoCoordinate object
GeoCoordinate vancouver = new GeoCoordinate( 49.2849,- 123.1252);

// Example code for creating ReverseGeocodeRequest
ResultListener<Location> listener = new ReverseGeocodeListener();
ReverseGeocodeRequest request = new ReverseGeocodeRequest(vancouver);
if (request.execute(listener) != ErrorCode.NONE) {
 // Handle request error
 ...
}
```

Search and Discovery

HERE Android SDK provides application developers with Places API which allows places discovery and information retrieval.

Steps for performing a search

1. Implement the `ResultListener` interface to handle the completion of the search
2. Create a request and apply request options
3. Invoke the request by calling `Request.execute(ResultListener)`
4. `ResultListener.onCompleted()` callback is triggered when the request is finished

Note: Applications that use Places API must honor the following prescribed workflow:

1. Search
2. Request for Details
3. Perform Actions

Do not preload results linked from a response to improve performance as doing so violates HERE guidelines. For more information about usage of restrictions see *API Implementation Check List* in REST HERE Places API documentation.

The Place Class

The `Place` class represents a detailed set of data about a physical place acting as a container for various attributes, collections of media about a place, and key-value pairs of related places. A `Place` object can belong to a specific `Category` and has attributes such as:

- A unique identifier (ID)
- A name
- A `Location` object representing the physical location of the place including access locations
- A `List` of `Category` objects that link to the categories assigned to the place
- A URL to the icon that best represents the place
- Optional information such as related places, user ratings, reviews, and other editorial media

For more information please see the API Reference.

Discovery Requests

HERE Places Search API supports the following discovery requests:

Request	HERE SDK class	Purpose
Search	`SearchRequest`	Finds places that match user provided search terms.
Explore	`ExploreRequest`	Finds interesting places nearby or in the map viewport sorted by popularity. Use this type of request if you are trying to answer the question "What are the interesting places nearby?" The results may be optionally restricted to a given set of categories which acts as a filter in terms of what places get returned.
Here	`HereRequest`	Helps users identify places at the given location by finding places of interest near a given point sorted by distance. Use this type of request if you are trying to answer the question "What is near this location?" or "Where am I?" You can use this endpoint to implement features like "check-in" (by identifying places at the user's current position) or "tap to get more information about this place of interest". **Note:** Normally, the closest known places are returned with the Here Discovery request but if the uncertainty in the given position is high, then some nearer places are excluded from the result in favor of more popular places in the area of uncertainty.

The following code example demonstrates how to perform a search discovery request:

```
// Example Search request listener
class SearchRequestListener implements ResultListener<DiscoveryResultPage> {

 @Override
 public void onCompleted(DiscoveryResultPage data, ErrorCode error) {
  if (error != ErrorCode.NONE) {
   // Handle error
   ...
  } else {
   // Process result data
   ...
  }
 }
}

// Create a request to search for restaurants in Seattle
try {
 GeoCoordinate seattle
  = new GeoCoordinate(47.592229, -122.315147);

 DiscoveryRequest request =
  new SearchRequest("restaurant").setSearchCenter(seattle);

 // limit number of items in each result page to 10
 request.setCollectionSize(10);

 ErrorCode error = request.execute(new SearchRequestListener());
 if( error != ErrorCode.NONE ) {
  // Handle request error
  ...
 }
} catch (IllegalArgumentException ex) {
 // Handle invalid create search request parameters
 ...
}
```

The result of a discovery request is a `DiscoveryResultPage`. The `DiscoveryResultPage` represents a paginated collection of items from which the following can be retrieved:

- Next page and previous page requests - discovery requests used to retrieve additional pages of search items
- Items for the current page - a `List` of `DiscoveryResult`

When additional pages of search results are needed, retrieve and invoke the `DiscoveryRequest` returned by `DiscoveryResultPage.getNextPageRequest()`. If the next page request is null, no additional results are available.

The following is an example:

```
DiscoveryResultPage mResultPage = null;

// Example Search request listener
class SearchRequestListener implements ResultListener<DiscoveryResultPage> {

 @Override
 public void onCompleted(DiscoveryResultPage data, ErrorCode error) {
  if (error != ErrorCode.NONE) {
   // Handle error
   ...
  } else {
   // Store the last DiscoveryResultPage for later processing
   mResultPage = data;
   ...
  }
 }
}
...

// When the next page of results is needed...
```

```
DiscoveryRequest nextPageRequest = mResultPage.getNextPageRequest();

if (nextPageRequest != null) {
 // More data is available if the nextPageRequest is not null
 ErrorCode error = nextPageRequest.execute(new SearchRequestListener());
 if( error != ErrorCode.NONE ) {
  // Handle request error
  ...
 }
}
```

Calling `DiscoveryResultPage.getItems()` returns a `List` containing one of the following types of objects which are `DiscoveryResult` instances. `DiscoveryResult` is a collection of `Link` subtypes.

- `PlaceLink` - Represents discovery information about a `Place`. The `PlaceLink` contains a brief summary about a place. Details about a place are available from the `Place` that the `PlaceLink` references.
- `DiscoveryLink` - Represents a discovery-related API link used to retrieve additional `DiscoveryResultPage`. This type of `Link` can be a result item in an Explore or Here type of search. The `DiscoveryLink` references refined discovery requests resulting in more specific results. For example, the `DiscoveryLink` may link to a discovery request to search for 'Eat & Drink', 'Going Out', 'Accommodation', and so on.

Since there may be new types of `Link` items in the future, it is recommended that each type of `DiscoveryResult` be checked before it is used (as shown in the following code snippet). In the following example, it is shown how a `Place` is retrieved through a `PlaceLink`:

```
// Implement a search result listener
ResultListener<DiscoveryResultPage> searchListener = new ResultListener<DiscoveryResultPage>() {
 @Override
 public void onCompleted(DiscoveryResultPage results, ErrorCode error) {

  if (error == ErrorCode.NONE) {
   // The results is a DiscoveryResultPage which represents a
   // paginated collection of items.
   List<DiscoveryResult> items = results.getItems();

   // Iterate through the found place items.
   for (DiscoveryResult item : items) {
    // A Item can either be a PlaceLink (meta information
    // about a Place) or a DiscoveryLink (which is a reference
    // to another refined search that is related to the
    // original search; for example, a search for
    // "Leisure & Outdoor").

    if (item.getResultType() == ResultType.PLACE) {
     PlaceLink placeLink = (PlaceLink) item;

     // PlaceLink should be presented to the user, so the link can be
     // selected in order to retrieve additional details about a place
     // of interest.
     ...

    } else if (item.getResultType() == ResultType.DISCOVERY) {
     DiscoveryLink discoveryLink = (DiscoveryLink) item;

     // DiscoveryLink can also be presented to the user.
     // When a DiscoveryLink is selected, another search request should be
     // performed to retrieve results for a specific category.
     ...
    }
   }
  } else {
   // Handle search request error.
```

```
    }
  }
};

...

// Implement a Place listener for handling user interaction with a displayed PlaceLink
class PlaceListener implements ResultListener<Place> {
  @Override
  public void onCompleted(Place data, ErrorCode error) {
    if (error != ErrorCode.NONE) {
      // Handle error
      ...
    } else {
      // Present the place details to the user.
      String placeName = data.getName();
      List<Category> placeCategories = data.getCategories();
      ...
    }
  }
}

// Retrieve the place details when the user selects a displayed PlaceLink.
private void onPlaceLinkSelected(PlaceLink placeLink) {
  PlaceRequest placeRequest = placeLink.getDetailsRequest();
  if( placeRequest.execute(new PlaceListener()) == ErrorCode.NONE ) {
    // Request successful. Additional work can be done here, however, place details will
    // be returned in PlaceListener.onCompleted().
    ...
  } else {
    // Handle the error
    ...
  }
}
```

Text AutoSuggestion Requests

HERE Places Search API also supports text autosuggestion requests. This type of request is used for retrieveing a list of suggested search terms (`AutoSuggestQuery`), instant results (`AutoSuggestPlace`), and refined search links (`AutoSuggestSearch`) that are related to a specified location context and a partial search term. For example, if you make a request with the String "rest" in Berlin, the results contain search terms such as "Restaurant", "Rest area", and "Restorf, Höhbeck, Germany".

To use text suggestions, implement a listener to handle a list of `AutoSuggest` objects and call `new TextAutoSuggestionRequest(String)` as follows:

```
// Example request listener
class AutoSuggestionQueryListener implements ResultListener<List<AutoSuggest>> {

  @Override
  public void onCompleted(List<AutoSuggest> data, ErrorCode error) {
    for (AutoSuggest r : data) {
      try {
        String term = "rest";
        TextAutoSuggestionRequest request = null;
        request = new TextAutoSuggestionRequest(term).setSearchCenter(myMap.getCenter());
        if (request.execute(new AutoSuggestionQueryListener()) !=
            ErrorCode.NONE ) {
          //Handle request error
          //...
        }
      } catch (IllegalArgumentException ex) {
        //Handle invalid create search request parameters
      }
    }
  }
}
```

You can retrieve the results of a `TextAutoSuggestionRequest` by first checking the autosuggest object type as shown in the following example. Note that it is possible for `AutoSuggestSearch` to contain additional paginated results through the `DiscoveryRequest` object. If the object is `AutoSuggestPlace`, you can request for more details through its `PlaceRequest` object. `AutoSuggestQuery` contains additional results through its `TextAutoSuggestionRequest` object.

```
//assume autoSuggestList contains the list of results
try {
    AutoSuggest autoSuggest = autoSuggestList.get(index);

    // set title
    String title = autoSuggest.getTitle();
    // get highlightedTitle
    String highlightedTitle = Html.fromHtml(autoSuggest.getHighlightedTitle()).toString();

    if (autoSuggest instanceof AutoSuggestPlace) {

        AutoSuggestPlace autoSuggestPlace = (AutoSuggestPlace)autoSuggest;

        // vicinity
        if (autoSuggestPlace.getVicinity() != null) {
            String vicinity = autoSuggestPlace.getVicinity();
        }

        // set category
        if (autoSuggestPlace.getCategory() != null) {
            String category = autoSuggestPlace.getCategory();
        }

        // set position
        if (autoSuggestPlace.getPosition() != null) {
            String position = autoSuggestPlace.getPosition().toString();
        }

        // set boundaryBox
        if (((AutoSuggestPlace)autoSuggest).getBoundingBox() != null) {
            String boundingBox = ((AutoSuggestPlace)autoSuggest).getBoundingBox().toString();
        }
    } else if (autoSuggest instanceof AutoSuggestSearch) {

        AutoSuggestSearch autoSuggestSearch = (AutoSuggestSearch)autoSuggest;

        // set category
        if (autoSuggestSearch.getCategory() != null) {
            String category = autoSuggestSearch.getCategory();
        }

        // set position
        if (autoSuggestSearch.getPosition() != null) {
            String position = autoSuggestSearch.getPosition().toString();
        }

        // set boundaryBox
        if (autoSuggestSearch.getBoundingBox() != null) {
            String boundingBox = autoSuggestSearch.getBoundingBox().toString();
        }

        DiscoveryRequest myDiscoveryRequest = autoSuggestSearch.getSuggestedSearchRequest();
        myDiscoveryRequest.execute(myDiscoveryResultPagelistener);
    } else if (autoSuggest instanceof AutoSuggestQuery) {

        AutoSuggestQuery autoSuggestQuery = (AutoSuggestQuery)autoSuggest;

        // set completion
        if (autoSuggestQuery.getQueryCompletion() != null) {
            String completion = autoSuggestQuery.getQueryCompletion();
        }

        TextAutoSuggestionRequest myAutoSuggestionRequest = autoSuggestQuery.getRequest();
        myAutoSuggestionRequest.execute(myAutoSuggestionResultPagelistener);

    }

} catch (Exception e) {
    Log.e("ERROR: ", e.getMessage());
}
```

Chapter 4
Supplemental Information

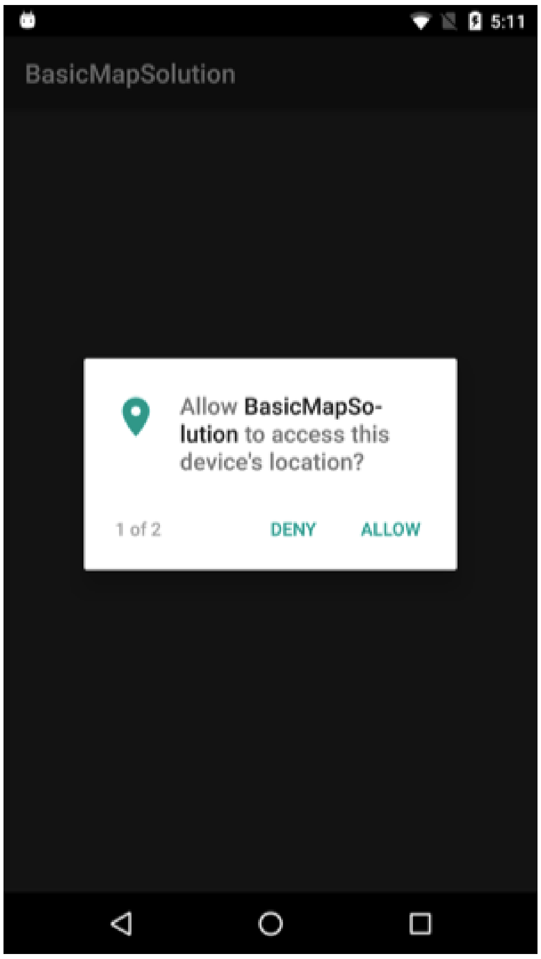
Topics:

This section provides supplemental information for using HERE Android SDK.

Requesting Android Permissions

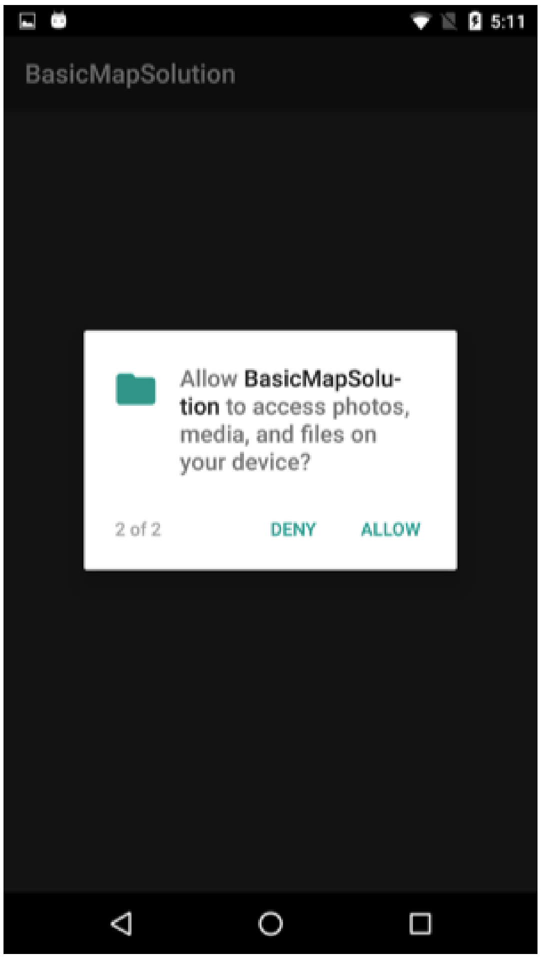
If your application supports Android 6.0 or above, add the following code in your activity file to ask the application users to grant Android permissions at runtime. For more information about this requirement, see the *Android Developer documentation*.

Figure 15: Request Location Permission



Allow **BasicMapSolution** to access this device's location?

Figure 16: Request File Access Permission



Allow **BasicMapSolution** to access photos, media, and files on your device?

1. Add the following `import` statements to the beginning of the file:

```
import android.content.pm.PackageManager;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.Manifest;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

2. Add these static variables to `BasicMapActivity` class:

```
/**
 * permissions request code
 */
private final static int REQUEST_CODE_ASK_PERMISSIONS = 1;

/**
 * Permissions that need to be explicitly requested from end user.
 */
private static final String[] REQUIRED_SDK_PERMISSIONS = new String[] {
  Manifest.permission.ACCESS_FINE_LOCATION };
```

Note: `android.permission.ACCESS_FINE_LOCATION` is not required to initialize or use the SDK. However, this Android permission is required if you use `PositioningManager`.

3. Add the following methods to `BasicMapActivity` class:

```
/**
 * Checks the dynamically-controlled permissions and requests missing permissions from end user.
 */
protected void checkPermissions() {
    final List<String> missingPermissions = new ArrayList<String>();
    // check all required dynamic permissions
    for (final String permission : REQUIRED_SDK_PERMISSIONS) {
        final int result = ContextCompat.checkSelfPermission(this, permission);
        if (result != PackageManager.PERMISSION_GRANTED) {
            missingPermissions.add(permission);
        }
    }
    if (!missingPermissions.isEmpty()) {
        // request all missing permissions
        final String[] permissions = missingPermissions
                .toArray(new String[missingPermissions.size()]);
        ActivityCompat.requestPermissions(this, permissions, REQUEST_CODE_ASK_PERMISSIONS);
    } else {
        final int[] grantResults = new int[REQUIRED_SDK_PERMISSIONS.length];
        Arrays.fill(grantResults, PackageManager.PERMISSION_GRANTED);
        onRequestPermissionsResult(REQUEST_CODE_ASK_PERMISSIONS, REQUIRED_SDK_PERMISSIONS,
                grantResults);
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String permissions[],
        @NonNull int[] grantResults) {
    switch (requestCode) {
        case REQUEST_CODE_ASK_PERMISSIONS:
            for (int index = permissions.length - 1; index >= 0; --index) {
                if (grantResults[index] != PackageManager.PERMISSION_GRANTED) {
                    // exit the app if one permission is not granted
                    Toast.makeText(this, "Required permission '" + permissions[index]
                            + "' not granted, exiting", Toast.LENGTH_LONG).show();
                    finish();
                    return;
                }
            }
            // all permissions were granted
            initialize();
            break;
    }
}
```

4. Finally, change the method call in `onCreate(Bundle)` from `initialize()` to `checkPermissions()` instead:

```
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 checkPermissions();
}
```

Device and Emulator Support

HERE Android SDK includes support for x86 and x86_64 Application Binary Interface (ABI). The x86 ABI enables you to run apps with x86 or x86_64 Android Virtual Devices (AVD).

For more information on ABIs, see *https://developer.android.com/ndk/guides/abis.html*.

For more information on how to create AVDs, see *https://developer.android.com/studio/run/managing-avds.html*.

Note: To get the best possible performance on an AVD, install and enable Intel HAXM by following these instructions: *https://developer.android.com/studio/run/emulator-acceleration.html*.

By default you do not need to make any changes to your development environment to include x86 or x86_64 native libraries. However, any `.apk` file that you build with HERE SDK will be larger since it contains more native libraries. Optionally, you can follow *this guide* to create multiple `.apk` files, with each containing native libraries for a specific ABI and thus reducing application sizes.

Adding a SupportMapFragment at Runtime

Earlier tutorials in this document featured adding a `SupportMapFragment` to an activity by editing the layout XML file. You can also add a `SupportMapFragment` to an activity dynamically, during runtime, by performing the following steps in the `Activity` class:

1. Create a layout container:

```
final int CONTAINER_ID = 1234567;
LinearLayout layoutContainer = new LinearLayout(this);
layoutContainer.setOrientation(LinearLayout.HORIZONTAL);
layoutContainer.setId(CONTAINER_ID);
```

2. Define a map tag:

```
final String MAP_TAG = "map_tag";
```

3. Create a map fragment and add it using the fragment manager:

```
mapFragment = new SupportMapFragment();
getFragmentManager().beginTransaction().add(layoutContainer.getId(), mapFragment,
 MAP_TAG).commit();
```

4. Initialize the map fragment by implementing `OnEngineInitListener`:

```
MyOnEngineInitListener onEngineInitListener = new MyOnEngineInitListener();
ApplicationContext context = new ApplicationContext(this);
mapFragment.init(context, onEngineInitListener);
```

5. Finally, show the content view:

```
setContentView(layoutContainer);
```

For more information on adding a fragment at runtime, see this article: *https://developer.android.com/training/basics/fragments/fragment-ui.html#AddAtRuntime*.

Development Tips

This section provides tips on building your application using HERE Android SDK.

Upgrading from Older Versions of HERE SDK

HERE Android SDK is now packaged as an Android archive (AAR) file instead of separate JAR, native library, and proguard components. If you are upgrading from an older HERE SDK release, the old components should be cleaned up before integrating the AAR version of HERE SDK. To do so, follow these steps:

1. Ensure that `HERE-sdk.jar` file is removed from your project and the compile entry is removed from your `build.gradle` file. The JAR may be located at `app/libs/HERE-sdk.jar` and included in your `build.gradle` file in one of the following ways:

   ```
   compile files('libs/HERE-sdk.jar')
   ```

   ```
   compile fileTree(dir: 'libs', include: ['*.jar'])
   ```

   Note that if you were previously using Google GSON library or JTS Topology Suite library with HERE SDK, it is still required to be included separately.
2. Remove HERE SDK proguard file and the proguard entry specific to HERE SDK from `build.gradle` file. The file to remove is named `proguard-here-sdk.txt`, and the entry with the same name should also be removed from `proguardFiles` property in your `build.gradle` file. The proguard instructions for newer versions of HERE SDK are now applied automatically and are included in the AAR.

You can find further info on integrating the AAR version of HERE SDK into your app in section *Run the Sample Application* on page 11 of the User Guide and associated `HERE-sdk/tutorial/BasicMapSolution/app/build.gradle` file.

Lapsed Listeners and Garbage Collection

HERE SDK provides a number of listener interfaces such as `Map.OnSchemeChangedListener`, `Map.OnTransformListener`, and `MapGesture.OnGestureListener`. To use these listeners, you are required to implement and create a listener instance, then register it with another object (using a method such as `addSchemeChangedListener()`) to receive event notifications. Unfortunately, this coding pattern can also lead to *lapsed listener problem* when available memory is consumed by listener objects that are not explicitly unregistered and not garbage collected.

To mitigate this problem, HERE SDK, in some cases, accepts listener objects in `WeakReference` containers. This has the advantage of avoiding lapsed listeners but it also means that you must be aware of registered listeners becoming garbage collected. To avoid any unintended issues with this coding pattern, be sure to retain a strong reference to your listener instances (for example, by assigning it to a class variable) if you would like to manage its garbage collection lifecycle. Listener objects are not garbage collected as long as a strong reference exists.

Working with Getters

Classes in HERE SDK return copies of objects in its getters. For example, `MapPolyline.getPolyline()` does not return the same `GeoPolyline` instance that was used to construct the `MapPolyline` object; instead, a copy of the `GeoPolyline` is returned. Since this returned object is a copy, you cannot dynamically modify the `MapPolyline` instance by modifying this object. If you would like to make changes to `MapPolyline`, you must call `setGeoPolyline(GeoPolyline)` instead.

Map Object Limitations

HERE SDK does not limit the number of map markers, polygons, and polylines that can be added to a map. However, rendering a large number of map objects can cause performance degradation in your application. It is recommended that you use techniques such as viewport clipping to avoid these issues.

Doze and App Standby

If you are using Android 6.0 (API level 23) or above, be aware that *Doze* and *App Standby* features may impact your HERE SDK app by disabling network access when the device is unplugged, stationary, and has the screen off for a period of time. While HERE Android SDK has the ability to work offline, you should design your app with these operating system features in mind.

For more information about Doze and App Standby including how to use notifications and whitelisting to ensure your app functions properly, see the Android article, *"Optimizing for Doze and App Standby"*.

Native Libraries and ABI Splits

Your app may encounter an error if it also *includes other dependencies that have unsupported ABIs*. To get around this issue, enable ABI splits to build for `armeabi-v7a`, `arm64-v8a`, or both architectures explicitly by modifying your app `build.gradle` file:

```
android {
 (...)
 splits {
  abi {
   enable true
   reset()
   include 'armeabi-v7a', 'arm64-v8a' // or choose one of them
   universalApk false
  }
 }
 (...)
}
```

For more information about the `splits` Gradle block, see *Configure multiple APKs for ABIs* in the Android Studio User Guide.

Chapter 5
Coverage Information

The following list provides coverage information for HERE Android SDK features. Feature support in HERE SDK may differ depending on the language and locale.

- *Satellite Imagery*
- *Public Transit*
- *Routing*
- *Point Address* (such as house numbers)
- *Online Geocoding / Reverse Geocoding*
- *Online Places and Search*