Archived:Using Java ME Location API with Google Static Maps

This example shows how to get the GPS position using the Java ME Location API, and plot the position using Google Maps.

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Since this article was written, alternative **Map Tile caching** solutions for Java ME have become available. **Map Tile caching** solutions such as the Nokia Maps API for Java ME have several advantages over **static mapping** solutions such as the static Google Maps API including:

- **Static** mapping services such as the Google Static Maps API or Nokia's RESTful Map API do not cache or tile the images when requested, therefore **each** request involves a round trip to the server. If the map on a mobile application needs to be refreshed at any time, using a caching library will result in a reduction in network traffic after around **three** maps have been displayed. An explanation of this can be found here
- As the name implies, Google's Static Maps API can only retrieve over http static images for a requested coordinate point, image size, image type and zoom level. Newer libraries offer additional functionality out of the box offering dynamic Map content and touch support, where the user can move around his/her current position, zoom in, zoom out, modify the view mode to satellite or translate an address to a coordinate point and show that on the map, among others. This abstraction of the underlying functionality is hidden from the developer,os much less coding is needed in order to achieve the same result .

Additionally the following points apply in favour of Nokia Maps when comparing to Google Maps:

- No legal restrictions of using the API outside a web browser application or need to provide a link to the native Google Maps App (if there is one), or to Google Maps (if there isn't one). See Terms of Service below.
- Higher free daily request limits. Nokia Maps API for Java ME supports up to 50,000 render requests per day and per unique IP address (as of January 2012), for Nokia Developer registered users (free of charge) while the limit for Google's Static Maps API is currently 1000 unique (different) image requests per viewer per day.

An article with the same functionality, written with Nokia Maps API for Java ME, that uses much less code can be found here

The Location API (JSR 179)

The Location API (JSR 179 - an optional package) javax.microedition.location aims at helping the development of location-based applications and services for resource-limited devices, like mobile phones. The application must be set for CLDC 2.0 (CLDC 1.0) does not support floating-point number, which the API uses to represent coordinates and other measurements).

The Google Map API

NOTE: Usage of this code with the free Google Maps API Key breaks Google's Terms and Conditions (section 10.8). You should purchase an Enterprise License if you wish to use the Google Maps API as shown in this example.

The Google Map API allows you embed Google Maps in your own web pages with JavaScript. Therefore, we will just use a web services REST to get the map image and plot in the device's screen. We just must send a request for

```
http://maps.google.com/staticmap?params
```

URL and set some parameters, such as, latitude and longitude coordinates, image's size, zoom, map type (mobile, etc) and, if you wish, apply a marker (a point) to the static image. It is important to note that the example below targets devices for S60 series.

Example

The simple ideia is: (1) get the GPS position and (2) pass the latitude and longitude coordinates to the Web Services REST (url) of the Google Maps, (3) get the server response image in byte array type. So, our first step is shown below: The algorithm below must be implemented in a different thread. Otherwise, user interface may be blocked until the response comes back.

```
Criteria cr = new Criteria();
cr.setHorizontalAccuracy(500);
cr.setPreferredPowerConsumption(Criteria.NO_REQUIREMENT);

LocationProvider provider = null;
Location location             = null;

try {
 provider = LocationProvider.getInstance(cr);
 location = provider.getLocation(120);
 provider.setLocationListener(this, -1, 0, 0);
} catch (LocationException e) {
 alert = new Alert("Error", "Location API Error", null, AlertType.ERROR);
} catch (InterruptedException e) {
 alert = new Alert("Error", "Thread Error", null, AlertType.ERROR);
}

Coordinates c = location.getQualifiedCoordinates();
if(c != null) {
 this.lat = c.getLatitude();
 this.lon = c.getLongitude();
}
```

In the beginning of the sample code above, we first set some criteria. Criteria fields include: accuracy, response time, need for altitude, and speed. This is

(C) Copyright Nokia 2013. All rights reserved.

up to the application to set these values. These criteria fields will filter the LocationProvider that meets the given values. After obtained a LocationProvider object, the application can use this object to obtain the location, in either of two ways:

- Invoke a method synchronously to get a single location.
- Register a listener and get periodic updates at application-defined intervals.

The location results can be got with the Location class. An instance of this class can available some critical informations, such as: coordinates, speed and textual address (if available), and a time stamp that indicates when the location measurements were made.

Coordinates can be represented with two different classes:

- Coordinates:a point's latitude and longitude in degrees, and altitude in meters.
- QualifiedCoordinates:latitude, longitude, and altitude, and also an indication of their accuracy, represented as the radius of an area.

Finally, we have our main information: latitude (through c.getLatitude() method) and longitude (through c.getLongitude() method). All we have to do now is open a HTTP connection with the Google Maps server.

```
HttpConnection httpConnection = null;
InputStream inputStream = null;
byte[] buffer = null;

try {
 httpConnection = (HttpConnection) Connector.open(url);
 httpConnection.setRequestMethod(HttpConnection.GET);
 is = httpConnection.openInputStream();
 int length = (int)httpConnection.getLength();
 if(length > 0) {
  buffer = new byte[length];
  is.read(buffer);
 } else {
         int c;
  ByteArrayOutputStream byteArray = new ByteArrayOutputStream();
  while ((c = inputStream.read()) != -1) {
   byteArray.write(c);
  }
  buffer = byteArray.toByteArray();
  byteArray.close();
 }
} catch(Exception e) {
 alert = new Alert("Error", "Connection Failed", null, AlertType.ERROR);
} finally {
 try {
  if(inputStream != null)
                 inputStream !=.close();
  if(httpConnection != null)
   httpConnection.close();
  }
 catch(Exception e2) {
  alert = new Alert("Error", "IO Failed!", null, AlertType.ERROR);
 }
}

return buffer;
```

The above code opens a http connection with a given URL and gets the server's response. This URL is a Web Service REST available by Google Maps, and is as follows:

```
http://maps.google.com/staticmap?center=lat,lon&amp;zoom=16&amp;size=200x200&amp;maptype=mobile&amp;markers=lat,lon,ref/&amp;key=MAP
```

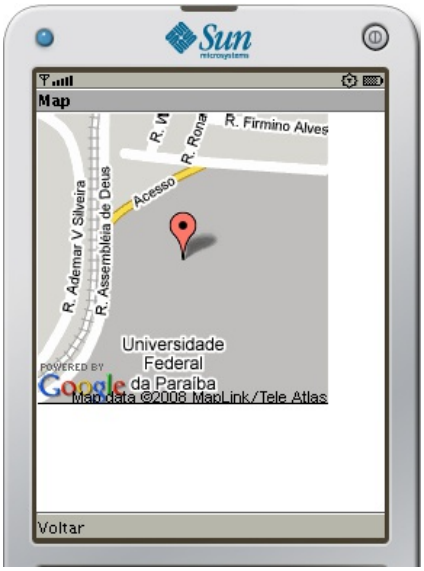
In this URL, we set some parameters for the static image: latitude and longitude, its zoom and its size (200x200), the map type (for mobile application) and a marker plotted based on lat and lon values. In our example, the application got (through GPS positioning) the Federal University of Campina Grande latitude and longitude values.

The image returned from the web services (byte[]) is used to print the map in the screen.

```
Form f = new Form("Map");
f.append(Image.createImage(buffer, 0, buffer.length))
```

The result is:

 (C) Copyright Nokia 2013. All rights reserved.



(C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.