# Archived:Google Static Maps API in Java ME

This article describes how to search for an address or a point of interest on Google Maps and display it on the phone's screen. It uses Google's Static Maps API for the translation of the search string into valid coordinates and for the retrieval of the relevant map image.

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:

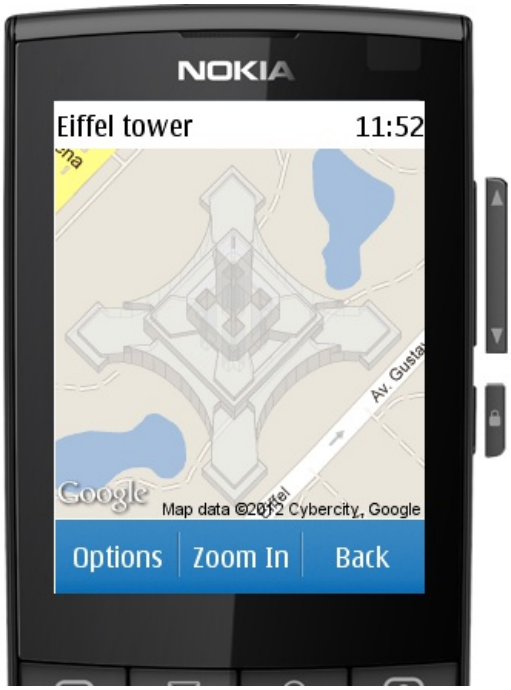
- **Static** mapping services such as the Google Static Maps API or Nokia's RESTful Map API do not cache or tile the images when requested, therefore **each** request involves a round trip to the server. If the map on a mobile application needs to be refreshed at any time, using a caching library will 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

(C) Copyright Nokia 2014. All rights reserved.




## Introduction

The example described in this article includes the following features:

- Retrieval of the geographic coordinates for a given address or a place
- Retrieval of static images with given custom size, format and zoom level
- Zooming in and out functionality

## Terms of Service

Note: Google's Static Maps API no longer requires a Google Maps API key. However if you are not a Google Maps API Premier user, you should not use the code on this page outside a web-based application according to the Google Maps API Terms of Service Terms and Conditions (section 10.1.1(h)), unless:

1. your platform does not have a web browser or
2. your platform does have a web browser, but you provide a link that shows the location concerned either in the Google Maps native application (if your platform has one) or on the Google Maps website.

Also for non Premier users there is a limitation of 1000 unique (different) image requests per viewer per day.

## Disabling the On Screen Keypad

In order to take advantage of the entire screen, the following line should be added to the application's file descriptor (jad file):

**Nokia-MIDlet-On-Screen-Keypad: no**

## Source code: GoogleMaps class

```
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Vector;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Image;

public class GoogleMaps {
```


(C) Copyright Nokia 2014. All rights reserved.

```
    private static final String URL_UNRESERVED =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ" +
        "abcdefghijklmnopqrstuvwxyz" +
        "0123456789-_.~";
    private static final char[] HEX = "0123456789ABCDEF".toCharArray();

    // these 2 properties will be used with map scrolling methods. You can remove them
if not needed
    public static final int offset = 268435456;
    public static final double radius = offset / Math.PI;

    public GoogleMaps() {
    }

    public double[] geocodeAddress(String address) throws Exception {
        byte[] res = loadHttpFile(getGeocodeUrl(address));
        String[] data = split(new String(res), ',');

        if (!data[0].equals("200")) {
            int errorCode = Integer.parseInt(data[0]);
            throw new Exception("Google Maps Exception: " + getGeocodeError(errorCode));
        }

        return new double[] {
                Double.parseDouble(data[2]), Double.parseDouble(data[3])
        };
    }

    public Image retrieveStaticImage(int width, int height, double lat, double lng, int
zoom,
            String format) throws IOException {
        byte[] imageData = loadHttpFile(getMapUrl(width, height, lng, lat, zoom,
format));

        return Image.createImage(imageData, 0, imageData.length);
    }

    private static String getGeocodeError(int errorCode) {
        switch (errorCode) {
        case 400:
            return "Bad request";
        case 500:
            return "Server error";
        case 601:
            return "Missing query";
        case 602:
            return "Unknown address";
        case 603:
            return "Unavailable address";
        case 604:
            return "Unknown directions";
        case 620:
            return "Too many queries";
        default:
            return "Generic error";
        }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    private String getGeocodeUrl(String address) {
        return "http://maps.google.com/maps/geo?q=" + urlEncode(address) +
"&output=csv";
    }

    private String getMapUrl(int width, int height, double lng, double lat, int zoom,
String format) {
        return "http://maps.google.com/staticmap?center=" + lat + "," + lng + "&format="
                + format + "&zoom=" + zoom + "&size=" + width + "x" + height ;
    }

    private static String urlEncode(String str) {
        StringBuffer buf = new StringBuffer();
        byte[] bytes = null;
        try {
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            DataOutputStream dos = new DataOutputStream(bos);
            dos.writeUTF(str);
            bytes = bos.toByteArray();
        } catch (IOException e) {
            // ignore
        }
        for (int i = 2; i < bytes.length; i++) {
            byte b = bytes[i];
            if (URL_UNRESERVED.indexOf(b) >= 0) {
                buf.append((char) b);
            } else {
                buf.append('%').append(HEX[(b >> 4) & 0x0f]).append(HEX[b & 0x0f]);
            }
        }
        return buf.toString();
    }

    private static byte[] loadHttpFile(String url) throws IOException {
        byte[] byteBuffer;

        HttpConnection hc = (HttpConnection) Connector.open(url);
        try {
            hc.setRequestMethod(HttpConnection.GET);
            InputStream is = hc.openInputStream();
            try {
                int len = (int) hc.getLength();
                if (len > 0) {
                    byteBuffer = new byte[len];
                    int done = 0;
                    while (done < len) {
                        done += is.read(byteBuffer, done, len - done);
                    }
                } else {
                    ByteArrayOutputStream bos = new ByteArrayOutputStream();
                    byte[] buffer = new byte[512];
                    int count;
                    while ( (count = is.read(buffer)) >= 0 ) {
                        bos.write(buffer, 0, count);
                    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    byteBuffer = bos.toByteArray();
                }
            } finally {
                is.close();
            }
        } finally {
            hc.close();
        }

        return byteBuffer;
    }

    private static String[] split(String s, int chr) {
        Vector res = new Vector();

        int curr;
        int prev = 0;

        while ( (curr = s.indexOf(chr, prev)) >= 0 ) {
            res.addElement(s.substring(prev, curr));
            prev = curr + 1;
        }
        res.addElement(s.substring(prev));

        String[] splitted = new String[res.size()];
        res.copyInto(splitted);

        return splitted;
    }
}
```

## Addition of a map scrolling functionality

This example can be extended to include scrolling, i.e. the ability to move around the current map point, by catching the user's input and by retrieving new images. For that, the new center for the static image should be calculated. The following **adjust()** method returns the latitude and longitude values for the new map's center. The method requires the following arguments:

- the current center's **latitude** and **longitude** coordinates
- the **deltaX** and **deltaY** values, which represent the deviation in pixels, of the new map center from the current one
- the map's **zoom** level

Original code, in JavaScript, can also be found here

Note: In order to use the methods described below, the **MicroFloat** library should be included in the project. The library is available here: MicroFloat website

Note: Instead of coding the scrolling functionality, as an alternative solution one could use **Nokia's Maps API for Java ME**, where this is provided by default as a MapCanvas. The API is provided as external library and it should therefore be included in the project. The library is available here].

```
public double[] adjust(double lat, double lng, int deltaX, int deltaY, int z)
{
 return new double[]{
  XToL(LToX(lng) + (deltaX<<(21-z))),
  YToL(LToY(lat) + (deltaY<<(21-z)))
```


(C) Copyright Nokia 2014. All rights reserved.

```
 };
}
double LToX(double x)
{
 return round(offset + radius * x * Math.PI / 180);
}

double LToY(double y)
{
 return round(
  offset - radius *
  Double.longBitsToDouble(MicroDouble.log(
   Double.doubleToLongBits(
   (1 + Math.sin(y * Math.PI / 180))
   /
   (1 - Math.sin(y * Math.PI / 180))
   )
  )) / 2);
}

double XToL(double x)
{
 return ((round(x) - offset) / radius) * 180 / Math.PI;
}

double YToL(double y)
{
 return (Math.PI / 2 - 2 * Double.longBitsToDouble(
    MicroDouble.atan(
     MicroDouble.exp(Double.doubleToLongBits((round(y)-offset)/radius))
    )
   )) * 180 / Math.PI;
}
double round(double num)
{
 double floor = Math.floor(num);

 if(num - floor >= 0.5)
  return Math.ceil(num);
 else
  return floor;
}
```

## Description of example's main calls

The GoogleMaps class, contains the methods for geocoding addresses and map image retrievals. It can be instantiated as follows:

```
GoogleMaps gMap = new GoogleMaps();
```

In order to geocode an address or a point of interest, the geocodeAddress() method of the GoogleMaps class should be called as follows:

```
double[] lanLng = gMap.geocodeAddress("Eiffel tower");
```


(C) Copyright Nokia 2014. All rights reserved.

In order to retrieve a map image that corresponds to a given point, we have to specify the image's width and height, the point's latitude and longitude values, as well as the zoom level and the image type:

```
Image map = gMap.retrieveStaticImage(width, height, lanLng[0], lanLng[1], 14, "png32");
```

Note that the zoom level can be any value from 0 to 21.

## Resources

The example's source code can be downloaded from here: File:GoogleMapsMIDletSource.zip

The example's installation binary files can be downloaded from here: File:GoogleMapsMIDletBinaries.zip

## See also

- Maps API for Java ME
- How to display an Address or a Point of Interest on a Map with Java ME
- Using a JSR-179 Location Provider and displaying the result on a Map
- How to create custom Map View in Java ME

## Version Hint

**Windows Phone:** [[Category:Windows Phone]]
[[Category:Windows Phone 7.5]]
[[Category:Windows Phone 8]]

**Nokia Asha:** [[Category:Nokia Asha]]
[[Category:Nokia Asha Platform 1.0]]

**Series 40:** [[Category:Series 40]]
[[Category:Series 40 1st Edition]] [[Category:Series 40 2nd Edition]]
[[Category:Series 40 3rd Edition (initial release)]] [[Category:Series 40 3rd Edition FP1]] [[Category:Series 40 3rd Edition FP2]]
[[Category:Series 40 5th Edition (initial release)]] [[Category:Series 40 5th Edition FP1]]
[[Category:Series 40 6th Edition (initial release)]] [[Category:Series 40 6th Edition FP1]] [[Category:Series 40 Developer Platform 1.0]] [[Category:Series 40 Developer Platform 1.1]] [[Category:Series 40 Developer Platform 2.0]]

**Symbian**: [[Category:Symbian]]
[[Category:S60 1st Edition]] [[Category:S60 2nd Edition (initial release)]] [[Category:S60 2nd Edition FP1]] [[Category:S60 2nd Edition FP2]] [[Category:S60 2nd Edition FP3]]
[[Category:S60 3rd Edition (initial release)]] [[Category:S60 3rd Edition FP1]] [[Category:S60 3rd Edition FP2]]
[[Category:S60 5th Edition]]
[[Category:Symbian^3]] [[Category:Symbian Anna]] [[Category:Nokia Belle]]


(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.