



**nutiteq**

## **Nutiteq Android mapping tutorial**

Version 1.1.1 (updated 24.08.2011)

© 2008-2011 Nutiteq LLC

**Nutiteq**

[www.nutiteq.com](http://www.nutiteq.com)

Skype: nutiteq

Address: Riia 181A, Tartu Science Park, Tartu 51014, Estonia



## **1 Contents**

2	Introduction.....	3
2.1	Document history.....	4
3	Simple Map.....	5
4	Zoom Controls .....	8
5	Correct way to close the app.....	9
6	Show GPS location on map.....	10
7	Adding KML data to the map.....	11
8	Logging for troubleshooting .....	12
9	Full Hello Map listing .....	12



## 2 Introduction

Nutiteq mobile mapping library is designed to enable adding custom slippy maps to mobile applications.

To get started, we assume that you know some basics of Android development, at least how to compile basic applications; there are a lot of good tutorials about it, e.g.

<http://developer.android.com/guide/tutorials/hello-world.html>

Following tutorial includes only most basic functions and examples of the library, full library has much more features and options, see *Nutiteq Maps Lib Developer guide* and on-line javadoc from [www.nutiteq.com](http://www.nutiteq.com) > **Developer** section to have full overview of functionality.

Google Android SDK package has also bundled **Google Maps External Library** package (com.google.android.maps), which provides also basic mapping features. This library is quite similar to Nutiteq library, following is feature comparison between Nutiteq and Google Maps library:

	<b>Nutiteq Library</b>	<b>Google Maps library</b>
<b>Map sources</b>	OpenStreetMap CloudMade Navteq (MapTP) DeCarta DigitalGlobe (satellite and aerial) BLOM Urbex (aerial) Bing (Microsoft) Maps Yahoo! Maps WMS API TMS API Ka-Map API <i>Custom map sources</i>	Google Maps Google Aerial images Google StreetView
<b>Offline maps, caching</b>	Caching in memory <b>Permanent caching in SD card</b> <b>Pre-loaded maps from SD card</b> <b>Pre-loaded maps from application installer package</b>	Caching in memory only
<b>Projections</b>	WGS84, Spherical Mercator Mercator <i>Custom projections</i>	WGS84 for data Spherical Mercator for maps
<b>Map overlays</b>	Points User dynamic location Lines, Polygons <b>KML overlays with points, lines and polygons</b> <b>Raster overlays</b>	Points Lines and Polygons in latest versions User dynamic location
<b>Routing service</b>	Included, using OSM-based ( CloudMade, Yournavigation) or	No

	own custom servers based on OpenLS	
<b>Geocoding (address search) service</b>	Included, using OSM-based ( CloudMade, Namefinder) or own custom servers	No, available as separate service
<b>Visual indicators</b>	Customizable DownloadDisplay Customizable ZoomDisplay Customizable Copyright overlay	None
<b>API Compatibility with other mobile platforms</b>	Java ME (J2ME) RIM BlackBerry Symbian (with J2ME)	None
<b>Licensing terms</b>	Commercial royalty-free licenses available from Nutiteq. Free for GPL applications and developers. Commercial support available. Source code available for licensees.	Free for most applications, expensive license needed for some application types (e.g. fleet management). No official commercial support, no source code available.

## 2.1 Document history

Who	When	Rev	What
JaakL	07.12.2009	1.0.2	First Android-specific version, for lib version 1.0.2.
JaakL	24.08.2011	1.1.1	Updated for 1.1.1 Nutiteq SDK version

### 3 Simple Map

Following are steps for your “Hello world” type of Android interactive mapping application called *Hello map*. We assume that you have installed Android SDK, Eclipse IDE (I use 3.4.2) with compatible Android Development Tools (ADT) Plug-in for Eclipse.

1. Create new Android project: in Eclipse, select File > New project > Android, Android Project. Let’s call it **hello\_map**. Pick any existing Android Build Target you have a working emulator for. I use Android 1.6. (MGMaps lib works with Android SDK 1.1 and up.)
2. Under Properties section: name your application to *Hello map* and use **com.tutorial** as a package name. Also make sure, that Create Activity is checked and use **HelloMap** as name of the activity class. Click *Finish*
3. You should end up with the default Android project code:

```
package com.tutorial;

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

public class HelloMap extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

4. Now configure Android Manifest file. Open **AndroidManifest.xml** file and go to *Permissions* tab. We need to add two permissions here. Under *Android Manifest Permissions* Click **Add** and pick **Uses Permission** from the list. Under *Attributes for Uses Permission* pick **android.permission.INTERNET** . Using the same steps add the second permission called **android.permission.ACCESS\_FINE\_LOCATION** . Save the file.
5. To run your code in the emulator: right-click project, select *Run As > Android Application*. Android phone emulator window should be opened with application running, just text “Hello World, Hellomap!”. So far so good, but there is yet no map image yet? We’ll add it with the next steps.
6. Add Nutiteq Library JAR to the project. Download from [www.nutiteq.com](http://www.nutiteq.com) >**Developer** > **Downloads** Android library package and unzip it. Copy **android\_maps\_lib-xxx.jar** file to project’s directory in Eclipse package browser, right-click to the file, and select *Build Path > Add to Build Path*. Eclipse should show now the library JAR in “Referenced Libraries” group in project explorer window.
7. First let’s add map to our layout. Open **main.xml** in project’s **res/layout** folder. By default you have **LinearLayout** and a **TextView** there. Remove **TextView** and add **MapView** and **ZoomControls** instead, so it looks as following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<com.nutiteq.android.MapView
    android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    />
</RelativeLayout>
```

8. Now modify your HelloMap.java to have a few extra lines needed for the actual map. New lines are in bold:

```
package com.tutorial;

import com.nutiteq.BasicMapComponent;
import com.nutiteq.android.MapView;
import com.nutiteq.components.WgsPoint;
import com.nutiteq.log.AndroidLogger;
import com.nutiteq.log.Log;
import com.nutiteq.maps.OpenStreetMap;
import com.nutiteq.ui.ThreadDrivenPanning;
import com.nutiteq.wrappers.AppContext;

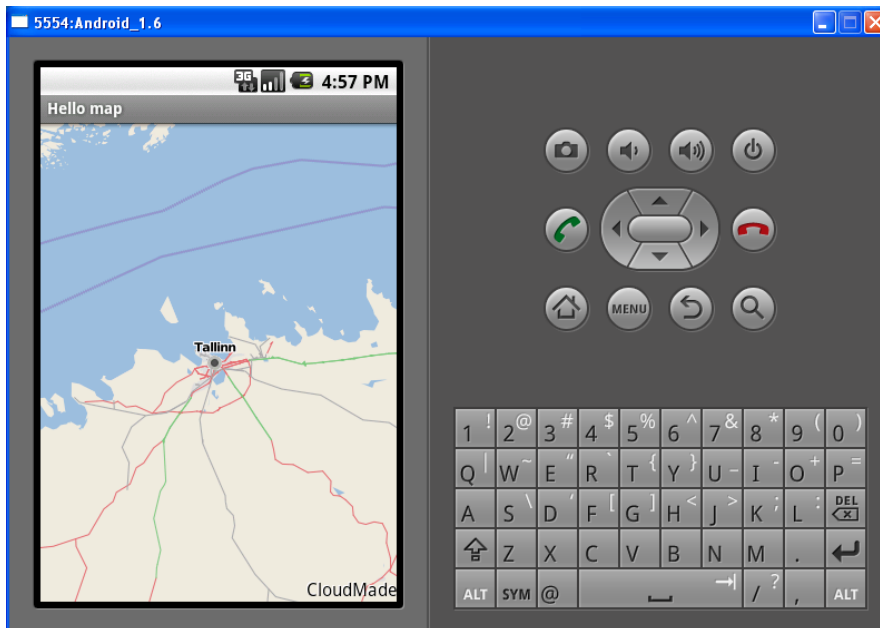
import android.app.Activity;
import android.os.Bundle;

public class HelloMapActivity extends Activity {
    private BasicMapComponent mapComponent;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mapComponent = new BasicMapComponent("tutorial", new AppContext(this), 1, 1, new WgsPoint(24.764580,
59.437420), 10);
        mapComponent.setMap(OpenStreetMap.MAPNIK);
        mapComponent.setPanningStrategy(new ThreadDrivenPanning());
        mapComponent.startMapping();

        // get the mapview that was defined in main.xml
        MapView mapView = (MapView)findViewById(R.id.mapview);
        // mapview requires a mapcomponent
        mapView.setMapComponent(mapComponent);
    }
}
```

9. Let's take a brief look into added code:
  - a. First we have imports, references to mapping library classes which are used and Android RelativeLayout widget where we start displaying our content.
  - b. Then we create **new BasicMapComponent** object. This is the main map object you deal with. The constructor parameters are:
    - i. license key. "tutorial" works if you use default Vendor and Application names like in the tutorial, but for your own application you should generate own key in Nutiteq.com homepage (you have to register first, and then from **Developers** section you'll find **My apps** page).
    - ii. New AppCompatActivity defines context for mapping.
    - iii. Maximum size of the map (width, height). Those parameter values are not relevant for Android MapView (that's why they are set to 1x1 in our example). On Android map size will be immediately reset by View (MapView parameters) and not by mapComponent parameters.
    - iv. initial map center location (24.764580 E, 59.437420 N) as WGS84 decimal coordinates. Btw, these numbers are for Tallinn in Estonia
    - v. initial zoom (10). Zoom = 0 means whole world and maximum zoom for OSM is 18
10. You have to define also panningStrategy, usually just like given in example
11. Define mapComponent for MapView, so it is aware of the object.
12. Now let's run it again. Pick *Run As > Android Application*. You should see following screen. You can pan the map by clicking and dragging with the mouse cursor, and even zoom it in and out with pinch-zoom gestures. But on real phone only – Android emulator does not know multi-touch, and for it you would need some buttons for zoom. Next chapter will be about it.



13. Now you can also run your application in the phone. Connect your device to PC with the USB cable and make sure that USB driver is installed and USB debugging is enabled on the device.

## 4 Zoom Controls

For zoom controls we are using Android built-in ZoomControl widgets.

We need to do two steps to add them:

1. Add them to the layout, so they overlay nicely map

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<com.nutiteq.android.MapView
    android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    />
<ZoomControls
    android:id="@+id/zoomcontrols"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentBottom="true"
    />
</RelativeLayout>
```

2. Define actions for + and – actions of the widgets:

```
import android.view.View;
import android.widget.ZoomControls;
...

public class AndroidMapper extends Activity {

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

        ZoomControls zoomControls = (ZoomControls)findViewById(R.id.zoomcontrols);
        // set zoomcontrols listeners to enable zooming
        zoomControls.setOnZoomInClickListener(new View.OnClickListener() {
            public void onClick(final View v) {
                mapComponent.zoomIn();
            }
        });
        zoomControls.setOnZoomOutClickListener(new View.OnClickListener() {
            public void onClick(final View v) {
                mapComponent.zoomOut();
            }
        });
    });
}
```

Now you should be able to zoom in/out map also.

Note that Also Nutiteq library has also own customizable zoom control buttons, these can be set to left-corner of map.

## 5 Correct way to close the app

You could have noticed, that the program sometimes crashes with an error message during closing or restarting, after you have pressed Back and tried to run the application again. To fix that we have to do some housecleaning. We override the default `onDestroy()` method and we will also use a custom boolean variable `onRetainCalled` to check if we should clean the `mapComponent` or not. Add the following lines to your code (new lines are marked with gray):

```
...
private boolean onRetainCalled;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    onRetainCalled = false;
    ...
}

@Override
public Object onRetainNonConfigurationInstance() {
    onRetainCalled = true;
    return mapComponent;
}
```

```
@Override
protected void onDestroy() {
    super.onDestroy();

    if (!onRetainCalled) {
        mapComponent.stopMapping();
        mapComponent = null;
    }
}
```

## 6 Show GPS location on map

Let's try to pan the map to your current GPS location. There are two options for it:

- a) Use Android Location API, get your location coordinates and add a point (Place, Polygon or Line) on map for that,
- b) Use Nutiteq library built-in GPS positioning support. Following describes the latter approach, as it has special dynamic object for user location display.

1. Import location packages to the source file

```
import com.nutiteq.location.LocationMarker;
import com.nutiteq.location.LocationSource;
import com.nutiteq.location.NutiteqLocationMarker;
import com.nutiteq.location.providers.AndroidGPSPProvider;
```

2. Define location source and define special type of object (LocationMarker) on map, to show GPS location. Note that an image is needed to show the marker. We can reuse bundled Android icon.png for it now. It does not look nice, but for basic proof should be fine

Then insert the following lines, right below the Zoom controls definition.

```
// Zoom controls
...
// GPS Location
final LocationSource locationSource = new AndroidGPSPProvider(
    (LocationManager) getSystemService(Context.LOCATION_SERVICE), 1000L);
Bitmap icon = BitmapFactory.decodeResource(getResources(),
    R.drawable.icon);
final LocationMarker marker = new NutiteqLocationMarker(new PlaceIcon(Image
    .createImage(icon), icon.getWidth()/2, icon.getHeight(), 3000, true);
locationSource.setLocationMarker(marker);
mapComponent.setLocationSource(locationSource);
```

3. Run your changed code. Note that you have to define `android.permission.ACCESS_FINE_LOCATION` permission in AndroidManifest.xml file to get

location information. To get the GPS coordinates inside the emulator you need to switch your Eclipse UI layout to **DDMS** (Dalvik Debug Monitor Server). This layout contains the debugging tools for Android. From here you can send the GPS coordinates to emulator by going under **Emulator Control > Location Controls** and pressing the **Send** button on the Manual tab. The default location is Google Headquarters in California, but you can change coordinates using Longitude and Latitude fields on the Manual tab.

4. On real devices you may be lucky to get your actual GPS coordinates. One thing what you will notice is that first GPS location fix takes time – depending on your GPS, A-GPS service provider, physical location (indoors/outdoors) and last location fix it will take from 30 seconds to several minutes.

## 7 Adding KML data to the map

Google geodata standard file format (KML) data can be read on-line by the Nutiteq mapping library. You may already have a web service which utilizes the format for Google Maps web or Google Earth, now you can just publish it on the mobile. Nutiteq mapping Lib currently supports following KML elements: points, lines and polygons, also style elements are supported, so you can have your custom markers.

[www.nutiteq.com](http://www.nutiteq.com) Developer section has a document describing which KML elements are exactly supported.

1. Import KML package to the Midlet

```
import com.nutiteq.kml.KmlUriReader;
```

2. Add dynamic KML layer to the map, with Panoramio popular images.

```
mapComponent.addKmlService(new KmlUriReader(  
    "http://www.panoramio.com/panoramio.kml?LANG=en_US.utf8&", true));
```

3. Zoom in map – it happened to me that the large GPS image covered all the places

Some notes for creation of compatible KML files:

- Only UTF-8 charset is supported
- KML should not have more than about 50 elements (placemarks). With real phones the memory limit may be quite tight. This is why BBOX and max parameters are used. Number of elements depends also on type of elements: complex polygons take much more memory than points.
- Mobile mapping lib does not support all KML features, e.g. KMZ, *link* elements, *refresh*, more advanced style parameters (e.g. scale of icons) of raster overlays.

## 8 Logging for troubleshooting

1. Add following lines to start showing log information for troubleshooting. This defines also Android logging tag to be used:

```
Log.enableAll();
Log.setLogger(new AndroidLogger("NutiteqTutorial"));
```

2. In DDMS *LogCat* window (shown in IDE, but not in real device) you should now start to see debug log from device or emulator. Notice that the URL request will have also some additional parameters automatically added to the define URL. Smart servers will use at least BBOX (this means Bounding Box of the current map view) parameter, and will return elements only elements inside of given area. Best servers should use also max element (maximum number of expected Placemarks) and may use zoom layering/clustering based on the current zoom parameter. You see something like:

```
Debug > Downloading
http://www.panoramio.com/panoramio.kml?LANG=en_US.utf8&BBOX=24.713058,59.424473,24.816055,59.450659&zoom=14
&max=10
```

3. If you move or zoom the map, you see from debug log that KML request is done again (because we defined parameter `needsUpdateAfterRead=true`). If user moves map a lot, then there can be a lot of on-line re-readings, which means a lot of traffic (which is good if you happen to be mobile operator who charges for it, but it is no good for users). So keep this in mind.

## 9 Full Hello Map listing

Following is final listing of **HelloMap.java**, including all tutorial steps :

```
package com.tutorial;

import com.nutiteq.BasicMapComponent;
import com.nutiteq.android.MapView;
import com.nutiteq.components.PlaceIcon;
import com.nutiteq.components.WgsPoint;
import com.nutiteq.kml.KmlUrlReader;
import com.nutiteq.location.LocationMarker;
import com.nutiteq.location.LocationSource;
import com.nutiteq.location.NutiteqLocationMarker;
import com.nutiteq.location.providers.AndroidGPSPProvider;
import com.nutiteq.log.AndroidLogger;
import com.nutiteq.log.Log;
import com.nutiteq.maps.OpenStreetMap;
import com.nutiteq.ui.ThreadDrivenPanning;
import com.nutiteq.utils.Utils;
import com.nutiteq.wrappers.AppContext;
```



```
import com.nutiteq.wrappers.Image;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.Drawable;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.View;
import android.widget.ZoomControls;

public class HelloMapActivity extends Activity {
    private BasicMapComponent mapComponent;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.enableAll();
        Log.setLogger(new AndroidLogger("myapp"));
        mapComponent = new BasicMapComponent("tutorial", new AppContext(this),
            1, 1, new WgsPoint(24.764580, 59.437420), 10);
        mapComponent.setMap(OpenStreetMap.MAPNIK);
        mapComponent.setPanningStrategy(new ThreadDrivenPanning());
        mapComponent.startMapping();

        // get the mapview that was defined in main.xml
        MapView mapView = (MapView) findViewById(R.id.mapview);

        // mapView requires a mapcomponent
        mapView.setMapComponent(mapComponent);

        ZoomControls zoomControls = (ZoomControls) findViewById(R.id.zoomcontrols);
        // set zoomcontrols listeners to enable zooming
        zoomControls.setOnZoomInClickListener(new View.OnClickListener() {
            public void onClick(final View v) {
                mapComponent.zoomIn();
            }
        });
        zoomControls.setOnZoomOutClickListener(new View.OnClickListener() {
            public void onClick(final View v) {
                mapComponent.zoomOut();
            }
        });

        // GPS Location
        final LocationSource locationSource = new AndroidGPSPProvider(
            (LocationManager) getSystemService(Context.LOCATION_SERVICE), 1000L);
        Bitmap icon = BitmapFactory.decodeResource(getResources(),
            R.drawable.icon);
        final LocationMarker marker = new NutiteqLocationMarker(new PlaceIcon(Image
            .createImage(icon), icon.getWidth()/2, icon.getHeight()/2), 3000, true);
        locationSource.setLocationMarker(marker);
        mapComponent.setLocationSource(locationSource);
    }
}
```



**nutiteq**

```
mapComponent.addKmlService(new KmlUrlReader(  
    "http://www.panoramio.com/panoramio.kml?LANG=en_US.utf8&", true));  
}  
}
```