



**nutiteq**

## **MGMaps BlackBerry RIM mapping lib tutorial**

Version 1.1.1 (updated 24.08.2011)

© 2008-2011 Nutiteq LLC

Nutiteq LLC

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

Skype: nutiteq



## 1 Contents

2	Introduction.....	2
2.1	Document history.....	3
3	Simple Map.....	4
4	Zoom Controls .....	7
4.1	For keyboard.....	7
4.2	For touch-screen.....	7
5	Closing the program .....	7
6	Show GPS location on map.....	8
7	Adding on-line KML file data to the map .....	9
8	Full BlackBerry tutorial listing.....	10

## 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 BlackBerry development, at least how to create and compile basic applications. We are using **Eclipse BlackBerry Plugin** approach in the tutorial, although some code works also with JDE. Only commands for compiling and managing code are somewhat different. There are a lot of good tutorials about BlackBerry development, specifically about Eclipse Plugin see <http://developerlife.com/tutorials/?p=427>

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.

You may already know that you can use also BlackBerry® Maps application within your application, either by opening the application or embedding map there. Nutiteq library is also for embedding maps, so it would be reasonable to compare the solutions following is quick comparison between BlackBerry Maps and Nutiteq Maps Library

	<b>Nutiteq Maps Library SDK</b>	<b>BlackBerry® Maps</b>
<b>Map Content</b>	OpenStreetMap CloudMade Navteq (MapTP) DeCarta DigitalGlobe (satellite and aerial) BLOM Urbex (aerial)	Hard-coded single source

	Bing (Microsoft) Maps Yahoo! Maps WMS API TMS API Ka-Map API <i>Custom map sources</i>	
<b>Map type</b>	Raster, tile-based	Vector
<b>Offline maps, caching</b>	Caching in memory Pre-loaded maps from SD card Pre-loaded maps from application installer package	Caching in memory only
<b>Map navigation</b>	Move, Zoom	Move, Zoom, Rotate
<b>Overlays on map</b>	Points, lines, polygons Raster overlays	No API for embedded maps Markers available with external app only
<b>KML data</b>	Yes	Limited: <ul style="list-style-type: none"> <li>• No API for embedded map</li> <li>• Only points for OS 4.5</li> <li>• Lines and Polygons since OS 5.0</li> </ul>
<b>Routing service</b>	Included, using OSM-based (CloudMade, Yournavigation) or own custom servers based on OpenLS	No
<b>Geocoding (address search) service</b>	Included, using OSM-based (CloudMade, Namefinder) or own custom servers	In LBS API package
<b>License</b>	Free for GPL and developers, license fee for commercial apps. Commercial support available. Source code available for licensees.	Free

## 2.1 Document history

Who	When	Rev	What
JaakL	08.12.2009	1.0.2	First version for BlackBerry, for lib version 1.0.2

JaakL	24.08.2011	1.1.1	Update for SDK 1.1.1 release
-------	------------	-------	------------------------------

### 3 Simple Map

Following are steps for your “Hello world” type of BlackBerry® interactive mapping application called *Hello map*. We assume that you have installed **BlackBerry Java Plug-in for Eclipse 1.3.0** with at least one BlackBerry SDK version.

1. Create new BlackBerry project: in Eclipse, select File > New project > BlackBerry, BlackBerry Project. Let’s call it *hello\_map*.
2. Try run default application code in the emulator: right-click project, select **Run As > BlackBerry Simulator** or **Debug as > BlackBerry Device**. BlackBerry phone emulator window should be opened. To start your program go under **Downloads > Hello\_map** (on the emulator or device). Your program should start and show MyTitle text.
3. Now we add mapping to it. First add Nutiteq Library JAR to the project. Download from [www.nutiteq.com](http://www.nutiteq.com) >Developer > Downloads **BlackBerry** package version and unzip it. Copy *rimui\_maps\_lib-xxx.jar* file to project’s directory, right-click to the file, and select *Build Path > Add to Build Path*. Note: if you create build for **BlackBerry SDK 6.0** or newer then take *rimui6\_maps\_lib-xxx.jar*. Eclipse should show now the library JAR in “Referenced Libraries” group in project explorer window.
4. One more click is needed here to ensure that necessary mapping JAR code will be also in your application JAR file: right-click project, select Build Path > Configure Build Path; select now tab “Order and Export”; here *rimui\_maps\_lib-1.1.1.jar* should be listed (probably as last item), make it selected and click OK.
5. Copy **MapFieldTouch.java** file to your **mypackage** package in the project. You can find from <http://www.nutiteq.com/rim-blackberry-mapping-api-sdk> . This provides easy high-level UI element to be used in BlackBerry screen manager. It may need internal tweaks for different BlackBerry versions, so it is not bundled with SDK itself yet.
6. Now modify your **MyScreen.java** to have a few extra lines needed for the actual map. New lines are highlighted:

```
package mypackage;

import com.nutiteq.BasicMapComponent;
import com.nutiteq.cache.Cache;
import com.nutiteq.cache.CachingChain;
import com.nutiteq.cache.MemoryCache;
import com.nutiteq.components.WgsPoint;
import com.nutiteq.log.Log;
import com.nutiteq.maps.OpenStreetMap;
import com.nutiteq.net.DefaultDownloadStreamOpener;
import com.nutiteq.ui.ThreadDrivenPanning;
import com.nutiteq.wrappers.AppContext;

import net.rim.device.api.ui.component.SeparatorField;
```

```

import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

/**
 * A class extending the MainScreen class, which provides default standard
 * behavior for BlackBerry GUI applications.
 */
public final class MyScreen extends MainScreen {
    private VerticalFieldManager _fieldManagerMiddle;
    private MapFieldTouch map;
    private BasicMapComponent mapComponent;

    /**
     * Creates a new MyScreen object
     */
    public MyScreen() {
        super(NO_VERTICAL_SCROLL);
        // Set the displayed title of the screen
        setTitle("MyTitle");

        Log.enableAll();

        mapComponent = new BasicMapComponent("tutorial", new ApplicationContext(this), 1,
        1, new WgsPoint(-71.023865,42.416867),12);

        mapComponent.setMap(OpenStreetMap.MAPNIK);

        final MemoryCache memoryCache = new MemoryCache(10 * 1024 * 1024);
        mapComponent.setNetworkCache(new CachingChain(new Cache[] {
            memoryCache});

        mapComponent.setPanningStrategy(new ThreadDrivenPanning());

        mapComponent.setDownloadStreamOpener(new DefaultDownloadStreamOpener(
            ";deviceside=true;interface=wifi"));

        mapComponent.startMapping();

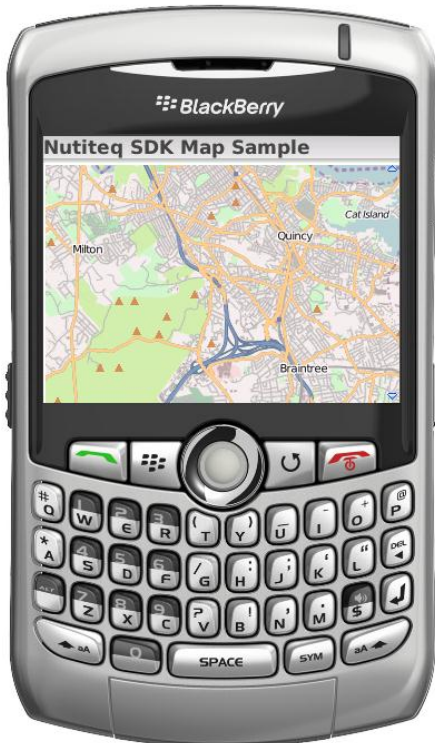
        map = new MapFieldTouch(mapComponent);
        add(map);
    }
    // rotate screen properly
    protected void sublayout(int width, int height) {
        super.sublayout(width, height);
        Log.debug("subLayout "+width+" "+height);
        if(map != null){
            mapComponent.resize(width,height);
        }
    }
};
}

```

7. Let's take now a brief look into added code:

- a. First we have imports, and fields what we will use later.
- b. `super(NO_VERTICAL_SCROLL);` is needed to avoid creation of really very long screen, we limit it to just visible area

- c. **Log.enableAll();** is useful for getting Nutiteq SDK logging messages to console log
  - d. We create **BasicMapComponent** object giving license key and initial size and geographical location as arguments. Initial size will be rewritten before you even see map, by **sublayout()** method below
  - e. We define map to be used – default **OpenStreetMap** view
  - f. We define memory-based cache and panning strategy. You can tweak cache size here, based on your target device capabilities
  - g. **mapComponent.setDownloadStreamOpener** is needed for BlackBerry connections – you set URL ending there. It can be tricky to find working string, and probably you just cannot use one static string. Given string works usually fine for wifi connections. See Nutiteq BlackBerry sample applications to get more powerful code to set download opener.
  - h. **startMapping()** is needed to start real mapping
  - i. **MapFieldTouch** – a Field with map is created, and **MapComponent** is given as argument.
  - j. The field is added to the screen
  - k. Finally, you need re-definition of **sublayout()** method to set correct size for map, based on screen size. This enables dynamic re-sizing of map when user rotates device. Otherwise, you can also define fixed size for map.
8. Now let's run it again. Pick **Run As > BlackBerry Simulator**. You should see following screen. You can pan the map by clicking and dragging with the mouse cursor. You cannot zoom in/out the map, as the definition of Zoom control keys comes next.



## 4 Zoom Controls

### 4.1 For keyboard

To add zoom control keys to the map we need to define two control keys – “O” for zooming out and “I” for zooming in (it requires qwerty keyboard). Insert the following lines into your code (before creating a new MapField object):

```
...  
import com.nutiteq.controls.UserDefinedKeysMapping;  
...  
    final UserDefinedKeysMapping keysMapping = new UserDefinedKeysMapping();  
        keysMapping.defineKey(ControlKeys.ZOOM_OUT_KEY, 79);  
        keysMapping.defineKey(ControlKeys.ZOOM_IN_KEY, 73);
```

And then add those keys to map element (right before `map.startMapping()` line) :

```
mapComponent.setControlKeysHandler(keysMapping);
```

### 4.2 For touch-screen

For touch-screen device you can add visual buttons on map where clicking zooms map. The code for it would be following:

```
if (Touchscreen.isSupported()) {  
    mapComponent.setOnScreenZoomControls(new OnScreenZoomControls(Utils  
        .createImage("/m-l-controls.png")));  
} else {  
    mapComponent.setCursor(new DefaultCursor(0xFFFF0000));  
}
```

The `m-l-controls.png` file is used for button images. This specific file is already bundled with Nutiteq SDK, but you can use own also.

## 5 Closing the program

We have to stop mapping before we close the program. It is needed for example if you use RMS cache, as its index will be written only during this call. Add following methods to your program:

```
protected void makeMenu(Menu menu, int instance) {  
    menu.add(_close);  
}  
  
private MenuItem _close = new MenuItem("Close", 110, 10) {  
    public void run() {  
        onClose();  
    }  
};  
  
public boolean onClose() {  
    mapComponent.stopMapping();
```

```
System.exit(0);  
return true;  
}
```

## 6 Show GPS location on map

Map center location is also defined with WGS84 latitude and longitude coordinates. Let's try to pan the map to your current GPS location, assuming that the phone has internal GPS

There are several ways how to get user location via Nutiteq SDK:

- a) JSR-179 API support, see below
- b) External Bluetooth GPS
- c) Cell-ID positioning (requires using third party Cell\_ID database, e.g. OpenCellId)

### 1. Import location packages to the source file

```
import com.nutiteq.location.LocationMarker;  
import com.nutiteq.location.LocationSource;  
import com.nutiteq.location.providers.LocationAPIProvider;  
import com.nutiteq.location.NutiteqLocationMarker;  
import com.nutiteq.components.PlaceIcon;
```

2. Define location source and define special type of object (LocationMarker) on map, to show GPS location. We reuse one SDK bundled image def\_kml as location marker, but you can have own in the application.

3. Then insert the following lines right below the map.startMapping(); line.

```
// Map  
...  
// GPS Location  
  
if (System.getProperty("microedition.location.version") != null) {  
    final LocationSource dataSource = new LocationAPIProvider(3000);  
    try {  
        final Image gpsPresentImage = Image.createImage("/def_kml.png");  
        final Image gpsConnectionLost = Image  
            .createImage("/def_kml.png");  
        final LocationMarker marker = new NutiteqLocationMarker(  
            new PlaceIcon(gpsPresentImage, 4, 16), new PlaceIcon(  
                gpsConnectionLost, 4, 16), 3000, true);  
  
        dataSource.setLocationMarker(marker);  
        mapComponent.setLocationSource(dataSource);  
    } catch (final IOException e) {  
    }  
}
```

4. Run your changed code. If you use BlackBerry simulator, make sure it has GPS emulation support (only some of them have it). With device make sure it has good satellite/GPS reception.

## 7 Adding on-line KML file data to the map

KML data can be read on-line by the mapping library. Mapping Lib currently supports following KML elements: points, lines and polygons, also style elements are supported.

1. Import KML package to the Application

```
import com.nutiteq.kml.*;
```

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

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

3. In the debug console window (shown in IDE, but not in real device) you should see debug log, if you are starting application in Debug mode. Notice that the URL request will have also some additional parameters automatically added to the define URL. Smart servers will use at least BBOX parameter, and will not return elements which are outside of defined area. Best servers should use also max element (maximum number of expected Placemarks) and may use zoom layering/clustering based on zoom parameter.

```
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
```

4. 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.

Some notes for creation of compatible KML files:

- Only UTF-8 charset is supported
- KML should not have more than about 20 elements (placemarks). It works fine with more if you use WTK emulator (which has a lot of memory), but with real phones the 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, time-based refreshes, certain more advanced style parameters (e.g. scale of icons). We plan to add these in



next releases, possibly also enhance rendering of more complex elements and styles. Also client-side caching will be enhanced further (currently only icon images are cached in RMS).

## 8 Full BlackBerry tutorial listing

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

```
package mypackage;

import java.io.IOException;

import com.nutiteq.BasicMapComponent;
import com.nutiteq.cache.Cache;
import com.nutiteq.cache.CachingChain;
import com.nutiteq.cache.MemoryCache;
import com.nutiteq.components.PlaceIcon;
import com.nutiteq.components.WgsPoint;
import com.nutiteq.controls.ControlKeys;
import com.nutiteq.controls.OnScreenZoomControls;
import com.nutiteq.controls.UserDefinedKeysMapping;
import com.nutiteq.location.LocationMarker;
import com.nutiteq.location.LocationSource;
import com.nutiteq.location.NutiteqLocationMarker;
import com.nutiteq.location.providers.LocationAPIProvider;
import com.nutiteq.log.Log;
import com.nutiteq.maps.OpenStreetMap;
import com.nutiteq.net.DefaultDownloadStreamOpener;
import com.nutiteq.ui.DefaultCursor;
import com.nutiteq.ui.ThreadDrivenPanning;
import com.nutiteq.utils.Utils;
import com.nutiteq.wrappers.AppContext;
import com.nutiteq.wrappers.Image;

import net.rim.device.api.ui.Touchscreen;
import net.rim.device.api.ui.component.SeparatorField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

/**
 * A class extending the MainScreen class, which provides default standard
 * behavior for BlackBerry GUI applications.
 */
public final class MyScreen extends MainScreen {
    private VerticalFieldManager _fieldManagerMiddle;
    private MapFieldTouch map;
    private BasicMapComponent mapComponent;

    /**
     * Creates a new MyScreen object
     */
    public MyScreen() {
        super(NO_VERTICAL_SCROLL);
        // Set the displayed title of the screen
        setTitle("MyTitle");
    }
}
```

```

    Log.enableAll();

    mapComponent = new BasicMapComponent("tutorial", new AppCompatActivity(this), 320,
    240, new WgsPoint(-71.023865,42.416867),12);
    mapComponent.setMap(OpenStreetMap.MAPNIK);
    final MemoryCache memoryCache = new MemoryCache(10 * 1024 * 1024);
    mapComponent.setNetworkCache(new CachingChain(new Cache[] {
        memoryCache}));
    mapComponent.setPanningStrategy(new ThreadDrivenPanning());

    mapComponent.setDownloadStreamOpener(new DefaultDownloadStreamOpener(
        ";deviceside=true;interface=wifi"));

    final UserDefinedKeysMapping keysMapping = new UserDefinedKeysMapping();
    keysMapping.defineKey(ControlKeys.ZOOM_OUT_KEY, 79);
    keysMapping.defineKey(ControlKeys.ZOOM_IN_KEY, 73);

    mapComponent.setControlKeysHandler(keysMapping);

    if (Touchscreen.isSupported()) {
        mapComponent.setOnScreenZoomControls(new OnScreenZoomControls(Utils
            .createImage("/m-l-controlls.png")));
    } else {
        mapComponent.setCursor(new DefaultCursor(0xFFFF0000));
    }

    // GPS Location

    if (System.getProperty("microedition.location.version") != null) {
        final LocationSource dataSource = new LocationAPIProvider(3000);
        try {
            final Image gpsPresentImage = Image.createImage("/def_kml.png");
            final Image gpsConnectionLost = Image
                .createImage("/def_kml.png");
            final LocationMarker marker = new NutiteqLocationMarker(
                new PlaceIcon(gpsPresentImage, 4, 16), new PlaceIcon(
                    gpsConnectionLost, 4, 16), 3000, true);
            dataSource.setLocationMarker(marker);
            mapComponent.setLocationSource(dataSource);
        } catch (final IOException e) {
        }
    }

    mapComponent.startMapping();

    map = new MapFieldTouch(mapComponent);
    add(map);

    }
    // rotate screen properly
    protected void sublayout(int width, int height) {
        super.sublayout(width, height);
        Log.debug("subLayout "+width+" "+height);
        if(map != null){
            mapComponent.resize(width,height);
        }
    }

```



**nutiteq**

```
};  
}
```