



## Nutiteq RIM BlackBerry mapping SDK tutorial

Version 1.0.2 (updated 11.12.2009)

© 2008-2009 Nutiteq LLC

**Nutiteq LLC**

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

Skype: nutiteq

Fix: (+372) 712 2334

Mob: +372) 509 2586

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	Closing the program.....	8
6	Show GPS location on map.....	9
7	Adding on-line KML file data to the map.....	10
8	Full Hello Map listing.....	11



## 2 Introduction

Nutiteq mobile mapping library is designed to enable adding custom interactive maps and related LBS features (routing, address search, positioning) to the mobile applications.

To get started, we assume that you know some basics of BlackBerry development, at least how to compile basic applications. We are using **Eclipse BlackBerry Plugin** approach in the tutorial, although exactly the same code snippets work also with JDE or anyother IDE. 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.

### 2.1 Comparison with BlackBerry® Maps

You may already know that you can embed also BlackBerry® Maps application within your application, or opening the Maps Application using application link. Nutiteq library is also for embedding maps, so it make sense to compare quickly 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) Bing (Microsoft) Maps Yahoo! Maps WMS API TMS API Ka-Map API <i>Custom map sources</i>	Hard-coded, single source
<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 in OS 5.0 only</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	No
<b>License</b>	Free for GPL and developers, license fee for commercial apps. Commercial support available. Also source code available for licensees and contributors.	Free

## 2.2 Document history

Who	When	Rev	What
JaakL	17.12.2009	1.0.2	First version for BlackBerry, for lib version 1.0.2.

### 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 SDK 4.5.0 Plugin, Eclipse IDE (we used here 3.4.2) with compatible BlackBerry JDE Plug-in for Eclipse (current default version is 4.5).

1. Create new BlackBerry project: in Eclipse, select File > New project > BlackBerry, BlackBerry Project. Let’s call it **hello\_map**.
2. Create very minimal Application code. First create package, I call it **com.tutorial**. Into the package new file called **HelloMap.java** . You can copy-paste following code into this:

```
package com.tutorial;

import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.container.MainScreen;

public class HelloMap extends UiApplication {
    public static void main(String[] args) {
        HelloMap theApp = new HelloMap();
        theApp.enterEventDispatcher();
    }

    public HelloMap() {
        pushScreen(new UserInterfaceScreen());
    }
}

final class UserInterfaceScreen extends MainScreen {
    //test
    public UserInterfaceScreen() {
        super();

        LabelField title = new LabelField("Nutiteq SDK Map Sample",
            LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
    }
}
```

3. Let’s try if it compiles properly. Right-click on your project and make sure that **Activate for BlackBerry** is checked. Then from the Eclipse menu pick **Project > Build Active BlackBerry Configuration**.
4. To run your code in the emulator: right-click project, select **Run As > BlackBerry Simulator**. BlackBerry phone emulator window should be opened. To start your program go under Applications > Hello\_map (on the emulator). Your program should start with white screen and just a title text “Nutiteq SDK Map Sample”. So far so good, but there is no map image yet? We’ll add it with the next steps.

5. Add Nutiteq Library JAR to the project. Download from [www.nutiteq.com](http://www.nutiteq.com) >Developer > Downloads **RIM library** package 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*. Eclipse should show now the library JAR in "Referenced Libraries" group in project explorer window.
6. 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-1.0.2.jar* should be listed (probably as last item), make it selected and click OK.
7. Copy **MapField.java** file located under uncompressed Nutiteq RIM Library package sample directory, in *src\com\nutiteq\fieldmanager* to your project. This provides easy high-level UI element to be used in BlackBerry screen manager.
8. Now modify your HelloMap.java to have a few extra lines needed for the actual map. New lines are highlighted:

```

package com.tutorial;

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

import com.nutiteq.components.WgsPoint;
import com.nutiteq.net.DefaultDownloadStreamOpener;

public class HelloMap extends UiApplication {
    public static void main(String[] args) {
        HelloMap theApp = new HelloMap();
        theApp.enterEventDispatcher();
    }

    public HelloMap() {
        pushScreen(new UserInterfaceScreen());
    }
}

final class UserInterfaceScreen extends MainScreen {

    VerticalFieldManager _fieldManagerMiddle;
    private MapField map;

    public UserInterfaceScreen() {
        super();

        LabelField title = new LabelField("Nutiteq SDK Map Sample",
            LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);

        _fieldManagerMiddle = new VerticalFieldManager();
        add(new SeparatorField());
        add(_fieldManagerMiddle);
    }
}

```

```
map = new MapField("Map", "tutorial", "Nutiteq", "proovin", 320, 240, new WgsPoint(-71.023865,42.416867),12);
_fieldManagerMiddle.add(map);

map.setDownloadStreamOpener(new DefaultDownloadStreamOpener(";deviceside=true"));
map.startMapping();
}
}
```

9. Let's take a brief look into added code:
- First we have imports, references to mapping library classes which are used and BlackBerry VerticalFieldManager widget where we start displaying our content.
  - Then we define *MapField* type private variable called *map* (this is UI custom element for mapping).
  - Then we create a new VerticalFieldManager .
  - Then we create new MapField . The parameters are:
    - Label.
    - 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 (Developers section).
    - Vendor name used by library for license check.
    - Application name used by library for license check.
    - Maximum size of the element (width, height).
    - initial map center location.
    - initial zoom (12). Zoom = 0 means whole world and maximum zoom for OSM is 18.
  - Then we add our newly created map element to VerticalFieldManager in order to get it to display on screen.
  - Then we must use *setDownloadStreamOpener()* to create connection to our downloaded resources. This specifically needed for BlackBerry applications. It is important that your *deviceside=* parameter value is appropriate for your device networking configuration.
  - Finally, we start mapping process by using *startMapping()*.
10. 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

To add zoom control keys to the map we need to define two control keys – “O” for zooming out and “I” for zooming in. 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) :

```
map.setControlKeysHandler(keysMapping);
```

## 5 Closing the program

We have to stop mapping before we close the program. It is needed because the RMS cash index gets written 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() {
    map.stopMapping();
    Dialog.alert("Goodbye!");
    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.Placelcon;
```

2. Define location source and define special type of object (LocationMarker) on map, to show GPS location. Note that two images are needed to show the marker – one when the connection is established and one when there is no connection with GPS. We use .PNG images with resolution of 10x17 pixels. You need to copy your images under **res/yourGPSimage.png** and **res/yourGPSimage\_connection\_lost.png**

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("/gps_marker.png");
        final Image gpsConnectionLost = Image.createImage("/gps_connection_lost.png");
```

```
final LocationMarker marker = new NutiteqLocationMarker(new PlaceIcon(gpsPresentImage, 4, 16), new
PlaceIcon(gpsConnectionLost, 4, 16), 3000, true); dataSource.setLocationMarker(marker);
    map.setLocationSource(dataSource);
}
catch (final IOException e) { }
```

3. Run your changed code. Note that you have to use BlackBerry simulator that has GPS support.

## 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. Enable debug log for MGMaps lib by adding `Log.enableAll()`; somewhere before creating a new mapfield.

```
import com.nutiteq.log.Log;
.....
Log.enableAll();
```

4. 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
```

5. 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 Hello Map listing

Following is final listing of **HelloMap.java**, including all tutorial steps (NB! **MapField.java** listing is not given here, this file should be taken from Nutiteq RIM library samples directory):

```
package com.tutorial;

import java.io.IOException;

import net.rim.device.api.ui.MenuItem;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.Menu;
import net.rim.device.api.ui.component.SeparatorField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

import com.nutiteq.components.WgsPoint;
import com.nutiteq.controls.ControlKeys;
import com.nutiteq.controls.UserDefinedKeysMapping;
import com.nutiteq.net.DefaultDownloadStreamOpener;
import com.nutiteq.wrappers.rimui.Image;
import com.nutiteq.kml.*;
import com.nutiteq.log.Log;
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;

public class HelloMap extends UiApplication {
    public static void main(String[] args) {
        HelloMap theApp = new HelloMap();
        theApp.enterEventDispatcher();
    }
}
```

```

public HelloMap() {
    pushScreen(new UserInterfaceScreen());
}
}

final class UserInterfaceScreen extends MainScreen {

    VerticalFieldManager _fieldManagerMiddle;
    private MapField map;

public UserInterfaceScreen() {
    super();

    LabelField title = new LabelField("Nutiteq SDK Map Sample",
        LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
    setTitle(title);

    _fieldManagerMiddle = new VerticalFieldManager();
    add(new SeparatorField());
    add(_fieldManagerMiddle);

    //Keys for zooming in and out
    final UserDefinedKeysMapping keysMapping = new UserDefinedKeysMapping();
        keysMapping.defineKey(ControlKeys.ZOOM_OUT_KEY, 79);
        keysMapping.defineKey(ControlKeys.ZOOM_IN_KEY, 73);

    Log.enableAll();
    map = new MapField("Map", "tutorial", "Nutiteq", "proovin", 320, 240, new WgsPoint(-71.023865,42.416867),12);
    _fieldManagerMiddle.add(map);

    map.setDownloadStreamOpener(new DefaultDownloadStreamOpener(";deviceside=true"));

    map.setControlKeysHandler(keysMapping);
    map.startMapping();

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

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

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() {
    map.stopMapping();
    Dialog.alert("Goodbye!");
    System.exit(0);
    return true;
}
}
```