



nutiteq

MGMaps J2ME mapping lib tutorial

Version 1.0.2 (updated 19.11.2009)

© 2008-2009 Nutiteq LLC

Nutiteq LLC

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.....	3
3	Simple Map.....	4
4	Control keys.....	7
5	Map Controls.....	8
6	Map Markers.....	8
7	Show GPS location on map.....	10
8	Basic Events.....	11
9	Adding on-line KML file data to the map.....	12
10	Full Hello Map listing.....	13
11	Next steps: using mapComponent API.....	17

2 Introduction

The J2ME mapping lib is designed to enable adding slippy maps to mobile java (J2ME) application as easily as possible. However, we assume that you know some basics of J2ME, at least how to compile basic midlets; there are a lot of good tutorials (e.g. from Sun <http://developers.sun.com/mobility/midp/articles/wtoolkit/>) and books available about this.

Depending on level of required interaction, there are two possible map component options:

- a) **MapItem** is a user interface Custom item. It gives easy way to have interactive maps to midlet's screen Forms, but it has no low-level control, which is noticeable for some keypresses (joystick keys), also you can control screen graphics in the limits of Forms. Internally, mapItem uses mapComponent.
- b) **MapComponent** enables to make interactive map to Canvas. This is also quite easy to use, and you have full low-level access to input and output, e.g. for enabling drawing own custom stuff on top of map, receiving all keypressing events etc.. It extends basicMapComponent, and defines some defaults to it: defines OSM as default map, caching, panning strategy.
- c) **BasicMapComponent** is the lowest level of map component interaction, it does not even define any default map.

Following tutorial includes only most basic functions and examples of the library, full library has much more features and options, see MGMaps Lib Developer guide and Javadocs to have full overview of functionality.

Note that MGMaps mapping library does not handle other necessary, but not map related tasks for typical J2ME application, for example nice user interface (texts, menus, additional graphics, i18n, L10n), database/RMS connectivity, device-specific customizations etc. The library is focused to the mobile mapping part, still it is designed to be as flexible as possible to enable to have customized and tailored logics, look and feel. Also tutorial has no additional logics (e.g. thread management, menu etc) which would be needed for a real application.

2.1 Document history

Who	When	Rev	What
JaakL	12.08.2008	0.2.2	First version, for lib version 0.2.2.
JaakL	17.10.2008	0.6.0	Updated full sample listing, added log viewer screen for easier debugging. Changed positioning from JSR-179 to API of library.
Jaakl	26.01.2009	0.8.0	Updated for 0.8.0 lib
Jaakl	02.02.2009	0.8.1 A	Changed code to fix Form repaint issue with Nokia S60
Jaakl	01.07.2009	1.0.0	Updated for 1.0.0 version of the Library
Jaakl	19.11.2009	1.0.2	Updated for 1.0.2 release, changes in default cursor and zoom controls calls, replace deprecated PlaceListener

3 Simple Map

Following are steps for your “Hello world” type of J2ME interactive mapping application called *Hello map*. We assume that you have installed Sun WTK 2.5.2 , Eclipse IDE (I use 3.3.2) with compatible EclipseME plug-in.

1. Create new Midlet Suite project: in Eclipse, select File > New project, J2ME Midlet Suite. Let’s call it **hello_map**. This creates necessary files and directories, also links to J2ME standard libraries
2. Create very minimal Midlet 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 javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class HelloMap
    extends MIDlet
    implements CommandListener {
    private Form mMainForm;

    public HelloMap() {
        mMainForm = new Form("Hello map");
    }

    public void startApp() {
        mMainForm.append(new StringItem(null, "Hello, map!\n"));
        mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
        mMainForm.setCommandListener(this);
        Display.getDisplay(this).setCurrent(mMainForm);
    }

    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {}

    public void commandAction(Command c, Displayable s) {
        notifyDestroyed();
    }
}
```

3. Save the java file.
4. Now configure JAD file. Open **hello_map.jad** file, which should be found in root directory of your project. It should be opened by default in graphical *Application description* editor (of course this is for sheep and brave ones will use plain text editor like vi to do this). Only mandatory thing to

be defined here is Midlet. Open second sheet in the editor called *Midlets* (yes, the editor has several sheets), click button **Add** and modify contents of new row: Name: **HelloMap** and select cell *Class*. Now selector icon should appear in the right part of the cell, click on this. This opens "Choose Midlet" dialog with pre-filled * character. This * means maybe wildcard in theory, but at least for me it never shows anything, so I delete it and right after typing *He*, magically HelloMap appears as the right choice. Select it with doubleclick. Now midlet class *com.tutorial>HelloMap* should be defined. Save the JAD file.

5. Now let's try if it compiles properly. Right-click on project, select J2ME > Create Package. JAR and JAD file should be now generated into build directory.
6. Run your Midlet in the emulator: right-click project, select *Run As... > Open Run Dialog*. Select *Wireless Toolkit Emulator* and click "New" to create new configuration. As Executable it is better to select "Midlet" (instead of Over the Air), and as Midlet select *com.tutorial>HelloMap*. Now click "Run". WTK phone emulator window should be opened with application running, just text "Hello, map!". So far so good, but there is yet no map image yet? We'll add it with the next steps.
7. Add MGMaps Library JAR to the project. Copy *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. 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 *maps_lib-xxx.jar* should be listed (probably as last item), make it selected and click OK.
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 javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import com.nutiteq.MapItem;
import com.nutiteq.maps.*;
import com.nutiteq.components.WgsPoint;
import com.nutiteq.controls.ControlKeys;

public class HelloMap
    extends MIDlet
    implements CommandListener {

    private Form mMainForm;
    private MapItem mapItem;

    public HelloMap() {
        mapItem = new MapItem("Map", "tutorial", this, 300, 150, new WgsPoint(24.764580, 59.437420), 12);

        mMainForm = new Form("Hello map");
    }
}
```

```
public void startApp() {
    mMainForm.append(new StringItem(null, "Hello, map!\n"));
    mMainForm.append(mapItem);

    mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
    mMainForm.setCommandListener(this);

    Display.getDisplay(this).setCurrent(mMainForm);
    mapItem.startMapping();
}

public void pauseApp() {}

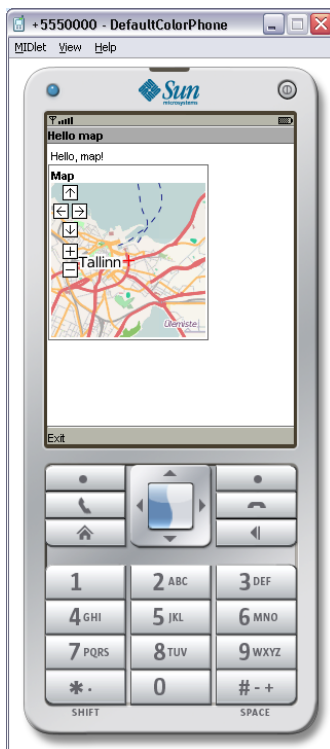
public void destroyApp(boolean unconditional) {}

public void commandAction(Command c, Displayable s) {
    notifyDestroyed();
}
}
```

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

- a. First we have imports, references to mapping library classes which are used
- b. Then we define mapItem local variable
- c. Then we create MapItem (which is Custom Form element). The parameters are:
 - i. name of Form element (shown as text label)
 - ii. 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).
 - iii. reference to the midlet object (*this*)
 - iv. maximum size of element (300x150 pixels). Depending on real device the form element may be smaller (but not smaller than 100x100 pixels). Typical mobiles will limit map image width more or less equal to the screen width, so I put 300 here just to ensure that it will be "full screen width" (for most devices), actual width would be 232 for the WTK default emulator, for instance. You can also define width=1000 to be sure that it really fills every possible screen (but then some phone like Nokia 6500 will surprise with a bug so the item is just 1000 pixels wide). Height of the Forms is not limited by screen, so typical phone will show vertical scrolling bar if map is too tall. With additional code or device-specific preprocessing it is also possible to find out actual size of screen (or Canvas); this is not covered here, but in real application I would suggest to resize map item according to particular screen size.
 - v. initial map center location (24.764580 E, 59.437420 N), this is Tallinn in Estonia
 - vi. initial zoom (12). Zoom = 0 means whole world and maximum zoom for OSM is 18.

- d. Then we just add the mapItem to the screen form.
 - e. Finally, after all the preparations, we start mapping process, this initiates downloading queue. Note that should be called in startApp class, otherwise it does not work in some real devices.
10. Now remember to build again the J2ME package, and then run it. You should see following screen. You cannot zoom in/out or pan the map, as the definition of control keys comes next.



11. You can copy your application also to the phone, the installation package is at project's *deployed* directory. For most phones (Nokia, SonyEricsson et least) the easiest way is to copy just JAR file to the phone using bluetooth, and then open file and phone will prompt for installation. Some phones may require also JAD file and over-the-air (i.e. over WAP/mobile data) installation.

4 Control keys

You will probably notice that you cannot pan the map using arrow/joystick buttons. The problem is that the arrows are just moving selection focus between form elements (Hello text and map), and map itself does not get any information about these keypresses. Unfortunately, this is how all real phones also do (at least most of them), so we should define some other keys to enable moving map. At least if you really need movable map. For example, you can define "numeric joystick" using following lines (add these before appending mapItem to the form), and define * and # as zooming keys:

```
mapItem.defineControlKey(ControlKeys.MOVE_UP_KEY, Canvas.KEY_NUM2);
```

```
mapItem.defineControlKey(ControlKeys.MOVE_DOWN_KEY, Canvas.KEY_NUM8);
mapItem.defineControlKey(ControlKeys.MOVE_LEFT_KEY, Canvas.KEY_NUM4);
mapItem.defineControlKey(ControlKeys.MOVE_RIGHT_KEY, Canvas.KEY_NUM6);
mapItem.defineControlKey(ControlKeys.ZOOM_IN_KEY, Canvas.KEY_POUND);
mapItem.defineControlKey(ControlKeys.ZOOM_OUT_KEY, Canvas.KEY_STAR);
```

Note that with most phones these keys will still work only after the map item is selected first. This is not so user-friendly for dedicated mapping application, and we suggest to use low-level mapping API (MapComponent) for that, but should be ok for just showing a few locations on map.

5 Touch Map Controls

MapItem supports also on-screen map controls. However, these can be used only with a phone has pointing stick and touch-screen, so it would be wise to check first whether phone has any use of it.

1. Create dummy canvas, just to be able to check whether phone has pointer events available

```
// add on-screen controls, if phone has pointer
final Canvas canvas = new Canvas() {
    protected void paint(Graphics arg0) {
    }
};
final boolean pointer = canvas.hasPointerEvents();
```

2. If phone has pointer, then show map controls on map, if no then do not show

```
if (pointer) {
    mapItem.setOnScreenZoomControls(new OnScreenZoomControls(Utils
        .createImage(OnScreenZoomControls.DEFAULT_ZOOM_IMAGE)));
} else {
    mapItem.setCursor(new DefaultCursor(0xFFFF0000));
}
```

If you have pointing stick and touch-screen phone (e.g. one of the Sony-Ericsson Symbian phones) then you can drag map to move map now. Note that Sun WTK emulator settings must to be re-configured to support the pointer, by default it is not turned on.

6 Map Markers

Now you have map, but want also add some points to map, typically to mark some location. For this you first need geographical coordinates (latitude and longitude) of the location. Application can get the coordinates from different places: read from on-line service (e.g. geocoder), read from a local resource file, or even ask from the user. Important part is that you must have the coordinates, in the most

common WGS84 coordinate system, same numbers like you possibly have in a web app, and what are used in e.g. KML files.

1. Import definition of Place (this is the item on map) and define Image variable called *icon*.

```
import com.nutiteq.components.Place;
// following should be in class startApp
private Image icon;
```

2. You have to create icon for the marker and read image into that. Currently there are no default markers available. For this you should copy a small PNG image (e.g. 16x16 pixels, just like the one what you use in web as custom marker) to the **res** directory of your project. I have here image "nutiteq.png". Then add following lines to startApp class. Note that the image reading must be made in the try-catch block:

```
try {
    icon = Image.createImage("/nutiteq.png");
} catch (IOException e) {
}
```

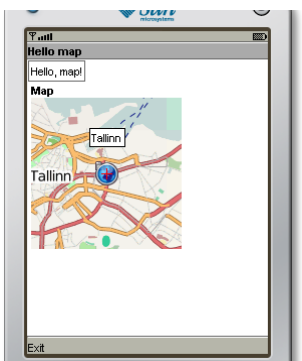
3. Finally add place to the map, with the icon and defined label.

```
mapItem.addPlace(new Place(1, "Tallinn", icon, 24.764580, 59.437420));
```

4. Parameters of the Place are:

- a. Id of the place (1). This is used to identify specific place, you should put unique number here.
- b. Label item of the place
- c. Marker image for the place
- d. Longitude and Latitude of the place.

Result should be like following (marker picture depends on what image do you use):



5. Finally, let's draw also a line to the map. We create first local object *line*; then array *linePoints* with some points (should be *WgsPoint*-s) inside it; then we add the *linePoints* to *line*, and finally

add line to the map Item. Note that the line is just a graphical line, it is not a Place, therefore you cannot get “onmouseover”/selection events for that.

```
WgsPoint[] linePoints={
    new WgsPoint(24.76382468302337,59.44325151314919),
    new WgsPoint(24.76344295658494,59.4462352840583),
    new WgsPoint(24.76593650384734,59.44530921763007),
    new WgsPoint(24.76804665483925,59.44616268729941),
    new WgsPoint(24.76810500478219,59.443291656657)};

Line line = new Line(linePoints, new LineStyle(0xFF0000, 5));
mapItem.addLine(line);
```

7 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, or Java Location API (JSR-179) support with external GPS (e.g. Nokia S60 v3 phone).

There are several ways how to get user location via MGMaps Lib SDK API,:

- a) JSR-179 API support, see below
- b) Bluetooth GPS
- c) Cell-ID positioning (works on some phones, like SonyEricsson Java phones)

1. Import location packages to the Midlet

```
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 (default provider, which is JSR-179), and define special type of object (LocationMarker) on map, to show GPS location. Note that some images are needed to show the marker

```
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);
    }
```

```
mapItem.setLocationSource(dataSource);  
} catch (final IOException e) {}  
}
```

3. Build and run your changed code. Note that the WTK emulator will ask for additional permission to get location information, also real devices will do similar way. You will probably notice that the WTK emulator gives default location coordinates 0,0; this is somewhere in Atlantic ocean, off Nigerian coast. So you will probably see blue sea and nothing more on the map. You can change coordinates using WTK Emulator's menu MIDlet > External events, and change Latitude and Longitude in the first tab there. However, when emulator and Midlet inside it already running, then this simple example coordinates will be not reloaded.
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 many minutes.

Note also that some phones just do not start application (without giving any error) if you try to use Location API, e.g. SonyEricsson phones up to JP-7 and SJP-3.

8 Basic Events

The mapItem can send messages to your midlet application, these are fired based on some events, like selecting particular marker on map, clicking, moving and zooming of the map, also notifying about possible errors in library. Let's try to get marker-related events to the application.

1. Import PlaceListener package to the Midlet

```
import com.nutiteq.listeners.OnMapElementListener;
```

2. Define Control key for place selection (-5 is select key in WTK emulator and in most Nokia and SonyEricsson phones)

```
mapItem.defineControlKey(ControlKeys.SELECT_KEY, -5);
```

3. Add PlaceListener implementation reference to the midlet's main class

```
public class HelloMap  
extends MIDlet  
implements CommandListener, OnMapElementListener {
```

4. Add one more element to the form, just to have place to display texts

```
// define property for HelloMap class
private StringItem message;

// define new item for the form, in startApp class, set initial value
message = new StringItem("", "");
mMainForm.append(message);
```

5. Add actual handler for the elementClicked, elementEntered and elementLeft events, all these update message in form. In real application perhaps more complicated logics should happen for place clicking, e.g. showing separate screen with object details. Note that all these methods must be defined, if OnMapElementListener is used.

```
public void elementClicked(final OnMapElement p) {
    message.setText("Clicked place name: " + p.getLabel().getLabel());
}

public void elementEntered(final OnMapElement p) {
    message.setText("Entered place name: " + p.getLabel().getLabel());
}

public void elementLeft(final OnMapElement p) {
    message.setText("Left place name: " + p.getLabel().getLabel());
}
```

6. Define listener class for OnMapElementListener (in our case it is HelloMap midlet itself)

```
mapItem.setOnMapElementListener(this);
```

Note that the OnMapElement can be Place (point), Line or Polygon object, and depending on the object different methods of it could be available.

9 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 Midlet

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

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

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

3. In J2ME console window (shown in IDE, but not in real device) you should see debug log. 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, 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).

10 Full Hello Map listing

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

```
package com.tutorial;  
  
import javax.microedition.midlet.MIDlet;  
import javax.microedition.midlet.MIDletStateChangeException;  
import javax.microedition.lcdui.*;  
import java.io.IOException;  
import com.nutiteq.MapItem;  
import com.nutiteq.ui.DefaultCursor;
```

```
import com.nutiteq.utils.Utils;
import com.nutiteq.components.WgsPoint; // for control keys
import com.nutiteq.controls.ControlKeys; // import definition of Place (this is the item on map)
import com.nutiteq.controls.OnScreenZoomControls;
import com.nutiteq.components.OnMapElement;
import com.nutiteq.components.Place; // imports for line drawing
import com.nutiteq.components.Line;
import com.nutiteq.components.LineStyle; //to get events from MapItem
import com.nutiteq.listeners.OnMapElementListener;
import com.nutiteq.kml.*; //KML package
import com.nutiteq.location.LocationMarker; // location packages
import com.nutiteq.location.LocationSource;
import com.nutiteq.location.providers.LocationAPIProvider;
import com.nutiteq.location.NutiteqLocationMarker; //for PlaceIcon type
import com.nutiteq.components.PlaceIcon;

public class HelloMap extends MIDlet implements CommandListener,
    OnMapElementListener {
    // A screen form for the MapItem
    private Form mMainForm;
    // MapItem
    private MapItem mapItem;
    // Place icon on map
    private Image icon;
    // StringItem for message display
    private StringItem message;
    // the display
    private Display display;

    public HelloMap() {
        // creation of MapItem (custom form element)
        mapItem = new MapItem("Map", "tutorial", this, 200, 150, new WgsPoint(
            24.764580, 59.437420), 12);

        // form definition
        mMainForm = new Form("Hello map");
    }

    // handlers for the OnMapElement events
    public void elementClicked(final OnMapElement p) {
        message.setText("Clicked place name: " + p.getLabel().getLabel());
    }

    public void elementEntered(final OnMapElement p) {
        message.setText("Entered place name: " + p.getLabel().getLabel());
    }

    public void elementLeft(final OnMapElement p) {
        message.setText("Left place name: " + p.getLabel().getLabel());
    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
        // TODO Auto-generated method stub
    }
}
```

```
}

protected void pauseApp() {
    // TODO Auto-generated method stub
}

protected void startApp() throws MIDletStateChangeException {
    // Display the form when the app starts

    mapItem.defineControlKey(ControlKeys.MOVE_UP_KEY, Canvas.KEY_NUM2);
    mapItem.defineControlKey(ControlKeys.MOVE_DOWN_KEY, Canvas.KEY_NUM8);
    mapItem.defineControlKey(ControlKeys.MOVE_LEFT_KEY, Canvas.KEY_NUM4);
    mapItem.defineControlKey(ControlKeys.MOVE_RIGHT_KEY, Canvas.KEY_NUM6);
    mapItem.defineControlKey(ControlKeys.ZOOM_IN_KEY, Canvas.KEY_POUND);
    mapItem.defineControlKey(ControlKeys.ZOOM_OUT_KEY, Canvas.KEY_STAR);

    // define a Control key for place selection (-5 = select key)
    mapItem.defineControlKey(ControlKeys.SELECT_KEY, -5);

    // On screen map controls (for touch-screen phones)
    final Canvas canvas = new Canvas() {
        protected void paint(Graphics arg0) {
        }
    };
    final boolean pointer = canvas.hasPointerEvents();

    // If phone has a pointer, then show map controls on the map
    if (pointer) {
        mapItem.setOnScreenZoomControls(new OnScreenZoomControls(Utils
            .createImage(OnScreenZoomControls.DEFAULT_ZOOM_IMAGE)));
    } else {
        mapItem.setCursor(new DefaultCursor(0xFFFF0000));
    }

    // your custom image file (this one is 16x16 pixels)
    try {
        icon = Image.createImage("/nutiteq.png");
    } catch (IOException e) {
    }
    // add place to the map with the icon and label
    mapItem.addPlace(new Place(1, "Tallinn", icon, 24.764580, 59.437420));

    // Drawing a line to map
    // first define the line points
    final WgsPoint[] linePoints = {
        new WgsPoint(24.76382468302337, 59.44325151314919),
        new WgsPoint(24.76344295658494, 59.4462352840583),
        new WgsPoint(24.76593650384734, 59.44530921763007),
        new WgsPoint(24.76804665483925, 59.44616268729941),
        new WgsPoint(24.76810500478219, 59.443291656657) };
    // ...and then define a new line using those points
    final Line line = new Line(linePoints, new LineStyle(0xFF0000, 5));
    // display the line on map
    mapItem.addLine(line);
}
```

```
// define listener class for PlaceListener
mapItem.setOnMapElementListener(this);

// add KML layer to the map with Panoramio popular images
mapItem
    .addKmlService(new KmlUrlReader(
        "http://www.panoramio.com/panoramio.kml?LANG=en_US.utf8",
        true));

// show GPS location on map using Library's location API
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 Placelcon(gpsPresentImage, 4, 16), new Placelcon(
                gpsConnectionLost, 4, 16), 3000, true);
        dataSource.setLocationMarker(marker);
        mapItem.setLocationSource(dataSource);
    } catch (final IOException e) {
    }
}

mMainForm.append(new StringItem(null, "Hello, map!"));
mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
mMainForm.setCommandListener(this);
// add the MapItem to screen form
mMainForm.append(mapItem);
// add the StringItem to the form
message = new StringItem("", "");
mMainForm.append(message);

// Display.getDisplay(this).setCurrent(mMainForm);
display = Display.getDisplay(this);
display.setCurrent(mMainForm);
// start mapping process
mapItem.startMapping();
}

// to handle key press events
public void commandAction(Command c, Displayable s) {
    notifyDestroyed();
}
}
```



nutiteq

11 Next steps: using mapComponent API

MGMaps mapping library has also low level mapComponent API, which extends Midlet's Canvas and has more or less full control over user input and given output. Using mapComponent requires more coding, developer must take care of user's screen size, handle specifically paints, key-presses etc, but it gives also full control over graphics and user logics; so you can more polished UI. Most real commercial applications will probably choose to use mapComponent API level.

See Mapping Lib API developer guide and example applications' sources to see how mapComponent is used.