



nutiteq

Nutiteq Maps SDK tutorial for Java ME (J2ME)

Version 1.1.1 (updated 29.08.2011)

© 2008-2011 Nutiteq LLC

Nutiteq LLC

www.nutiteq.com

Skype: nutiteq

support@nutiteq.com



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.....	12
8	Basic Events.....	10
9	Adding on-line KML file data to the map.....	11
10	Full Hello Map listing.....	12
11	Next steps: using mapComponent API.....	16

2 Introduction

The J2ME mapping lib enables adding interactive maps feature to mobile java (J2ME) applications, 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, Nutiteq SDK provides following key API objects:

- 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 (e.g. cannot catch arrow/pointer keys).
- 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 define any default map.

Following tutorial includes only most basic functions and examples of the library, full SDK has much more features and options, see Nutiteq SDK Developer guide, Javadocs and sample applications.

Note that Nutiteq mapping SDK 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. You are free to use J2ME Polish, LWUIT or some other Java ME application framework.

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 with OnMapElementListener

Jaakl	29.08.2011	1.1.1	Updated for 1.1.1 release
-------	------------	-------	---------------------------

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 **Mobile Tools for Java** plug-in (<http://www.eclipse.org/mtj/>).

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. Select SRC folder and New > Java ME MIDlet in context menu. Enter com.tutorial as package name and HelloMap as midlet name.
3. Modify the class so it:
 - a) the class implements CommandListener – to catch commands
 - b) has field mMainForm, which is created in constructor
 - c) append one StringItem and commands for Exit to the mMainForm.
 - d) exit application upon command action
4. You should get following code as result:

```
package com.tutorial;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class HelloMap extends MIDlet implements CommandListener {

    private Form mMainForm;

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

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

```

5. Save the java file.
6. Now let's try if it compiles properly. Right-click on project, select Run As > Emulated J2ME MIDlet. It should start emulator and show screen with just *Hello, Map!* in it. So far so good, but there is no map image yet? We'll add it with the next steps.
7. Add Nutiteq Maps 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.
8. 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.
9. 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.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import com.nutiteq.MapItem;
import com.nutiteq.components.WgsPoint;

public class HelloMap extends MIDlet implements CommandListener {

    private Form mMainForm;
    private MapItem mapItem;

    public HelloMap() {

```

```

        mMainForm = new Form("Hello map");
        mapItem = new MapItem("Map", "tutorial", this, 300, 150, new WgsPoint(24.764580, 59.437420), 12);
    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
        mapItem.stopMapping();
    }

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

    protected void startApp() throws MIDletStateChangeException {
        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 commandAction(Command c, Displayable s) {
        notifyDestroyed();
    }
}

```

10. 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 field
- 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.
11. 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.



12. 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 notice that you cannot pan the map using arrow/joystick buttons nor with mouse/touch. 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 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 this is why we suggest to use low-level mapping API (MapComponent) for better user experience. 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 enable touch screen emulation, by default it is turned off. You can find the setting from file:

C:\WTK2.5.2_01\wtllib\devices\DefaultColorPhone\DefaultColorPhone.properties and you must set there **touch_screen=true**

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. You have to create icon for the marker and read image into that. SDK has one useful image file what we use here, called "def_kml.png". Add following line to startApp method

```
Image icon = Image.createImage("/images/def_kml.png");
```

NB! This Image is not J2ME Image, it is Nutiteq wrapper, so make sure that you import it like following:

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

2. Now we can add Place (which is a point element) to the map, with the icon and defined label.

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

3. Parameters of the Place are:

- a. Id of the place (1). This is used to identify specific place later in OnElementListener, 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 as following:



4. 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*. We define also textual label for the line, which will be visible when user clicks to the line.

```
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), new
PlaceLabel("My Line", PlaceLabel.DISPLAY_BOTTOM));
    mapItem.addLine(line);
```

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

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

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

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

3. 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);
```

4. 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());
    }
}

```

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

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

6. Run application again and try to click on places.

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

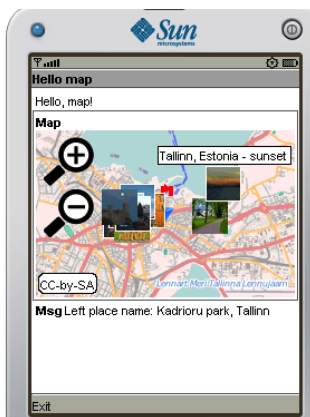
8 Adding on-line KML file data to the map

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 generally supported.

1. 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));
```

2. Run application again and you should start to see a lot of small photos on map:



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 the KML features, e.g. KMZ (but gzipped KML content is actually supported and very useful to reduce data amount), *link* elements, time-based refreshes, more advanced style parameters (e.g. scale of icons).

9 Show GPS location on map

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 Nutiteq SDK:

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

All them provide same API for application developer, but you still need to configure specific location provider first with specific methods.

1. Define location source (default provider, which is JSR-179), and define special type of object (LocationMarker) on map, to show GPS location. We use same image from SDK as in previous sample, you'll find probably better for your own app.

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

2. 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. Sun 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 only location marker on the map. You can change emulated GPS coordinates using WTK Emulator's menu MIDlet > External events, and change Latitude and

Longitude over there.

3. On real devices you may need some luck to get your actual GPS coordinates. First of all – on many phones it does not work indoors, at least get next to a window. Another thing what you will notice is that first GPS location fix takes time – depending on your phone, A-GPS service provider, physical location (indoors/outdoors) and last location fix it will take from 30 seconds to many minutes (could be even 15-20 minutes!).

Note also that some phones just do not start application if you try to use Location API, e.g. SonyEricsson phones up to JP-7 and SJP-3, also Nokia S40 phones without GPS. They will not give any error, or give unspecific installation error.

Some phones, like Nokia S60 require that you add following line to the JAD file, otherwise phone does not provide GPS to your application:

```
MIDlet-Permissions-Opt: javax.microedition.location.Location
```

10 Full Hello Map listing

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

```
package com.tutorial;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.StringItem;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import com.nutiteq.MapItem;
import com.nutiteq.components.Line;
import com.nutiteq.components.LineStyle;
import com.nutiteq.components.OnMapElement;
import com.nutiteq.components.Place;
import com.nutiteq.components.Placelcon;
import com.nutiteq.components.PlaceLabel;
import com.nutiteq.components.WgsPoint;
import com.nutiteq.controls.ControlKeys;
import com.nutiteq.controls.OnScreenZoomControls;
import com.nutiteq.kml.KmlUrlReader;
import com.nutiteq.listeners.OnMapElementListener;
```



```
import com.nutiteq.location.LocationMarker;
import com.nutiteq.location.LocationSource;
import com.nutiteq.location.NutiteqLocationMarker;
import com.nutiteq.location.providers.LocationAPIProvider;
import com.nutiteq.ui.DefaultCursor;
import com.nutiteq.utils.Utils;
import com.nutiteq.wrappers.Image;

public class HelloMap extends MIDlet implements CommandListener,
    OnMapElementListener {

    private Form mMainForm;
    private MapItem mapItem;
    private StringItem message;

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

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
        mapItem.stopMapping();
    }

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

    protected void startApp() throws MIDletStateChangeException {

        // define numeric arrow 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);

        // select key for OnMapElementListener events
        mapItem.defineControlKey(ControlKeys.SELECT_KEY, -5);

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

        // set pointer for non-touch and on-screen zoom buttons for touch
        if (pointer) {
            mapItem.setOnScreenZoomControls(new OnScreenZoomControls(Utils
                .createImage(OnScreenZoomControls.DEFAULT_ZOOM_IMAGE)));
        }
    }
}
```

```

    } else {
        mapItem.setCursor(new DefaultCursor(0xFFFF0000));
    }

    // add Point to map
    Image icon = Image.createImage("/images/def_kml.png");
    mapItem.addPlace(new Place(1, "Tallinn", icon, 24.764580, 59.437420));

    // add Line to map
    WgsPoint[] linePoints = {
        new WgsPoint(24.76382468302337, 59.44325151314919),
        new WgsPoint(24.76344295658494, 59.4462352840583),
        new WgsPoint(24.765936503884734, 59.44530921763007),
        new WgsPoint(24.76804665483925, 59.44616268729941),
        new WgsPoint(24.76810500478219, 59.443291656657) };

    Line line = new Line(linePoints, new LineStyle(0xFF0000, 5),
        new PlaceLabel("My Line", PlaceLabel.DISPLAY_BOTTOM));
    mapItem.addLine(line);

    // add KML on-line service
    mapItem.addKmlService(new KmlUrlReader(
        "http://www.panoramio.com/panoramio.kml?LANG=en_US.utf8", true));

    // GPS location to map

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

    // add form elements to screen
    mMainForm.append(new StringItem(null, "Hello, map!\n"));
    mMainForm.append(mapItem);

    // String item for events
    message = new StringItem("Msg", "Here comes some message");
    mMainForm.append(message);

    // define listener for map element events
    mapItem.setOnMapElementListener(this);

    mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
    mMainForm.setCommandListener(this);
    Display.getDisplay(this).setCurrent(mMainForm);
    mapItem.startMapping();
}

```

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

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

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

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

11 Next steps: using MapComponent API

Nutiteq mobile mapping library has also low level MapComponent API, which 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. Basic functions like adding overlays, getting events, getting GPS location etc are same as in the tutorial above.