

Implementing a custom MapUrlProvider overlay with Maps API for Java ME

This article explains how to implement a custom MapUrlProvider and how to add it as overlay over a map by using the [HERE Maps API for Java ME](#)

Introduction



The [HERE Maps API for Java ME](#) offers a feature to easily add new transparent layers over the default map tiles, or to overlay the default tiles with custom ones, thanks to the MapProvider object.

MapProvider and MapUrlProvider

A MapProvider object is responsible for generating the map tile Images that are displayed by a MapDisplay. Those Image objects are generated and returned by the `getTileImage()` method, that returns a tile corresponding to a row/column/zoom and with a given width and height. The row and column parameters are generated by using the [Mercator projection](#), and a sample implementation is available on the MapProvider JavaDocs page.

The MapProvider base class has a direct subclass, MapUrlProvider, that makes it possible to retrieve tiles from remote providers, and integrates all the logic for fetching and generating the tile Image objects. This is done by creating an implementation of the abstract `getTileUrl()` method that returns the URL of the remote tile, corresponding to the given row, column, zoom, width and height parameters:

Implementing a Custom MapUrlProvider

In this example, we will be implementing a custom MapUrlProvider which retrieves Map Tiles from a tile server run by the [National Library of Scotland](#). The mapping is based on out-of-copyright Ordnance Survey maps, dating from the 1920s to the 1940s.

The NLSMapProvider class must extend MapUrlProvider, as shown below; the base class accepts two parameters, a tile size and a duration of tile validity. Usually tiles are 128 pixels by 128 pixels for Java ME phones. The duration of validity is a maximum time (in seconds) the device will cache a tile for.

```
public class NLSMapProvider extends MapUrlProvider {  
    private static final long MAP_VALIDITY = (24L * 3600L * 1000L); // Valid 1000 days.
```

```

public NLSMapProvider() {
    super(MapDisplay.MAP_RESOLUTION_128_x_128, MAP_VALIDITY);
}
}

```

The MapUrlProvider defines several abstract methods, all of which must be defined in the concrete implementation class:

- public String getName() returns a hard-coded string that defines the name of the custom map overlay provider.
- public String getTileUrl(int zoom, int column, int row) is a method for constructing the URL of a map tile from the map tiler service
- public boolean isTileSupported(int zoom, int column, int row) returns a boolean true/false stating whether the given overlay tile exists.
- public final Image decode(byte[] data) that returns the actual overlay as an Image from the given data that has been received from the URL.

The getName() method can be simply implemented by returning a name for the implemented custom map provider. In this case we will return the name "historic" for out-of-copyright Ordnance Survey maps, dating from the 1920s to the 1940s.

```

public String getName() {
    return "historic";
}

```

Since the tile image must be generated by a remote script or page, the getTileUrl() method has to pass all its parameter to the remote Web server, and so the generated URL has to contain those parameters in its query string. In the case of the National Library of Scotland tile server, URLs are of the form http://t4.uk.tileservers.com/_os1/r0/5/15/9.jpg where The last three digits are the zoom, the column and the row respectively, and the first digit is used for load balancing. Since the tiles returned are JPEGs. we also need to override the default MIME type (which expects PNG files)

```

public String getTileUrl(int zoom, int column, int row, int width, int height)
{
    int no = (column + row) % 5;
    StringBuffer buffer = new StringBuffer(TILE_SERVER_1);
    buffer.append(no);
    buffer.append(TILE_SERVER_2);
    buffer.append(zoom);
    buffer.append("/");
    buffer.append(column);
    buffer.append("/");
    buffer.append(row);
    buffer.append(TILE_SERVER_SUFFIX);
    return buffer.toString();
}

public String getMimeType() {
    return "image/jpeg";
}

```

If you attempt to download a tile and the tile server cannot respond, an application error is thrown. It is therefore necessary to check for the validity of the request prior to requesting a tile. If this calculation is simple, then the implementation of the MapUrlProvider class can provide this information directly. If it is more complex, it is necessary to make a server call to find out whether the tile can be found **prior** to downloading it. In our case, the tile server appears to return a 404 with an image type of a single pixel GIF if the request is for an area uncovered by the overlay. Therefore a check needs to be made to avoid downloading any "missing" tiles.

Since it is not ideal to make an additional download request for each tile like this, ideally the server should offer a RESTful validity

test so we can find out whether the area is covered. Of course, if the overlay is available at all zooms/areas, the function could be hard coded to just return `true`.

In an attempt to reduce network traffic, this function will automatically reject any tile requests outside of a rectangle over the United Kingdom.

```
public static final int MAX_ZOOM = 15;
public static final int MIN_ZOOM = 5;
/* The following tiles are over the UK at zoom 5. */
private static final int COL_TILE_MIN_LIMIT = 14;
private static final int COL_TILE_MAX_LIMIT = 17;
private static final int ROW_TILE_MIN_LIMIT = 8;
private static final int ROW_TILE_MAX_LIMIT = 11;

private boolean isTileInUK(int zoom, int column, int row) {

    int zoomOffset = zoom - MIN_ZOOM;
    int multi = 1;
    while (zoomOffset > 0) {
        multi = multi * 2;
        zoomOffset--;
    }
    if (column / multi <= COL_TILE_MIN_LIMIT || column / multi >=
COL_TILE_MAX_LIMIT) {
        return false;
    } else if (row / multi <= ROW_TILE_MIN_LIMIT || row / multi >=
ROW_TILE_MAX_LIMIT) {
        return false;
    }
    return true;
}
```

If a tile request is within the UK and at an appropriate zoom, we will still need to check for validity.

```
public boolean isTileSupported(int zoom, int column, int row) {
    if (zoom < MIN_ZOOM || zoom > MAX_ZOOM || !isTileInUK(zoom, column, row)) {
        // This tile is definitely not supported.
        return false;
    }

    // Maintain a hash of supported tiles.
    Boolean retValue = Boolean.FALSE;
    String hashKey = "" + zoom + column + row;

    if (tiles.containsKey(hashKey)) {
        // This tile has been downloaded previously.
        retValue = (Boolean) tiles.get(hashKey);
    } else {
        // Try downloading it to see if it succeeds.
        try {
            byte[] ignored = get(zoom, column, row);
            retValue = Boolean.TRUE;
            tiles.put(hashKey, retValue);
        } catch (IOException ioe) {
            tiles.put(hashKey, retValue);
        }
    }
}
```

```


        } catch (InterruptedException ie) {
        }

    }

    return retValue.booleanValue();
}

```

The `decode()` method takes the received data and creates an image for display. It is also possible to do any necessary additional processing. In this case, the tiles returned are too big, and need to be reduced in size. Having received data for an image we need to decode and display it. Unfortunately in our case, the tiles are returned at 256 x 256 so we need to reduce the size.

 **Note:** If you have total control over your tile server, it would make sense to produce tiles of the correct size server side, since it would reduce network traffic.

```

public final Image decode(byte[] data) {
    return scaleImage(Image.createImage(data, 0, data.length));
}

private Image scaleImage(Image sourceImage) {

    Image tmp = Image.createImage(MapDisplay.MAP_RESOLUTION_128_x_128,
MapDisplay.MAP_RESOLUTION_256_x_256 );
    Graphics g = tmp.getGraphics();
    int ratio = (MapDisplay.MAP_RESOLUTION_256_x_256 << 16) /
MapDisplay.MAP_RESOLUTION_128_x_128;
    int pos = ratio/2;

    //Horizontal Resize

    for (int x = 0; x < MapDisplay.MAP_RESOLUTION_128_x_128; x++) {
        g.setClip(x, 0, 1, MapDisplay.MAP_RESOLUTION_256_x_256);
        g.drawImage(sourceImage, x - (pos >> 16), 0, Graphics.LEFT | Graphics.TOP);
        pos += ratio;
    }

    Image resizedImage = Image.createImage(MapDisplay.MAP_RESOLUTION_128_x_128,
MapDisplay.MAP_RESOLUTION_128_x_128);
    g = resizedImage.getGraphics();
    ratio = (MapDisplay.MAP_RESOLUTION_256_x_256 << 16) /
MapDisplay.MAP_RESOLUTION_128_x_128;
    pos = ratio/2;

    //Vertical resize

    for (int y = 0; y < MapDisplay.MAP_RESOLUTION_128_x_128; y++) {
        g.setClip(0, y, MapDisplay.MAP_RESOLUTION_128_x_128, 1);
        g.drawImage(tmp, 0, y - (pos >> 16), Graphics.LEFT | Graphics.TOP);
        pos += ratio;
    }
    return resizedImage;
}

```

Adding the CustomMapUrlProvider to a MapDisplay

Once the custom provider has been implemented, it is necessary to add it to the desired `MapDisplay` object. This can be done by using the `MapDisplay.addMapOverlay()` method.

```
historic = new NLSMapProvider();
map.addMapOverlay(historic);
```

Adding Attribution

see also: [Adding an Attribution component](#)

Unless you own your own map tile source, you will have to comply with the licensing agreement of the map tile provider. In many cases this will mean giving some **attribution** to the original owner, even if the tiles are being offered without cost. Here is the licensing agreement from the National Library of Scotland:

"You can embed the map in your own website, display your own markers or mapping data on top of it, use it for research purposes, or create derivative work from it. The only condition is that you must display an attribution to the National Library of Scotland, together with a link to the National Library of Scotland website <http://www.nls.uk/> whenever our map is used. If you create derivative work, the documentation of your work must contain this attribution"

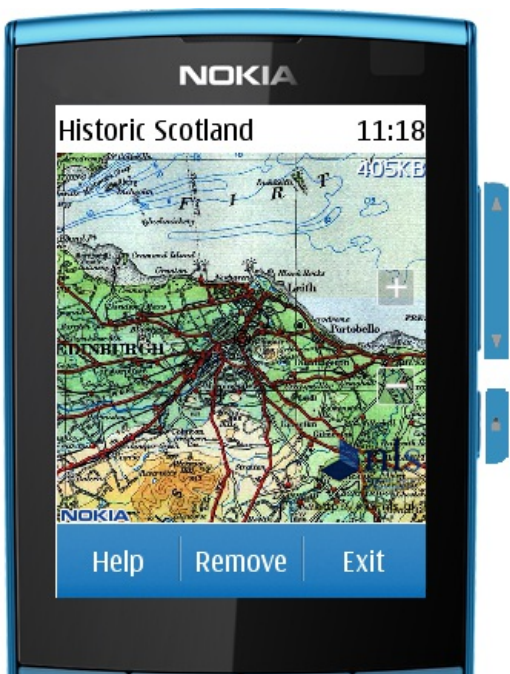
This condition may be satisfied by overlaying an appropriate graphic **on top** of the custom tiles. The following PNG file is added to the resources of the JAR file



The PNG needs to be displayed whenever the overlay is used, this can be checked by looking at the `MapDisplay.getAllMapOverlays()` method. If an overlay is found, the graphic is painted over the map canvas. Since the PNG file has a transparent background, the map display will show through under the graphic.

```
protected void paint(Graphics g) {
    super.paint(g);
    if (map.getAllMapOverlays().length > 0) {
        Image attribution = loadAttributionImage();
        g.drawImage(attribution,
            getWidth() - attribution.getWidth(),
            getHeight() - 5,
            Graphics.BOTTOM | Graphics.LEFT);
    }
}
```

The final result of applying the `NLSMapProvider` defined above is visible in the following screenshot.



Creating your own map provider server

The example above uses an existing map tile service. However a provider of Map tiles can be implemented in any available technology (for instance, Java, PHP, Perl). The following code snippet shows how to write a simple script that generate tiles with a thin border and with the column and row parameters displayed on the tile itself.

```
<?php

$row = $_REQUEST['row'];
$column = $_REQUEST['column'];
$width = $_REQUEST['width'];
$height = $_REQUEST['height'];
$zoom = $_REQUEST['zoom'];

// create an empty image with the given width and height
$image = imagecreatetruecolor($width, $height);

// set the background color as transparent
$black = imagecolorallocate($image, 0, 0, 0);
imagecolortransparent($image, $black);

// create the needed colors
$redColor = imagecolorallocatealpha($image, 255, 0, 0, 0);
$grayColor = imagecolorallocatealpha($image, 10, 10, 10, 0);

// paint a border around the tile image
imagerectangle($image, 0, 0, $width - 1, $height - 1, $redColor);

// write the column and row parameters on the tile
$text = $column . ' - ' . $row;
imagestring($image, 5, $width / 2 - 50, $height / 2 + 1, $text, $grayColor);
imagestring($image, 5, $width / 2 - 50, $height / 2, $text, $redColor);

// send the image to the client
header("Content-type: image/png");
imagepng($image);
```

Summary

The `MapUrlProvider` allows us to easily define interfaces to remote providers of map tiles. Most initial processing should be provided server side, but if an application needs more control over the generated map tiles it is possible to do post-processing with a `MapProvider` as well.