Drawing shapes with Windows Phone maps API

Introduction



09 Dec 2012

This article shows how to draw shape objects on a map in Windows Phone applications. Generally you can add polylines, polygons and MapOverlay objects into the map, and there are no predefined shapes which could be drawn into the map. The complete source code for the example illustrated in this article can be found from Windows Phone 8 Maps Examples project, the code used here is implemented for example in MoreMapContent example inside that project.

Implementation

With MapOverlay in theory you could add any types of drawable objects as content, thus you could add rectangle or any other pre-defined objects into the map. The code could for example look like this:

```
MapLayer Rectangle1 = new MapLayer();
Rectangle rectanggle = new Rectangle();
rectanggle.Fill = new SolidColorBrush(Colors.Green);
rectanggle.Stroke = new SolidColorBrush(Colors.Blue);
rectanggle.StrokeThickness = 4;
rectanggle.Width = 200;
rectanggle.Height = 200;
MapOverlay pin1 = new MapOverlay();
pin1.GeoCoordinate = new GeoCoordinate(60.22, 24.81);
pin1.Content = rectanggle;
Rectangle1.Add(pin1);
map1.Layers.Add(Rectangle1);
```

The problem with this kind of approach is that MapOverlay objects are sized with pixel size, and they are not re-sized with map zooming (unless there are some custom logic implemented for doing this)

Thus if you need to add the shape to a geographical area, and it should be fitted right automatically with all zoom levels, you should use polylines and polygons, and just shape the area with GeoCoordinates.

The Windows Phone 8 Maps Examples project and the MoreMapContent example implements simple codes for generating GeoCoordinateCollection's which could be used as polyline and polygon Path variables.

The rectangle is constructed from two points given and the circle is constructed for given center and radius values.

The code for rectangle simply returning four corner points for which the polyline or polygon could be drawn with:

```
public static GeoCoordinateCollection CreateRectangle(GeoCoordinate topLeft,
GeoCoordinate bottomRight)
{
    var locations = new GeoCoordinateCollection();
    locations.Add(new GeoCoordinate(topLeft.Latitude, topLeft.Longitude));
    locations.Add(new GeoCoordinate(topLeft.Latitude, bottomRight.Longitude));
    locations.Add(new GeoCoordinate(bottomRight.Latitude, bottomRight.Longitude));
    locations.Add(new GeoCoordinate(bottomRight.Latitude, topLeft.Longitude));
    return locations;
}
```

And the code for the circle returns points for creating the whole 360 degree circle:

```
public static GeoCoordinateCollection CreateCircle(GeoCoordinate center, double radius)
{
    var earthRadius = 6367.0; // radius in kilometers
```

```
    var lat = ToRadian(center.Latitude); //radians
    var lng = ToRadian(center.Longitude); //radians
    var d = radius / earthRadius; // d = angular distance covered on earth's surface
    var locations = new GeoCoordinateCollection();

   for (var x = 0; x <= 360; x++)
    {
        var brng = ToRadian(x);
        var latRadians = Math.Asin(Math.Sin(lat) * Math.Cos(d) + Math.Cos(lat) *
Math.Sin(d) * Math.Cos(brng));
        var lngRadians = lng + Math.Atan2(Math.Sin(brng) * Math.Sin(d) * Math.Cos(lat),
Math.Cos(d) - Math.Sin(lat) * Math.Sin(latRadians));
        locations.Add(new GeoCoordinate(ToDegrees(latRadians), ToDegrees(lngRadians)));
    }

      return locations;
}
```