Location Racing

Code

By

Philip Stafford

Table of Contents

1. Introduction

Disclaimer. Most of the code in this document is my code. There are some sections of code which were developed by other people and the have been credited accordingly.

2. ExampleLauncher.Java

This file was originally developed by Nicolas Gramlich and modified by Philip Stafford. The code can be found in Nicolas's Github repository. (Gramlich, 2010)

All of my code is clearly marked within the code.

```
package org.andengine.examples.launcher;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

import org.andengine.AndEngine;
import org.andengine.examples.R;
import org.andengine.util.debug.Debug;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.app.ExpandableListActivity;
```

```
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.os.Environment;
import android.provider.Settings;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ExpandableListView;
import android.widget.Toast;

import com.locationracing.app.DBTools;
import com.locationracing.app.DownloaderThread;
import com.locationracing.app.NortSouthEastWestBoundaries;

/**
 * (c) 2010 Nicolas Gramlich
 * (c) 2011 Zynga Inc.
 *
 * @author Nicolas Gramlich
 * @since 22:56:46 - 16.06.2010
 */
```

```
public class ExampleLauncher extends ExpandableListActivity implements LocationListener {

	private static final String PREF_LAST_APP_LAUNCH_VERSIONCODE_ID = "last.app.launch.versioncode";

	private static final int DIALOG_FIRST_APP_LAUNCH = 0;
	private static final int DIALOG_NEW_IN_THIS_VERSION = ExampleLauncher.DIALOG_FIRST_APP_LAUNCH + 1;
	private static final int DIALOG_BENCHMARKS_SUBMIT_PLEASE =
ExampleLauncher.DIALOG_NEW_IN_THIS_VERSION + 1;
	private static final int DIALOG_DEVICE_NOT_SUPPORTED =
ExampleLauncher.DIALOG_BENCHMARKS_SUBMIT_PLEASE + 1;

	// ============================================================
	// My code - START
	// ============================================================

	// ============================================================
	// Constants
	// ============================================================
	// debug statement tags
	private final String LOGTAG = "[GPS]";
	private final String LOGTAGROUNDED = "[GPS Rounded]";
	private final String LOGTAGSTRING = "[GPS String]";
	private final String LOGTAG2 = "[Test Loc]";
	private final String LOCTAG = "[LAT & LON]";
	private static final String TAGBUTTON = "[Button]";
	private static final String XMLTAG = "[XMLPARSE]";
	private static final String FILETAG = "[FILE]";

	private static final String DB_FULL_PATH = "/data/data/org.andengine.examples/databases/mapdata.db";

	// ============================================================
	// Variables
	// ============================================================
	// variables for current location and map data
	private LocationManager locationManager;
```

```
    private String provider;

    // values to add and subtract to the users position
//  private double latBoundary = .0105;
//  private double lonBoundary = .0199;

    // for a smaller more manageable area
    private double latBoundary = .01;
    private double lonBoundary = .0056;

    // variables to store the coordinate calculations
    private double minLatSouth = 0;
    private double minLonWest = 0;
    private double maxLatNorth = 0;
    private double maxLonEast = 0;
    private double latCurrentPos = 0;
    private double lonCurrentPos = 0;

    // variables for the API call URL's
    public String nodeApiUrl = "http://overpass.osm.rambler.ru/cgi/interpreter?data=node%28";
    public String wayApiUrl = "http://overpass.osm.rambler.ru/cgi/interpreter?data=way%28";
    private String commonPartOfApiUrl = "";
    private String endOfApiUrl = "%29%3Bout%3B";
    private String asciiCommaValue = "%2C";
    private String asciiMinusValue = "%2D";
    private String asciiFullStopValue = "%2E";

    // object to store the coordinates to be used later in another class
    public NortSouthEastWestBoundaries coordinates;

    // Used to communicate state changes in the DownloaderThread
    public static final int MESSAGE_DOWNLOAD_STARTED = 1000;
    public static final int MESSAGE_DOWNLOAD_COMPLETE = 1001;
    public static final int MESSAGE_UPDATE_PROGRESS_BAR = 1002;
    public static final int MESSAGE_DOWNLOAD_CANCELED = 1003;
```

```
public static final int MESSAGE_CONNECTING_STARTED = 1004;
public static final int MESSAGE_ENCOUNTERED_ERROR = 1005;

// instance variables
private ExampleLauncher thisActivity;
private Thread downloaderThread;
private ProgressDialog progressDialog;

// file names
private String nodeFileName = "nodeInfo.xml";
private String wayFileName = "wayInfo.xml";

// XML tags
private static final String LAT = "lat";
private static final String LON = "lon";
private static final String NODE = "node";
private static final String NODE_ID = "id";
private static final String WAY = "way";
private static final String WAY_ID = "id";
private static final String REF = "ref";
private static final String ND = "nd";
private static final String TAG = "tag";
private static final String K = "k";
private static final String HIGHWAY = "highway";

// Used to make the URL to call for XML data
private String nodeFile = "/nodeInfo.xml";
private String wayFile = "/wayInfo.xml";

// Holds values pulled from the XML document using XmlPullParser
String[][] xmlPullParserArray = {{"ID", "0"}, {"Lat", "0"}, {"Lon", "0"}};

// will contain GPS coordinates
private Location location;
```

```
// reference to the DBTools class
DBTools dbTools = new DBTools(this);

boolean dataEnabled;

// ===========================================================
// My code - END
// ===========================================================

private ExpandableExampleLauncherListAdapter mExpandableExampleLauncherListAdapter;
private int mVersionCodeCurrent;
private int mVersionCodeLastLaunch;

@Override
public void onCreate(final Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);

	if(!AndEngine.isDeviceSupported()) {
		this.showDialog(ExampleLauncher.DIALOG_DEVICE_NOT_SUPPORTED);
	}
	this.setContentView(R.layout.list_examples);
	this.mExpandableExampleLauncherListAdapter = new ExpandableExampleLauncherListAdapter(this);
	this.setListAdapter(this.mExpandableExampleLauncherListAdapter);

	// ===========================================================
	// My code - START
	// ===========================================================

	// variables for location detection
	LocationManager service = (LocationManager) getSystemService(LOCATION_SERVICE);
	boolean enabled = service.isProviderEnabled(LocationManager.GPS_PROVIDER);

	// check for data connection
	dataEnabled = isNetworkAvailable();
```

```
        if (!dataEnabled) {
              new AlertDialog.Builder(this)
             .setTitle("Data Connections are Turned Off")
             .setMessage("Would you like to be brought to the WiFi settings?")
             .setPositiveButton(android.R.string.yes, new DialogInterface.OnClickListener() {
                 public void onClick(DialogInterface dialog, int which) {
                    // if not so to the location settings
                          Intent intent = new Intent(Settings.ACTION_WIFI_SETTINGS);
                          startActivity(intent);
                 }
              })
             .setNegativeButton(android.R.string.no, new DialogInterface.OnClickListener() {
                 public void onClick(DialogInterface dialog, int which) {
                    Toast.makeText(ExampleLauncher.this, "We can't make a map without location
services", Toast.LENGTH_SHORT).show();

                 }
              })
             .setIcon(android.R.drawable.ic_dialog_alert)
              .show();
        }

        // check if GPS is enabled
        if (!enabled) {
              new AlertDialog.Builder(this)
             .setTitle("Location Services Are Turned Off")
             .setMessage("Would you like to be brought to the location settings?")
             .setPositiveButton(android.R.string.yes, new DialogInterface.OnClickListener() {
                 public void onClick(DialogInterface dialog, int which) {
                    // if not so to the location settings
                          Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                          startActivity(intent);
                 }
              })
             .setNegativeButton(android.R.string.no, new DialogInterface.OnClickListener() {
```

```
                    public void onClick(DialogInterface dialog, int which) {
                        Toast.makeText(ExampleLauncher.this, "We can't make a map without location
services", Toast.LENGTH_SHORT).show();

                    }
                })
                .setIcon(android.R.drawable.ic_dialog_alert)
                .show();
        }

        // Get the location manager
        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        // Define the criteria how to select the location provider
        Criteria criteria = new Criteria();
        provider = locationManager.getBestProvider(criteria, false);
        location = locationManager.getLastKnownLocation(provider);

        locationManager.requestLocationUpdates(provider, 400, 1, this);

        // location button
        this.findViewById(R.id.btn_get_location).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(final View pView) {

                // reset the API URL's in case the user has pressed the button twice
                resetAPIURLs();

                // get current location coordinates
                coordinates = new NortSouthEastWestBoundaries();

                // Display message
                Toast.makeText(ExampleLauncher.this, "Getting coordinates...",
Toast.LENGTH_SHORT).show();

                getCoordinates();
```

```
                        // check if a GPS location was determined
                        if (latCurrentPos == 0){
                              Toast.makeText(ExampleLauncher.this, "GPS detection failed\nMake sure you can
receive a GPS signal\nPlease try again", Toast.LENGTH_LONG).show();
                        }
                        else{
                              generateBoundaryBoxCoordinates();

                              // build up the two API URL's
                              generateApiUrlUsingCoordinates(minLatSouth);
                              generateApiUrlUsingCoordinates(minLonWest);
                              generateApiUrlUsingCoordinates(maxLatNorth);
                              buildApiUrlLastCoordinate(maxLonEast);

                              nodeApiUrl += commonPartOfApiUrl;
                              wayApiUrl += commonPartOfApiUrl;

                              System.out.println(nodeApiUrl);
                              System.out.println(wayApiUrl);
                              // start 2 threads to make the API call and download the map data
                              downloaderThread = null;
                           progressDialog = null;

//                         new DownloaderThread(this, nodeApiUrl, nodeFileName);
                              downloaderThread = new DownloaderThread(ExampleLauncher.this, nodeApiUrl,
nodeFileName);
                              downloaderThread.start();
                              downloaderThread = new DownloaderThread(ExampleLauncher.this, wayApiUrl,
wayFileName);
                              downloaderThread.start();
                        }
                  }
            });
```

```
            this.findViewById(R.id.btn_create_map).setOnClickListener(new OnClickListener() {
                  @Override
                  public void onClick(final View pView) {
                        boolean databaseExistsAlready = false;
                        // display message
                        Toast.makeText(ExampleLauncher.this, "Creating map, please.wait...",
Toast.LENGTH_SHORT).show();

                        // check if the db already exists
                        databaseExistsAlready = checkDataBase();
                        if (databaseExistsAlready == true ){
                              dbTools.deleteAllData();
                        }
                        // try to parse the XML map data files
                        try {
                              parseWayXML(wayFile);
                              Log.d(XMLTAG, "Finished parseEdgeXML();");
                              parseNodeXML(nodeFile);
                              Log.d(XMLTAG, "Finished parseNodeXML();");
                              Toast.makeText(ExampleLauncher.this, "BOOM! Map created, happy racing :)",
Toast.LENGTH_SHORT).show();
                        }

                        catch (XmlPullParserException e) {
                              Log.d(XMLTAG, "Error parsing XML files");
                              e.printStackTrace();
                              Toast.makeText(ExampleLauncher.this, "Sorry :(, there was an error creating
the map\n Determine your location first", Toast.LENGTH_SHORT).show();
                        }

                        catch (IOException e) {
                              Log.d(XMLTAG, "Error parsing XML files");
                              e.printStackTrace();
                              Toast.makeText(ExampleLauncher.this, "Sorry :(, there was an error creating
the map\n Determine your location first", Toast.LENGTH_SHORT).show();
```

```
                        }
                    }
                });
                // ===========================================================
                // My code - END
                // ===========================================================

                final SharedPreferences prefs = this.getPreferences(Context.MODE_PRIVATE);

                this.mVersionCodeCurrent = this.getVersionCode();
                this.mVersionCodeLastLaunch =
prefs.getInt(ExampleLauncher.PREF_LAST_APP_LAUNCH_VERSIONCODE_ID, -1);

                if(this.isFirstTime("first.app.launch")) {
                        this.showDialog(ExampleLauncher.DIALOG_FIRST_APP_LAUNCH);
                } else if((this.mVersionCodeLastLaunch != -1) && (this.mVersionCodeLastLaunch <
this.mVersionCodeCurrent)){
                        this.showDialog(ExampleLauncher.DIALOG_NEW_IN_THIS_VERSION);
                } else if(this.isFirstTime("please.submit.benchmarks")){
                        this.showDialog(ExampleLauncher.DIALOG_BENCHMARKS_SUBMIT_PLEASE);
                }

                prefs.edit().putInt(ExampleLauncher.PREF_LAST_APP_LAUNCH_VERSIONCODE_ID,
this.mVersionCodeCurrent).commit();
        }

        @Override
        public void onGroupExpand(final int pGroupPosition) {
                switch(this.mExpandableExampleLauncherListAdapter.getGroup(pGroupPosition)){
                }
                super.onGroupExpand(pGroupPosition);
        }

        @Override
```

```
	public boolean onChildClick(final ExpandableListView pParent, final View pV, final int
pGroupPosition, final int pChildPosition, final long pId) {
		final Example example = this.mExpandableExampleLauncherListAdapter.getChild(pGroupPosition,
pChildPosition);

		this.startActivity(new Intent(this, example.CLASS));

		return super.onChildClick(pParent, pV, pGroupPosition, pChildPosition, pId);
	}

	public boolean isFirstTime(final String pKey){
		final SharedPreferences prefs = this.getPreferences(Context.MODE_PRIVATE);
		if(prefs.getBoolean(pKey, true)){
			prefs.edit().putBoolean(pKey, false).commit();
			return true;
		}
		return false;
	}

	public int getVersionCode() {
		try {
			final PackageInfo pi = this.getPackageManager().getPackageInfo(this.getPackageName(), 0);
			return pi.versionCode;
		} catch (final PackageManager.NameNotFoundException e) {
			Debug.e("Package name not found", e);
			return -1;
		}
	}

	// ===========================================================
	// My code - START
	// ===========================================================

	// reset the API URL's
	private void resetAPIURLs() {
```

```
            // TODO Auto-generated method stub
            nodeApiUrl = "http://overpass.osm.rambler.ru/cgi/interpreter?data=node%28";
            wayApiUrl = "http://overpass.osm.rambler.ru/cgi/interpreter?data=way%28";
            commonPartOfApiUrl = "";
      }

      // retrieve the coordinates of the users location
      private void getCoordinates() {
            // Get the location manager
            locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
            // Define the criteria how to select the location provider
            Criteria criteria = new Criteria();
            provider = locationManager.getBestProvider(criteria, false);
            location = locationManager.getLastKnownLocation(provider);

            try {
                  // get the GPS coordinates from location
                  latCurrentPos = location.getLatitude();
                  lonCurrentPos = location.getLongitude();

//                // test Dubai coordinates
//                latCurrentPos = 24.975999;
//                lonCurrentPos = 55.036119;

//                // test IT Carlow coordinates
//                latCurrentPos = 52.826698;
//                lonCurrentPos = -6.935357;

//                // test Carlow Bourlum coordinates
//                latCurrentPos = 52.821946;
//                lonCurrentPos = -6.933501;

//                // test Wexford coordinates
//                latCurrentPos = 52.335456;
//                lonCurrentPos = -6.463159;
```

```
//                // test Wexford coordinates beside the crescent
//                latCurrentPos = 52.337016;
//                lonCurrentPos = -6.458841;

//                // test New York coordinates
//                latCurrentPos = 40.736013;
//                lonCurrentPos = -73.993592;

                  Log.d(LOGTAG, "GPS Detection Successful");

            } catch (Exception e) {
                  Log.d(LOGTAG, "GPS Detection Failed");
            }
      }

      // generate the Coordinates for the boundary box for the API
      private void generateBoundaryBoxCoordinates() {
            minLatSouth = latCurrentPos - latBoundary;
            minLonWest = lonCurrentPos - lonBoundary;
            maxLatNorth = latCurrentPos + latBoundary;
            maxLonEast = lonCurrentPos + lonBoundary;

            // store them in NortSouthEastWestBoundaries for retrieval later
            NortSouthEastWestBoundaries.setMaxLatNorth(maxLatNorth);
            NortSouthEastWestBoundaries.setMinLatSouth(minLatSouth);
            NortSouthEastWestBoundaries.setMaxLonEast(maxLonEast);
            NortSouthEastWestBoundaries.setMinLonWest(minLonWest);
      }

      // add Last Negative Coordinate To Common Api Url String
      private void addLastNegativeCoordinateToCommonApiUrlString(String tempCoordinateAsString) {
            commonPartOfApiUrl += asciiMinusValue;
            commonPartOfApiUrl += tempCoordinateAsString.substring(0,tempCoordinateAsString.indexOf("."));
            commonPartOfApiUrl += asciiFullStopValue;
```

```
            commonPartOfApiUrl += tempCoordinateAsString.substring(tempCoordinateAsString.indexOf(".") +
1,tempCoordinateAsString.length());
            commonPartOfApiUrl += endOfApiUrl;
      }

      // add Last Positive Coordinate To Common Api Url String
      private void addLastPositiveCoordinateToCommonApiUrlString(String tempCoordinateAsString) {
            commonPartOfApiUrl += tempCoordinateAsString.substring(0,tempCoordinateAsString.indexOf("."));
            commonPartOfApiUrl += asciiFullStopValue;
            commonPartOfApiUrl += tempCoordinateAsString.substring(tempCoordinateAsString.indexOf(".") +
1,tempCoordinateAsString.length());
            commonPartOfApiUrl += endOfApiUrl;
      }

      // Build the API URL
      private void buildApiUrlLastCoordinate(double originalCoordinate) {
            double tempCoordinate = 0;
            String tempCoordinateString = "";

            // check if the coordinate is negative
            if (originalCoordinate < 0) {

                  // make the negative coordinate positive
                  tempCoordinate = makeNegativeCoordinatePositive(originalCoordinate);
                  // convert coordinate to string
                  tempCoordinateString = coordinateToString(tempCoordinate);
                  // add it to the API URL
                  addLastNegativeCoordinateToCommonApiUrlString(tempCoordinateString);
            }
            // if the coordinate is positive
            else {
                  // convert coordinate to string
                  tempCoordinateString = coordinateToString(originalCoordinate);
                  // add it to the API URL
                  addLastPositiveCoordinateToCommonApiUrlString(tempCoordinateString);
```

```
        }
}

// generate the API URl
private void generateApiUrlUsingCoordinates(double originalCoordinate){

        double tempCoordinate = 0;
        String tempCoordinateString = "";

        // check if the coordinate is negative
        if (originalCoordinate < 0){
                // make the negative coordinate positive
                tempCoordinate = makeNegativeCoordinatePositive(originalCoordinate);
                // convert coordinate to string
                tempCoordinateString = coordinateToString(tempCoordinate);
                // add it to the API URL
                buildNegativeCommonApiUrlString(tempCoordinateString);
        }
        // if the coordinate is positive
        else{
                // convert coordinate to string
                tempCoordinateString = coordinateToString(originalCoordinate);
                // add it to the API URL
                buildPositiveCommonApiUrlString(tempCoordinateString);
        }
}

// convert a negative coordinate to a positive coordinate
private double makeNegativeCoordinatePositive(double negativeCoordinate){

        double positiveCoordinate = 0;
        positiveCoordinate = negativeCoordinate * -1;

        return positiveCoordinate;
}
```

```
	// convert a coordinate to a string
	private String coordinateToString(double convertThisCoordinate){

		String result = "";
		result = String.valueOf(convertThisCoordinate);

		return result;
	}

	// build the string with NEGATIVE coordinates
	private void buildNegativeCommonApiUrlString(String tempCoordinateAsString){

		commonPartOfApiUrl += asciiMinusValue;
		commonPartOfApiUrl += tempCoordinateAsString.substring(0,
tempCoordinateAsString.indexOf("."));
		commonPartOfApiUrl += asciiFullStopValue;
		commonPartOfApiUrl += tempCoordinateAsString.substring(tempCoordinateAsString.indexOf(".") +
1, tempCoordinateAsString.length());
		commonPartOfApiUrl += asciiCommaValue;
	}

	// build the string with NEGATIVE coordinates
	private void buildPositiveCommonApiUrlString(String tempCoordinateAsString) {
		commonPartOfApiUrl += tempCoordinateAsString.substring(0,
tempCoordinateAsString.indexOf("."));
		commonPartOfApiUrl += asciiFullStopValue;
		commonPartOfApiUrl += tempCoordinateAsString.substring(tempCoordinateAsString.indexOf(".") +
1, tempCoordinateAsString.length());
		commonPartOfApiUrl += asciiCommaValue;
	}

	// parse node XML file
	public void parseNodeXML(String fileName) throws XmlPullParserException, IOException {
```

```
// variables
String temporaryParsedData = "";
long nodeId = 0;
double lat = 0;
double lon = 0;
int index = 0;
long tempNodeId = 0;
int tableSize = 0;
ArrayList<Long> validHighwayNodesAl = new ArrayList<Long>();

// create the XMLPullParser
XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
factory.setNamespaceAware(true);
XmlPullParser xpp = factory.newPullParser();

// get a reference to the file.
File file = new File(Environment.getExternalStorageDirectory() + fileName);

// create an input stream to be read by the stream reader.
FileInputStream fis = new FileInputStream(file);

// set the input for the parser using an InputStreamReader
xpp.setInput(new InputStreamReader(fis));

// set an int to store the event type// start the XMLPullParser
int eventType = xpp.getEventType();

// retrieve a list of all the valid highway nodes
validHighwayNodesAl = dbTools.getAllValidHighwayNodes();
tableSize = validHighwayNodesAl.size();
      // start the XMLPullParser
      while (eventType != XmlPullParser.END_DOCUMENT){

            // check for a start tag
            if (eventType == XmlPullParser.START_TAG){
```

```
                        // get the value from a node tag
                        if (xpp.getName().equals(NODE)){
                              temporaryParsedData = xpp.getAttributeValue(null, NODE_ID);
                              nodeId = Long.parseLong(temporaryParsedData);

                              // send the node value to the db to see if it exists in the db
                              tempNodeId = dbTools.getValidNodeId(nodeId);

                              // if tempNode is in the db
                              if (nodeId == tempNodeId){

                                    // get the latitude and longitude
                                    temporaryParsedData = xpp.getAttributeValue(null, LAT);
                                    lat = Double.parseDouble(temporaryParsedData);
                                    temporaryParsedData = xpp.getAttributeValue(null, LON);
                                    lon = Double.parseDouble(temporaryParsedData);

                                    // send values to the db
                                    dbTools.insertNode(nodeId, lat, lon);
                              }
                        }
                  }
                  // get the next eventType
                  eventType = xpp.next();
            }
            index ++;
}

// parse edge XML file
public void parseWayXML(String fileName) throws XmlPullParserException, IOException {
      // variables
      String temporaryParsedData = "";
      long nodeA = 0;
      long nodeB = 0;
```

```
long tempNode = 0;
long wayId = 0;
int index = 1;
boolean newWayStarted = false;
boolean highwayFound = false;
ArrayList<Integer> indexAl = new ArrayList<Integer>();
ArrayList<Long> wayIdAl = new ArrayList<Long>();
ArrayList<Long> nodeAAl = new ArrayList<Long>();
ArrayList<Long> nodeBAl = new ArrayList<Long>();
ArrayList<Long> validHighwayIds = new ArrayList<Long>();
ArrayList<Long> validHighwayNodes = new ArrayList<Long>();

// create the XMLPullParser
XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
factory.setNamespaceAware(true);
XmlPullParser xpp = factory.newPullParser();

// get a reference to the file.
File file = new File(Environment.getExternalStorageDirectory() + fileName);

// create an input stream to be read by the stream reader.
FileInputStream fis = new FileInputStream(file);

// set the input for the parser using an InputStreamReader
xpp.setInput(new InputStreamReader(fis));

// set an int to store the event type
int eventType = xpp.getEventType();

// start the XMLPullParser
while (eventType != XmlPullParser.END_DOCUMENT){

      // check for a start tag
      if (eventType == XmlPullParser.START_TAG){
```

```
// get the value from a way tag and set newWayStarted flag to true
if (xpp.getName().equals(WAY)){
      newWayStarted = true;
      temporaryParsedData = xpp.getAttributeValue(null, WAY_ID);
      wayId = Long.parseLong(temporaryParsedData);
}

// get the value of the nd tag
if (xpp.getName().equals(ND)){

      // check if its the first node in the road and get the value if it is
      if ((newWayStarted == true) && (tempNode == 0)){
            temporaryParsedData = xpp.getAttributeValue(null, REF);
            tempNode = Long.parseLong(temporaryParsedData);

            // set this value to nodeA and add to validHighwayNodes
            nodeA = tempNode;
            validHighwayNodes.add(nodeA);

            // set wayId to validHighwayIds
            validHighwayIds.add(wayId);
      }

      // if its not the first node in a road
      else{
            temporaryParsedData = xpp.getAttributeValue(null, REF);
            tempNode = Long.parseLong(temporaryParsedData);

            // set this value to nodeB and add to validHighwayNodes
            nodeB = tempNode;
            validHighwayNodes.add(nodeB);

            // set wayId to validHighwayIds
            validHighwayIds.add(wayId);
      }
```

```
                // check if nodaA and nodeB have values
                if ((nodeA != 0) && (nodeB != 0)){

                      // use as an index
                      index ++;

                      // add the values retrieved to the array lists
                      indexAl.add(index);
                      wayIdAl.add(wayId);
                      nodeAAl.add(nodeA);
                      nodeBAl.add(nodeB);

                      // set nodeA to nodeB
                      nodeA = nodeB;
                }
          }

          // check the value from the TAG tag
          if (xpp.getName().equals(TAG)){

                temporaryParsedData = xpp.getAttributeValue(null, K);

                // if a highway is found set highwayFound flag to true
                if (temporaryParsedData.equalsIgnoreCase(HIGHWAY)){

                      highwayFound = true;
                }
          }
    }

    // check for an end tag
    else if (eventType == XmlPullParser.END_TAG){

          // if a way end tag is found
```

```
if (xpp.getName().equals(WAY)){

      // check if highwayFound is true
      if (highwayFound == true){

            // send all of the valued to the databas
            dbTools.insertWayArrayList(indexAl, wayIdAl, nodeAAl, nodeBAl);
            dbTools.insertValidHighwayNodes(validHighwayIds,validHighwayNodes);
            dbTools.insertSingleWayId(wayId);

            // clear the array lists
            indexAl.clear();
            wayIdAl.clear();
            nodeAAl.clear();
            nodeBAl.clear();
            validHighwayIds.clear();
            validHighwayNodes.clear();
      }
      // clear the array lists
      else {
            indexAl.clear();
            wayIdAl.clear();
            nodeAAl.clear();
            nodeBAl.clear();
            validHighwayIds.clear();
            validHighwayNodes.clear();
      }

      // reset all values
      highwayFound = false;
      newWayStarted = false;
      wayId = 0;
      tempNode = 0;
      nodeA = 0;
      nodeB = 0;
```

```
                    }
                }
                // get the next eventType
                eventType = xpp.next();
            }
        }

        // check if a db already exists
        private boolean checkDataBase() {
            SQLiteDatabase database = null;
            boolean exists = false;
            try {
                database = SQLiteDatabase.openDatabase(DB_FULL_PATH, null,
                        SQLiteDatabase.OPEN_READONLY);
                database.close();
                exists = true;
            } catch (SQLiteException e) {
                // error if the db doesnt exist
              System.out.println("db doesn't exist");
              exists = false;
            }
            return exists;
        }

        @Override
        public void onLocationChanged(Location location) {
            latCurrentPos = location.getLatitude();
              lonCurrentPos = location.getLongitude();
        }

        private boolean isNetworkAvailable() {
            ConnectivityManager connectivityManager
                  = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
            return activeNetworkInfo != null && activeNetworkInfo.isConnected();
```

```
}
// ============================================================
// My code - END
// ============================================================

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
	// TODO Auto-generated method stub

}

@Override
public void onProviderEnabled(String provider) {
	// TODO Auto-generated method stub

}

@Override
public void onProviderDisabled(String provider) {
	// TODO Auto-generated method stub

}

/**
 * If there is a progress dialog, dismiss it and set progressDialog to
 * null.
 */
public void dismissCurrentProgressDialog()
{
	if(progressDialog != null)
	{
		progressDialog.hide();
		progressDialog.dismiss();
		progressDialog = null;
	}
```

```
	}

	/**
	 * Displays a message to the user, in the form of a Toast.
	 * @param message Message to be displayed.
	 */
	public void displayMessage(String message)
	{
		if(message != null)
		{
			Toast.makeText(thisActivity, message, Toast.LENGTH_SHORT).show();
		}
	}
}
```

3. RacerGameActivity.java

This file was originally developed by Nicolas Gramlich and modified by Philip Stafford. The code can be found in Nicolas's Github repository. (Gramlich, 2010)

All of my code is clearly marked within the code.

```
package org.andengine.examples.game.racer;

import java.util.ArrayList;
import java.util.HashMap;

import org.andengine.engine.camera.Camera;
import org.andengine.engine.camera.hud.controls.AnalogOnScreenControl;
import org.andengine.engine.camera.hud.controls.AnalogOnScreenControl.IAnalogOnScreenControlListener;
import org.andengine.engine.camera.hud.controls.BaseOnScreenControl;
import org.andengine.engine.options.EngineOptions;
import org.andengine.engine.options.ScreenOrientation;
import org.andengine.engine.options.resolutionpolicy.RatioResolutionPolicy;
import org.andengine.entity.primitive.Line;
import org.andengine.entity.scene.Scene;
import org.andengine.entity.scene.background.Background;
import org.andengine.entity.sprite.TiledSprite;
import org.andengine.entity.util.FPSLogger;
import org.andengine.extension.physics.box2d.FixedStepPhysicsWorld;
import org.andengine.extension.physics.box2d.PhysicsConnector;
import org.andengine.extension.physics.box2d.PhysicsFactory;
import org.andengine.extension.physics.box2d.PhysicsWorld;
import org.andengine.extension.physics.box2d.util.Vector2Pool;
import org.andengine.opengl.texture.TextureOptions;
import org.andengine.opengl.texture.atlas.bitmap.BitmapTextureAtlas;
import org.andengine.opengl.texture.atlas.bitmap.BitmapTextureAtlasTextureRegionFactory;
import org.andengine.opengl.texture.region.ITextureRegion;
import org.andengine.opengl.texture.region.TiledTextureRegion;
import org.andengine.opengl.vbo.VertexBufferObjectManager;
```

```
import org.andengine.ui.activity.SimpleBaseGameActivity;
import org.andengine.util.math.MathUtils;

import android.opengl.GLES20;
import android.util.Log;

import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef.BodyType;
import com.badlogic.gdx.physics.box2d.FixtureDef;
import com.locationracing.app.*;

/**
 * (c) 2010 Nicolas Gramlich
 * (c) 2011 Zynga
 *
 * @author Nicolas Gramlich
 * @since 22:43:20 - 15.07.2010
 */
public class RacerGameActivity extends SimpleBaseGameActivity {
	// ===========================================================
	// Constants
	// ===========================================================
	private static final int CAR_SIZE = 16;

	// ===========================================================
	// My code - START
	// ===========================================================
//	private static final int CAMERA_WIDTH = 3200;
//	private static final int CAMERA_HEIGHT = 1920;

	private static final int CAMERA_WIDTH = 800;
	private static final int CAMERA_HEIGHT = 480;
	// ===========================================================
	// My code - END
```

```
	// ===========================================================

	// ===========================================================
	// Fields
	// ===========================================================

	private Camera mCamera;
//	private BoundCamera mCamera;

	private BitmapTextureAtlas mVehiclesTexture;
	private TiledTextureRegion mVehiclesTextureRegion;

	private BitmapTextureAtlas mBoxTexture;

	private BitmapTextureAtlas mRacetrackTexture;

	private BitmapTextureAtlas mOnScreenControlTexture;
	private ITextureRegion mOnScreenControlBaseTextureRegion;
	private ITextureRegion mOnScreenControlKnobTextureRegion;

	private Scene mScene;

	private PhysicsWorld mPhysicsWorld;

	private Body mCarBody;
	private TiledSprite mCar;

	// ===========================================================
	// My code - START
	// ===========================================================

	// debug tag
	private static final String TAGRACER = "[Racer]";
	// for the track creation method
	private static final String WAY_ID = "way_id";
```

```
	private static final String NODE_A = "node_a";
	private static final String NODE_B = "node_b";
	private static final String NODE_ID = "node_id";

	// boundary coordinates
	double maxLatNorth = 0;
	double minLatSouth = 0;
	double maxLonEast = 0;
	double minLonWest = 0;

	// create an instance of DBTools
	DBTools dbTools = new DBTools(this);
	// ============================================================
	// My code - END
	// ============================================================

	@Override
	public EngineOptions onCreateEngineOptions() {
		this.mCamera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);

		return new EngineOptions(true, ScreenOrientation.LANDSCAPE_FIXED, new
RatioResolutionPolicy(CAMERA_WIDTH, CAMERA_HEIGHT), this.mCamera);
	}

	@Override
	public void onCreateResources() {
		BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("gfx/");

		this.mVehiclesTexture = new BitmapTextureAtlas(this.getTextureManager(), 128, 16,
TextureOptions.BILINEAR);
		this.mVehiclesTextureRegion =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(this.mVehiclesTexture, this, "vehicles.png",
0, 0, 6, 1);
		this.mVehiclesTexture.load();
```

```
		this.mRacetrackTexture = new BitmapTextureAtlas(this.getTextureManager(), 128, 256,
TextureOptions.REPEATING_NEAREST);
		this.mRacetrackTexture.load();

		this.mOnScreenControlTexture = new BitmapTextureAtlas(this.getTextureManager(), 256, 128,
TextureOptions.BILINEAR);
		this.mOnScreenControlBaseTextureRegion =
BitmapTextureAtlasTextureRegionFactory.createFromAsset(this.mOnScreenControlTexture, this,
"onscreen_control_base.png", 0, 0);
		this.mOnScreenControlKnobTextureRegion =
BitmapTextureAtlasTextureRegionFactory.createFromAsset(this.mOnScreenControlTexture, this,
"onscreen_control_knob.png", 128, 0);
		this.mOnScreenControlTexture.load();

		this.mBoxTexture = new BitmapTextureAtlas(this.getTextureManager(), 32, 32,
TextureOptions.BILINEAR);
		this.mBoxTexture.load();
	}

	@Override
	public Scene onCreateScene() {
		this.mEngine.registerUpdateHandler(new FPSLogger());
		this.mScene = new Scene();
		// set background colour
		this.mScene.setBackground(new Background(0.09804f, 0.6274f, 0.8784f));

		this.mPhysicsWorld = new FixedStepPhysicsWorld(30, new Vector2(0, 0), false, 8, 1);

		this.initRacetrack();
		this.initCar();
		this.initOnScreenControls();

		this.mScene.registerUpdateHandler(this.mPhysicsWorld);

		return this.mScene;
```

```
	}

	@Override
	public void onGameCreated() {

	}

	// ===========================================================
	// Methods
	// ===========================================================

	private void initOnScreenControls() {
		final AnalogOnScreenControl analogOnScreenControl = new AnalogOnScreenControl(0, CAMERA_HEIGHT
- this.mOnScreenControlBaseTextureRegion.getHeight(), this.mCamera,
this.mOnScreenControlBaseTextureRegion, this.mOnScreenControlKnobTextureRegion, 0.1f,
this.getVertexBufferObjectManager(), new IAnalogOnScreenControlListener() {
			@Override
			public void onControlChange(final BaseOnScreenControl pBaseOnScreenControl, final float
pValueX, final float pValueY) {
				final Body carBody = RacerGameActivity.this.mCarBody;

				final Vector2 velocity = Vector2Pool.obtain(pValueX * 5, pValueY * 5);
				carBody.setLinearVelocity(velocity);
				Vector2Pool.recycle(velocity);

				final float rotationInRad = (float)Math.atan2(-pValueX, pValueY);
				carBody.setTransform(carBody.getWorldCenter(), rotationInRad);

				RacerGameActivity.this.mCar.setRotation(MathUtils.radToDeg(rotationInRad));
			}

			@Override
			public void onControlClick(final AnalogOnScreenControl pAnalogOnScreenControl) {
				/* Nothing. */
			}
```

```
            });
            analogOnScreenControl.getControlBase().setBlendFunction(GLES20.GL_SRC_ALPHA,
GLES20.GL_ONE_MINUS_SRC_ALPHA);
            analogOnScreenControl.getControlBase().setAlpha(0.5f);
            //          analogOnScreenControl.getControlBase().setScaleCenter(0, 128);
            //          analogOnScreenControl.getControlBase().setScale(0.75f);
            //          analogOnScreenControl.getControlKnob().setScale(0.75f);
            analogOnScreenControl.refreshControlKnobPosition();

            this.mScene.setChildScene(analogOnScreenControl);
      }

      private void initCar() {
            this.mCar = new TiledSprite(20, 20, CAR_SIZE, CAR_SIZE, this.mVehiclesTextureRegion,
this.getVertexBufferObjectManager());
            this.mCar.setCurrentTileIndex(0);

            final FixtureDef carFixtureDef = PhysicsFactory.createFixtureDef(1, 0.5f, 0.5f);
            this.mCarBody = PhysicsFactory.createBoxBody(this.mPhysicsWorld, this.mCar,
BodyType.DynamicBody, carFixtureDef);

            this.mPhysicsWorld.registerPhysicsConnector(new PhysicsConnector(this.mCar, this.mCarBody,
true, false));

            // ===========================================================
            // My code - START
            // ===========================================================
            // set the camera to chase the car
//          this.mCamera.setChaseEntity(mCar);
            // ===========================================================
            // My code - END
            // ===========================================================

            this.mScene.attachChild(this.mCar);
      }
```

```
private void initRacetrack() {
	// ============================================================
	// My code - START
	// ============================================================
	// variables
	ArrayList<HashMap<String, String>> wayArrayList;
	int wayArrayListSize = 0;
	String tempNodeAString = "" ;
	String tempNodeBString = "" ;
	long nodeAasLong = 0;
	long nodeBasLong = 0;

	String tempEntryIdAsString = "";
	String tempWayIdAsString = "";

	int entryIdAsLong = 0;
	long wayIdAsLong = 0;

	boolean startOfNewRoad = true;

	HashMap<String, String> nodeAInfo;
	HashMap<String, String> nodeBInfo;
	int nodeCount = 0;
	double nodeAX = 0;
	double nodeAY = 0;
	double nodeBX = 0;
	double nodeBY = 0;
	double calculatedAXDouble = 0;
	double calculatedAYDouble = 0;
	double calculatedBXDouble = 0;
	double calculatedBYDouble = 0;
	float calculatedAXFloat = 0;
	float calculatedAYFloat = 0;
	float calculatedBXFloat = 0;
```

```
            float calculatedBYFloat = 0;

            double angleOfLine = 0;
            double lengthOfLine = 0;

            // trying to calculate the lines for each side of the road
//          double parallelAXDouble = 0;
//          double parallelAYDouble = 0;
//          double parallelBXDouble = 0;
//          double parallelBYDouble = 0;
//          float parallelDifferenceAXFloat = 0;
//          float parallelDifferenceAYFloat = 0;
//          float parallelDifferenceBXFloat = 0;
//          float parallelDifferenceBYFloat = 0;
//          float tcalculatedAXFloat = 0;
//          float tcalculatedBXFloat = 0;
//          float tcalculatedAYFloat = 0;
//          float tcalculatedBYFloat = 0;

            wayArrayList = dbTools.getAllWays();
            wayArrayListSize = wayArrayList.size();
            getMinMaxBoundaries();

            // begin drawing map
            final VertexBufferObjectManager vertexBufferObjectManager =
this.getVertexBufferObjectManager();
            for(int i = 0; i < wayArrayListSize; i++) {

                  // the first edge of a new road is started
                  if (startOfNewRoad == true){
                        // get the wayId to calculate when an new road ends or starts
                        tempWayIdAsString = wayArrayList.get(i).get(WAY_ID);
                        wayIdAsLong = Long.parseLong(tempWayIdAsString);

                        // get the id of the 2 nodes which are joined to make an edge
```

```
                    tempNodeAString = wayArrayList.get(i).get(NODE_A);
                    nodeAasLong = Long.parseLong(tempNodeAString);

                    tempNodeBString = wayArrayList.get(i).get(NODE_B);
                    nodeBasLong = Long.parseLong(tempNodeBString);

                    startOfNewRoad = false;
              }
              // if this is not the first or last edge in a road
              else if (i+1 < wayArrayList.size() &&
wayArrayList.get(i).get(WAY_ID).equalsIgnoreCase(wayArrayList.get(i+1).get(WAY_ID)) == true){
                    // get the wayId to calculate when an new road ends or starts
                    tempWayIdAsString = wayArrayList.get(i).get(WAY_ID);
                    wayIdAsLong = Long.parseLong(tempWayIdAsString);

                    // set nodeB to nodeA
                    nodeAasLong = nodeBasLong;

                    // get the id of the next node
                    tempNodeBString = wayArrayList.get(i).get(NODE_B);
                    nodeBasLong = Long.parseLong(tempNodeBString);
              }
              // the last edge of a road
              else{
                    // reset the new road flag
                    startOfNewRoad = true;

                    // set nodeB to nodeA
                    nodeAasLong = nodeBasLong;

                    // get the wayId to calculate when an new road ends or starts
                    tempWayIdAsString = wayArrayList.get(i).get(WAY_ID);
                    wayIdAsLong = Long.parseLong(tempWayIdAsString);

                    // get the id of the next node
```

```
                tempNodeBString = wayArrayList.get(i).get(NODE_B);
                nodeBasLong = Long.parseLong(tempNodeBString);
            }
            // send the nodeA and nodeB id to the db to retrieve the node info
            nodeAInfo = dbTools.getNodeInfo(NODE_ID,nodeAasLong);
            nodeBInfo = dbTools.getNodeInfo(NODE_ID,nodeBasLong);

            // if the node exists in the db
            if (nodeAInfo.get(NODE_ID) != null){
                // get the node info
                String tempLat = nodeAInfo.get("latitude");
                String tempLon = nodeAInfo.get("longitude");

                // parse the info to a double
                nodeAX = Double.parseDouble(tempLat);
                nodeAY = Double.parseDouble(tempLon);

                // if the info is within the boundaries
                if((nodeAX <= maxLatNorth && nodeAX >= minLatSouth) && (nodeAY <= maxLonEast &&
nodeAY >= minLonWest)) {

                    // calculate X and Y coordinates
                    calculatedAXDouble = ((nodeAX - minLatSouth)/(maxLatNorth - minLatSouth) *
3200);
                calculatedAYDouble = ((nodeAY - minLonWest)/(maxLonEast - minLonWest) * 1920);

                // parse the info to a double
                calculatedAXFloat = (float)calculatedAXDouble;
                calculatedAYFloat = (float)calculatedAYDouble;
                }
            }

            // if the node exists in the db
            if (nodeBInfo.get(NODE_ID) != null){
                // get the node info
```

```
                    String tempLat = nodeBInfo.get("latitude");
                    String tempLon = nodeBInfo.get("longitude");

                    // parse the info to a double
                    nodeBX = Double.parseDouble(tempLat);
                    nodeBY = Double.parseDouble(tempLon);

                    // if the info is within the boundaries
                    if((nodeBX <= maxLatNorth && nodeBX >= minLatSouth) && (nodeBY <= maxLonEast &&
nodeBY >= minLonWest)) {

                         // calculate X and Y coordinates
                         calculatedBXDouble = ((nodeBX - minLatSouth)/(maxLatNorth - minLatSouth) *
3200);
                    calculatedBYDouble = ((nodeBY - minLonWest)/(maxLonEast - minLonWest) * 1920);

                    // parse the info to a double
                    calculatedBXFloat = (float)calculatedBXDouble;
                    calculatedBYFloat = (float)calculatedBYDouble;
                    }
               }

               // trying to calculate the lines for each side of the road
               // if all of the X and Y coordinates have a value
//             if (calculatedAXDouble != 0 && calculatedAYDouble != 0 && calculatedBXDouble != 0 &&
calculatedBYDouble != 0){
//
//                  //
//                  lengthOfLine = getLengthOfLineBetweenPoints(calculatedAXDouble, calculatedAYDouble,
calculatedBXDouble, calculatedBYDouble);
//                  angleOfLine = getAngleOfLineBetweenPoints(calculatedAXDouble, calculatedAYDouble,
calculatedBXDouble, calculatedBYDouble);
//
//                  parallelAXDouble = lengthOfLine * Math.cos(angleOfLine);
//                  parallelAYDouble = lengthOfLine*Math.sin(angleOfLine);
```

```
//                    parallelBXDouble = (lengthOfLine * -1) * Math.cos(angleOfLine);
//                    parallelBYDouble = (lengthOfLine * -1) * Math.sin(angleOfLine);
//
//                    parallelDifferenceAXFloat = (float)parallelAXDouble;
//                    parallelDifferenceAYFloat = (float)parallelAYDouble;
//                    parallelDifferenceBXFloat = (float)parallelBXDouble;
//                    parallelDifferenceBYFloat = (float)parallelBYDouble;
//                }

                if (calculatedAXFloat != 0 && calculatedAYFloat != 0 && calculatedBXFloat != 0 &&
calculatedBYFloat != 0){
                    // set the calculated X and Y coordinates for the line drawing method
                    final float x1 = calculatedAXFloat;
                    final float x2 = calculatedBXFloat;
                    final float y1 = calculatedAYFloat;
                    final float y2 = calculatedBYFloat;
                    final float lineWidth = 1;

                    // draw a line
                    final Line line = new Line(x1, y1, x2, y2, lineWidth, vertexBufferObjectManager);

                    // set the colour
                    line.setColor(1,1,1);

                    // attach it to the scene
                    mScene.attachChild(line);

                    // trying to calculate the lines for each side of the road
//                    tcalculatedAXFloat = calculatedAXFloat - (parallelDifferenceAXFloat);
//                    tcalculatedAYFloat = calculatedAYFloat - (parallelDifferenceAYFloat);
//                    tcalculatedBXFloat = calculatedBXFloat - (parallelDifferenceBXFloat);
//                    tcalculatedBYFloat = calculatedBYFloat - (parallelDifferenceAYFloat);
                }

                // trying to calculate the lines for each side of the road
```

```
//              if (tcalculatedAXFloat != 0 && tcalculatedAYFloat != 0 && tcalculatedBXFloat != 0 &&
tcalculatedBYFloat != 0){
//                  final float x1 = tcalculatedAXFloat;
//                  final float x2 = tcalculatedBXFloat;
//                  final float y1 = tcalculatedAYFloat;
//                  final float y2 = tcalculatedBYFloat;
//                  final float lineWidth = 1;
//
//                  final Line line = new Line(x1, y1, x2, y2, lineWidth, vertexBufferObjectManager);
//
//                  line.setColor(1,1,1);
//
//                  mScene.attachChild(line);
//
//                  System.out.println("Top x1= "+tcalculatedAXFloat+" y1= "+tcalculatedAYFloat +" x2=
"+tcalculatedBXFloat +" y2= "+tcalculatedBYFloat);
//
//                  tcalculatedAXFloat = calculatedAXFloat + (parallelDifferenceAXFloat);
//                  tcalculatedAYFloat = calculatedAYFloat + (parallelDifferenceAYFloat);
//                  tcalculatedBXFloat = calculatedBXFloat + (parallelDifferenceBXFloat);
//                  tcalculatedBYFloat = calculatedBYFloat + (parallelDifferenceAYFloat);
//              }
//
//              if (tcalculatedAXFloat != 0 && tcalculatedAYFloat != 0 && tcalculatedBXFloat != 0 &&
tcalculatedBYFloat != 0){
//                  final float x1 = calculatedAXFloat;
//                  final float y1 = calculatedAYFloat;
//                  final float x2 = calculatedBXFloat;
//                  final float y2 = calculatedBYFloat;
//                  final float lineWidth = 1;
//
//                  final Line line = new Line(x1, y1, x2, y2, lineWidth, vertexBufferObjectManager);
//
//                  line.setColor(1,1,1);
//
```

```
//                    mScene.attachChild(line);
//                }
                  // reset all these variables
                  calculatedAXFloat = 0;
                  calculatedBXFloat = 0;
                  calculatedAYFloat = 0;
                  calculatedBYFloat = 0;

                  calculatedAXDouble = 0;
                  calculatedAYDouble = 0;
                  calculatedBXDouble = 0;
                  calculatedBYDouble = 0;

                  // trying to calculate the lines for each side of the road
//                parallelDifferenceAXFloat = 0;
//                parallelDifferenceAYFloat = 0;
//                parallelDifferenceBXFloat = 0;
//                parallelDifferenceBYFloat = 0;
            }
      }

      // trying to calculate the lines for each side of the road
//    private double getLengthOfLineBetweenPoints(double x1,double y1,double x2,double y2) {
//
//          double lineLength = 0;
//          double x2MinusX1 = 0;
//          double y2MinusY1 = 0;
//          double x2MinusX1Minusy2MinusY1 = 0;
//          x2MinusX1 = x2-x1;
//          y2MinusY1 = y2-y1;
//          x2MinusX1Minusy2MinusY1 = x2MinusX1 - y2MinusY1;
//
//          if (x2MinusX1Minusy2MinusY1 < 0){
//                x2MinusX1Minusy2MinusY1 = x2MinusX1Minusy2MinusY1 * -1;
//          }
```

```
//
//          lineLength = Math.sqrt(x2MinusX1Minusy2MinusY1);
//          return lineLength;
//    }

      // Angle of line
      /** * Determines the angle of a straight line drawn between point one and two.
       * The number returned, which is a double in degrees, tells us how much we have
       * to rotate a horizontal line clockwise for it to match the line between the two
       * points. * If you prefer to deal with angles using radians instead of degrees,
       * just change the last line to: "return Math.atan2(yDiff, xDiff);" */
//    public static double getAngleOfLineBetweenPoints(double x1,double y1,double x2,double y2) {
//          double xDiff = x2 - x1;
//          double yDiff = y2 - y1;
//          return Math.toDegrees(Math.atan2(yDiff, xDiff));
//    }

      // ============================================================
      // generate the boundaries of the area to download
      // ============================================================
      private void getMinMaxBoundaries(){
            maxLatNorth = NortSouthEastWestBoundaries.getMaxLatNorth();
            minLatSouth = NortSouthEastWestBoundaries.getMinLatSouth();
            maxLonEast = NortSouthEastWestBoundaries.getMaxLonEast();
            minLonWest = NortSouthEastWestBoundaries.getMinLonWest();

            Log.d(TAGRACER, "maxLatNorth = " + Double.toString(maxLatNorth));
            Log.d(TAGRACER, "minLatSouth = " + Double.toString(minLatSouth));
            Log.d(TAGRACER, "maxLonEast = " + Double.toString(maxLonEast));
            Log.d(TAGRACER, "minLonWest = " + Double.toString(minLonWest));
      }
      // ============================================================
      // My code - END
      // ============================================================
}
```

4. DBTools.java

All of the code in this file was created by Philip Stafford.

```
package com.locationracing.app;
//
//public class DBTools {
//
//}
//DBTools.java
import java.util.ArrayList;
import java.util.HashMap;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

//SQLiteOpenHelper helps you open or create a database

public class DBTools  extends SQLiteOpenHelper {

	// constants
	private static final String DBTAG = "[DB Tools]";
	private static final String TAGNULL = "[NULL]";

	// variables
	private int nodeInsertIndex = 1;
	private int wayInsertIndex = 1;
	private int validHighwayInsertIndex = 1;
	private int singleWayIdIndex = 1;
```

```
	// Context : provides access to application-specific resources and classes
	public DBTools(Context applicationcontext) {

		// Call use the database or to create it
		super(applicationcontext, "mapdata.db", null, 1);
	}

	// onCreate is called the first time the database is created
	public void onCreate(SQLiteDatabase database) {

		// create tables and fields
		String createNodeTable = "CREATE TABLE nodes ( entry_id INTEGER PRIMARY KEY, node_id INTEGER,
latitude REAL, longitude REAL)";
		String createWayTable = "CREATE TABLE ways ( entry_id INTEGER PRIMARY KEY, way_id INTEGER,
node_a INTEGER, node_b INTEGER)";
		String createValidNodeTable = "CREATE TABLE valid_highway_nodes ( entry_id INTEGER PRIMARY
KEY, valid_highway_id INTEGER, valid_node INTEGER)";
		String createSingleWayTable = "CREATE TABLE single_ways ( entry_id INTEGER PRIMARY KEY, way_id
INTEGER)";

		// execute the SQL statements
		database.execSQL(createNodeTable);
		database.execSQL(createWayTable);
		database.execSQL(createValidNodeTable);
		database.execSQL(createSingleWayTable);

//		// close the db
//		database.close();
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
	}
```

```
	public void deleteAllData() {

		System.out.println("in deleteAllData()");

		SQLiteDatabase database = this.getWritableDatabase();
		// delete data if db already exists
		database.execSQL("delete from nodes");
		database.execSQL("delete from ways");
		database.execSQL("delete from valid_highway_nodes");
		database.execSQL("delete from single_ways");

//		// close the db
//		database.close();
	}

	// add a nodeId, latitude and longitude to the db
	public void insertNode(long nodeId, double lat, double lon) {

		// open db to read or write
		SQLiteDatabase database = this.getWritableDatabase();

		// Stores key value pairs being the column name and the data
		// ContentValues data type is needed because the database
		// requires its data type to be passed
		ContentValues values = new ContentValues();

		// put the values into value
		values.put("entry_id", nodeInsertIndex);
		values.put("node_id", nodeId);
		values.put("latitude", lat);
		values.put("longitude", lon);

		// insert data into db nodes
		database.insert("nodes", null, values);
```

```
            // increment the index - use it as primary key
            nodeInsertIndex ++;

            // close the db
            database.close();
      }

      public void deleteNode(String id) {
            // Open a database for reading and writing

            SQLiteDatabase database = this.getWritableDatabase();

            String deleteQuery = "DELETE FROM  nodes where node_id='"+ id +"'";

            // Executes the query provided as long as the query isn't a select
            // or if the query doesn't return any data
            database.execSQL(deleteQuery);

//          // close the db
//          database.close();
      }

      // get array list of all valid highway nodes
      public ArrayList<Long> getAllValidHighwayNodes() {

            // variables
            ArrayList<Long> nodeArrayList = new ArrayList<Long>();

            // SQL query
            String selectQuery = "SELECT valid_node FROM valid_highway_nodes";

            // open db to read or write
           SQLiteDatabase database = this.getWritableDatabase();

           // Cursor provides read and write access for the
```

```
        // data returned from a database query
        // rawQuery executes the query and returns the result as a Cursor
        Cursor cursor = database.rawQuery(selectQuery, null);

        // Move to the first row
        if (cursor.moveToFirst()) {
            do {
               // Access the Cursor data by index
               nodeArrayList.add(cursor.getLong(0));

            } while (cursor.moveToNext()); // Move Cursor to the next row
        }

//        // close the db
//         database.close();
        return nodeArrayList;
    }

    // get info for specific node
    public HashMap<String, String> getNodeInfo(String columnName, long idNum) {
         HashMap<String, String> nodeMap = new HashMap<String, String>();

         // open db to read or write
         SQLiteDatabase database = this.getReadableDatabase();

         // SQL query
         String selectQuery = "SELECT * FROM nodes where "+ columnName + "='"+idNum+"'";

         // Cursor provides read and write access for the
        // data returned from a database query
         // rawQuery executes the query and returns the result as a Cursor
         Cursor cursor = database.rawQuery(selectQuery, null);
         if (cursor.moveToFirst()) {
            do {
               nodeMap.put("entry_id", cursor.getString(0));
```

```
                nodeMap.put("node_id", cursor.getString(1));
                nodeMap.put("latitude", cursor.getString(2));
                nodeMap.put("longitude", cursor.getString(3));

            } while (cursor.moveToNext());
        }

        // close the db
        database.close();
        return nodeMap;
    }

    // insert info about ways
    public void insertWayArrayList(ArrayList<Integer> indexAl, ArrayList<Long> wayIdAl, ArrayList<Long>
nodeAAl, ArrayList<Long> nodeBAl) {

        // variables
        int arrayListSize = indexAl.size();
        int index = 0;

        // open db to read or write
        SQLiteDatabase database = this.getWritableDatabase();

        // Stores key value pairs being the column name and the data
        // ContentValues data type is needed because the database
        // requires its data type to be passed
        ContentValues values = new ContentValues();

        while (index < arrayListSize){

            // put the values into values
            values.put("entry_id", wayInsertIndex);
            values.put("way_id", wayIdAl.get(index));
            values.put("node_a", nodeAAl.get(index));
            values.put("node_b", nodeBAl.get(index));
```

```
                // insertdata into db ways
                database.insert("ways", null, values);
                values.clear();
                index ++;
                wayInsertIndex ++;
            }

            // close the db
            database.close();
        }

        // insert nodes that belong to a highway
        public void insertValidHighwayNodes(ArrayList<Long> validHighwayIds, ArrayList<Long>
validHighwayNodes) {

            // variables
            int arrayListSize = validHighwayIds.size();
            int index = 0;

            // open db to read or write
            SQLiteDatabase database = this.getWritableDatabase();

            // Stores key value pairs being the column name and the data
            // ContentValues data type is needed because the database
            // requires its data type to be passed
            ContentValues values = new ContentValues();

            while (index < arrayListSize){

                // put the values into values
                values.put("entry_id", validHighwayInsertIndex);
                values.put("valid_highway_id", validHighwayIds.get(index));
                values.put("valid_node", validHighwayNodes.get(index));
```

```
            // insertdata into db valid_highway_nodes
            database.insert("valid_highway_nodes", null, values);
            values.clear();
            index ++;
            validHighwayInsertIndex ++;
        }

        // close the db
        database.close();
    }


    public void insertSingleWayId(long wayId) {

        // open db to read or write
        SQLiteDatabase database = this.getWritableDatabase();

        // Stores key value pairs being the column name and the data
        // ContentValues data type is needed because the database
        // requires its data type to be passed
        ContentValues values = new ContentValues();

        // put the values into values
        values.put("entry_id", singleWayIdIndex);
        values.put("way_id", wayId);

        // insert data into db valid_highway_nodes
        database.insert("single_ways", null, values);
        values.clear();

        singleWayIdIndex ++;

        // close the db
        database.close();
    }
```

```
//
public long getValidNodeId(long nodeId) {
	long wayId = 0;

	// open db to read or write
	SQLiteDatabase database = this.getReadableDatabase();

	// SQL query
	String selectQuery = "SELECT * FROM valid_highway_nodes where valid_node='"+ nodeId + "'";

	// Cursor provides read and write access for the
	// data returned from a database query
	// rawQuery executes the query and returns the result as a Cursor
	Cursor cursor = database.rawQuery(selectQuery, null);
	if (cursor.moveToFirst()) {
		do {
			wayId = cursor.getLong(2);
		} while (cursor.moveToNext());
	}

	// close the db
	database.close();
	return wayId;
}

//
public ArrayList<HashMap<String, String>> getAllWays() {

	// ArrayList that contains every row in the database
	// and each row key / value stored in a HashMap

	ArrayList<HashMap<String, String>> wayArrayList;

	wayArrayList = new ArrayList<HashMap<String, String>>();
```

```
        // SQL query
        String selectQuery = "SELECT * FROM ways";

        // open db to read or write
        SQLiteDatabase database = this.getWritableDatabase();

        // rawQuery executes the query and returns the result as a Cursor
        Cursor cursor = database.rawQuery(selectQuery, null);

        // Move to the first row
        if (cursor.moveToFirst()) {
            do {
                HashMap<String, String> wayMap = new HashMap<String, String>();

                // Store the key / value pairs in a HashMap
                // Access the Cursor data by index that is in the same order
                // as used when creating the table
                wayMap.put("entry_id", cursor.getString(0));
                wayMap.put("way_id", cursor.getString(1));
                wayMap.put("node_a", cursor.getString(2));
                wayMap.put("node_b", cursor.getString(3));

                wayArrayList.add(wayMap);
            } while (cursor.moveToNext()); // Move Cursor to the next row

        }
        // close the db
        database.close();
        return wayArrayList;
    }
}
```

5. DownloaderThread.java

All credit goes to Mujtaba Hassanpur for the code for the DownLoaderThread.java class.

(Hassanpur, 2011)

I only slightly modified some parts to suit my needs. I figured out how to send back toast messages to the ExampleLauncher.java class.

```
/**

 * Copyright (c) 2011 Mujtaba Hassanpur.
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use,
 * copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following
 * conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 */

package com.locationracing.app;
```

```
import org.andengine.examples.R;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

import org.andengine.examples.launcher.*;

import android.os.Environment;
import android.os.Looper;
import android.os.Message;
import android.util.Log;
import android.view.View.OnClickListener;
import android.widget.Toast;

/**
 * Downloads a file in a thread. Will send messages to the
 * AndroidFileDownloader activity to update the progress bar.
 */
public class DownloaderThread extends Thread
{
	// constants
	private static final int DOWNLOAD_BUFFER_SIZE = 4096;


	// ===========================================================
	// My code - START
	// ===========================================================
	// instance variables
```

```
	private ExampleLauncher parentActivity;
	private String downloadUrl;
	private String fileName = "didntWork.xml";

	private final String TAGDL = "[DL File]";

//	ToastHandler mToastHandler = new ToastHandler(parentActivity);
	// ============================================================
	// My code - END
	// ============================================================
	/**
	 * Instantiates a new DownloaderThread object.
	 * @param parentActivity Reference to AndroidFileDownloader activity.
	 * @param inUrl String representing the URL of the file to be downloaded.
	 */
	public DownloaderThread(ExampleLauncher inParentActivity, String inUrl, String fileName)
	{
		this.fileName = fileName;
		downloadUrl = "";
		if(inUrl != null)
		{
			downloadUrl = inUrl;
		}
		parentActivity = inParentActivity;
	}

	/**
	 * Connects to the URL of the file, begins the download, and notifies the
	 * AndroidFileDownloader activity of changes in state. Writes the file to
	 * the root of the SD card.
	 */
	@Override
	public void run()
	{
		URL url;
```

```
            URLConnection conn;
            int fileSize, lastSlash;
            BufferedInputStream inStream;
            BufferedOutputStream outStream;
            File outFile;
            FileOutputStream fileStream;
            Message msg;

            try
            {
                  parentActivity.runOnUiThread(new Runnable() {

                     @Override
                     public void run() {
                         Toast.makeText(parentActivity, fileName + " download started...",
Toast.LENGTH_SHORT).show();

                     }
                });
                  Log.d(TAGDL, "File DL Started");
                  url = new URL(downloadUrl);
                  conn = url.openConnection();
                  conn.setUseCaches(false);
                  fileSize = conn.getContentLength();

                  /**
                  // get the filename
                  lastSlash = url.toString().lastIndexOf('/');
                  fileName = "file.bin";
                  if(lastSlash >=0)
                  {
                        fileName = url.toString().substring(lastSlash + 1);
                  }
                  if(fileName.equals(""))
                  {
```

```
                fileName = "file.bin";
            }*/

            // notify download start
            int fileSizeInKB = fileSize / 1024;

            // start download
            inStream = new BufferedInputStream(conn.getInputStream());

            // ===========================================================
            // My code
            // ===========================================================

            outFile = new File(Environment.getExternalStorageDirectory() + "/" + fileName);

            Log.d(TAGDL, "String Path -> " + Environment.getExternalStorageDirectory() + "/" +
fileName);

            // ===========================================================
            // My code
            // ===========================================================

            fileStream = new FileOutputStream(outFile);
            outStream = new BufferedOutputStream(fileStream, DOWNLOAD_BUFFER_SIZE);
            byte[] data = new byte[DOWNLOAD_BUFFER_SIZE];
            int bytesRead = 0, totalRead = 0;
            while(!isInterrupted() && (bytesRead = inStream.read(data, 0, data.length)) >= 0)
            {
                outStream.write(data, 0, bytesRead);

                // update progress bar
                totalRead += bytesRead;
                int totalReadInKB = totalRead / 1024;
            }
```

```
                outStream.close();
                fileStream.close();
                inStream.close();

                if(isInterrupted())
                {
                      // the download was canceled, so let's delete the partially downloaded file
                      outFile.delete();
                      Log.d(TAGDL, "File DL Failed");
                      parentActivity.runOnUiThread(new Runnable() {

                         @Override
                         public void run() {
                             Toast.makeText(parentActivity, fileName + " download failed :(\nPlease try
again", Toast.LENGTH_SHORT).show();

                         }
                     });
                }
                else
                {
                      Log.d(TAGDL, "File DL Successful");
                      parentActivity.runOnUiThread(new Runnable() {

                         @Override
                         public void run() {
                             Toast.makeText(parentActivity, fileName + " download successful :)\nNow,
create map", Toast.LENGTH_SHORT).show();

                         }
                     });
                }

            }
            catch(MalformedURLException e)
```

```
            {
                  Log.d(TAGDL, "File DL Failed");
                  parentActivity.runOnUiThread(new Runnable() {

                @Override
                public void run() {
                    Toast.makeText(parentActivity, fileName + " download failed :(\nPlease try again",
Toast.LENGTH_SHORT).show();

                }
            });
             }
             catch(FileNotFoundException e)
             {
                  Log.d(TAGDL, "File DL Failed");
                  parentActivity.runOnUiThread(new Runnable() {

                     @Override
                     public void run() {
                         Toast.makeText(parentActivity, "The server is busy :(\nPlease try again",
Toast.LENGTH_SHORT).show();

                     }
                 });
             }
             catch(Exception e)
             {
                  Log.d(TAGDL, "File DL Failed");
                  parentActivity.runOnUiThread(new Runnable() {

                     @Override
                     public void run() {
                         Toast.makeText(parentActivity, fileName + " download failed :(\nPlease try again",
Toast.LENGTH_SHORT).show();
```

```
                        }
                    });
                }
            }

}
```

6. NorthSouthEastWestBoundaries.java

All of the code in this file was created by Philip Stafford.

```
// ===========================================================
// My code - START
// ===========================================================

package com.locationracing.app;

import android.util.Log;

public class NortSouthEastWestBoundaries {

	private static double minLatSouth;
	private static double minLonWest;
	private static double maxLatNorth;
	private static double maxLonEast;

	private static final String TAGNSEW = "[NSEW]";

	public NortSouthEastWestBoundaries() {

		minLatSouth = 0;
		minLonWest = 0;
		maxLatNorth = 0;
		maxLonEast = 0;
	}

	public static double getMinLatSouth() {
		Log.d(TAGNSEW, "getMinLatSouth= " + minLatSouth);
		return minLatSouth;
	}
	public static void setMinLatSouth(double minLatSouth) {
		NortSouthEastWestBoundaries.minLatSouth = minLatSouth;
```

```
            Log.d(TAGNSEW, "setMinLatSouth= " + minLatSouth);
      }
      public static double getMinLonWest() {
            Log.d(TAGNSEW, "setMinLonWest= " + minLonWest);
            return minLonWest;
      }
      public static void setMinLonWest(double minLonWest) {
            NortSouthEastWestBoundaries.minLonWest = minLonWest;
            Log.d(TAGNSEW, "setMinLonWest= " + minLonWest);
      }
      public static double getMaxLatNorth() {
            Log.d(TAGNSEW, "maxLatNorth= " + maxLatNorth);
            return maxLatNorth;
      }
      public static void setMaxLatNorth(double maxLatNorth) {
            NortSouthEastWestBoundaries.maxLatNorth = maxLatNorth;
            Log.d(TAGNSEW, "setMaxLatNorth= " + maxLatNorth);
      }
      public static double getMaxLonEast() {
            Log.d(TAGNSEW, "maxLonEast= " + maxLonEast);
            return maxLonEast;
      }
      public static void setMaxLonEast(double maxLonEast) {
            NortSouthEastWestBoundaries.maxLonEast = maxLonEast;
            Log.d(TAGNSEW, "setMaxLonEast= " + maxLonEast);
      }
}
//=============================================================
//My code - END
//=============================================================
```

References

Gramlich, N. (2010, May 5). *nicolasgramlich (Nicolas Gramlich) · GitHub*. Retrieved from Github.com: https://github.com/nicolasgramlich/

Hassanpur, M. (2011, April 17). *Android Development: Downloading a file from the web*. Retrieved from Hassanpur.com: http://www.hassanpur.com/blog/2011/04/android-development-downloading-a-file-from-the-web/