CoordinatesContainer.java
available at: http://sourceforge.net/p/walkingtoolsgpx/code/HEAD/tree/tbtool/src/edu/ucsd/calit2/TransBorderTool/CoordinatesContainer.java

```
/*  WalkingtoolsGpx: XML, APIs, and Apps for Walking Artists
    Copyright (C) 2007-2012 Walkingtools project/b.a.n.g. lab UCSD

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

package edu.ucsd.calit2.TransBorderTool;

/*
 * XML Parsing using kxml2 example by Naveen Balani
 */
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;
import javax.microedition.location.Coordinates;

/** This class extends a Hashtable to add methods that can sort the data
 *  geographically or topologically to produce some notion of which of the
 *  "nearby" points might be "closest". Various methods are expected to be
 *  added to the class, which sort these differently.
 * @author Jason Najarro and Brett Stalbaum
 * @version 0.5.5
 */
public class CoordinatesContainer extends Hashtable {
/**
 * Default constructor
 */
public CoordinatesContainer() {
    super();
}

/** Gathers the TBCoordinates
 * @param currentPos the current position
 * @param range with in range meters
 * @return a Vector of TBCoordinates
 */
public Vector getNearestCoords(Coordinates currentPos, int range) {
    Vector nearestWaypoints = new Vector();

    // Get coordinates within range as Vector
    for (Enumeration e = this.elements(); e.hasMoreElements();) {
        TBCoordinates hCoords = (TBCoordinates) e.nextElement();
        float distance = currentPos.distance(hCoords);
        if (distance <= range) {
            nearestWaypoints.addElement(hCoords);
        }
    }

    // Sort the Vector by distance from current position
    int n = nearestWaypoints.size();
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            float distance1 = currentPos.distance((TBCoordinates) nearestWaypoints.elementAt(j));
            float distance2 = currentPos.distance((TBCoordinates) nearestWaypoints.elementAt(j + 1));
            if (distance1 > distance2) {
                TBCoordinates temp = (TBCoordinates) nearestWaypoints.elementAt(j);
                nearestWaypoints.setElementAt(nearestWaypoints.elementAt(j + 1), j);
                nearestWaypoints.setElementAt(temp, j + 1);
            }
        }
    }

    //System.out.println("vector " + nearestWaypoints.size());
    return nearestWaypoints;
    }
}
```

13 **DowsingCompass.java**

available at: http://sourceforge.net/p/walkingtoolsgpx/code/HEAD/tree/tbtool/src/edu/ucsd/calit2/TransBorderTool/DrowsingCompass.java

```
/*  WalkingtoolsGpx: XML, APIs, and Apps for Walking Artists
    Copyright (C) 2007-2012 Walkingtools project/b.a.n.g. lab UCSD

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

package edu.ucsd.calit2.TransBorderTool;

import edu.ucsd.calit2.TransBorderTool.international.*;
import java.util.Vector;
import javax.microedition.location.LocationProvider;
import javax.microedition.location.Location;
import net.walkingtools.javame.canvas.CompassCanvas;

/**
 * @author Brett Stalbaum and Jason Najarro
 * @version 0.5.5
 */
public class DowsingCompass extends CompassCanvas {

    //private DowsingCompassListener MIDlet = null;
    private CoordinatesContainer waypointList = null;
private Vector nearbyWPVector = null;
private boolean gotFirstFix = false;
private final byte TRIGGER_RADIUS = 30;
private DowsingCompassListener dowsingListener = null;

/**
 * Constructor for a DowsingCompass
 *
 * @param gpxFileName the GPX file name
 */
public DowsingCompass(String gpxFileName) {
    super(Translation.loadTranslation("en"));
    super.setTriggerRadius(TRIGGER_RADIUS);
    // simply making a GPXparser causes it to come to life an parse the file given.
    TBGpxParser gpxParser = new TBGpxParser(TBGpxParser.RES, "../../../../" + gpxFileName);
    // get the CoordinatesContainer (a Hashtable) from the parser - it should
    // contain all the the waypoints in the GPX file
    waypointList = gpxParser.getCoordsContainer();
    nearbyWPVector = new Vector();
}

public void setTarget(TBCoordinates targetCoords) {
    target = targetCoords;
    super.setTarget(target);
    navigating = true;
}

public void stopNavigation() {
    super.stopNavigation();
    navigating = false;
}

public Vector getNearbyWaypoints(int distance) {
    nearbyWPVector = waypointList.getNearestCoords(currentCoords, distance);
    return nearbyWPVector;
}

public void addNavigatorListener(DowsingCompassListener navigatorListener) {
    super.addNavigatorListener(navigatorListener);
```

```
            dowsingListener = navigatorListener;
        }

        public void removeNavigatorListener(DowsingCompassListener navigatorListener) {
            super.removeNavigatorListener(navigatorListener);
            dowsingListener = null;
        }

        // override CompassCanvas locationUpdated
        public void locationUpdated(LocationProvider provider, Location location) {
            // call to superclass
            super.locationUpdated(provider, location);
            if (gotFirstFix == false && location.isValid()) {
                gotFirstFix = true;
            } else if (!location.isValid()) {
                return;
            }

            if (!navigating && isMoving()) { // here we are dowsing, but only if moving
                //System.out.println("search waypoint list begun");
                for (int i = 0; i < nearbyWPVector.size(); i++) {
                    TBCoordinates mc = (TBCoordinates) nearbyWPVector.elementAt(i);
                    if (currentCoords.distance(mc) <= TRIGGER_RADIUS + 20) { // + a little
                        continue; // omits sites that we are already at
                    }
                    float directionPointerAzimuth = (int)currentCoords.azimuthTo(mc);
                    //System.out.println(directionPointerAzimuth);
                    if ((currentCompassHeading >= (directionPointerAzimuth - 5) &&
                            currentCompassHeading <= (directionPointerAzimuth + 5))) {
                        if (dowsingListener != null) {
                            dowsingListener.witchingEvent(mc);
                        }
                        break;
                    }
                }
            }
        }
    }
```

DowsingCompassListener.java
available at: http://sourceforge.net/p/walkingtoolsgpx/code/HEAD/tree/tbtool/src/edu/ucsd/calit2/TransBorderTool/DrowsingCompassListener.java

```
/*  WalkingtoolsGpx: XML, APIs, and Apps for Walking Artists
    Copyright (C) 2007-2012 Walkingtools project/b.a.n.g. lab UCSD

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

package edu.ucsd.calit2.TransBorderTool;

import net.walkingtools.javame.canvas.NavigatorListener;

/**
 * This interface extends edu.ucsd.calit2.TransBorderTool.GPS.CompassListener
 * adding the witchingEvent method for the TransBorder tool
 * @author Brett Stalbaum
 * @version 0.5.5
 */
public interface DowsingCompassListener extends NavigatorListener {

      /**
       * Allows the implementing class to receive witching events
       * @param mc the MetaCoordinates object describing the witched site
       */
      public void witchingEvent(TBCoordinates mc);

}
```

TBCoordinates.java

available at: http://sourceforge.net/p/walkingtoolsgpx/code/HEAD/tree/tbtool/src/edu/ucsd/calit2/TransBorderTool/TBCoordinates.java

```
/*  WalkingtoolsGpx: XML, APIs, and Apps for Walking Artists
    Copyright (C) 2007-2012 Walkingtools project/b.a.n.g. lab UCSD

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

package edu.ucsd.calit2.TransBorderTool;

import java.io.IOException;
import javax.microedition.location.Coordinates;
import javax.microedition.lcdui.Image;

/**
 * The TBCoordinates class extends javax.microedition.location.Coordinates
 * to include various MetaData
 * @author Brett Stalbaum and Jason Najarro
 * @version 0.5.5
 */
public class TBCoordinates extends Coordinates {

    public final int WATER = 0;
    public final int BEACON = 1;
    public final int CITY = 2;
    private String name = "";
private String type = "";

public TBCoordinates(double latitude, double longitude, float altitude, String name, String type) {
    super(latitude, longitude, altitude);
    this.name = name;
    this.type = type;
}

public TBCoordinates() {
    super(0, 0, 0);
    name = "";
    type = "";
}

// Convert Coordinates to TBCoordinates
//
/**
 * @param c
 * @return
 */
public static TBCoordinates toMetaCoords(Coordinates c, String name, String type) {
    TBCoordinates mc = new TBCoordinates(c.getLatitude(), c.getLongitude(),
      c.getAltitude(), name, type);
    return mc;
}

// Convert Coordinates to TBCoordinates
//
/**
 * @param c
 * @return
 */
public static TBCoordinates toMetaCoords(Coordinates c) {
    TBCoordinates mc = toMetaCoords(c, "", "");
    return mc;
}

public void setName(String n) {
    name = n;
}
```

```
    /** Set TBCoordinates type
     * e.g water station, safety beacon, etc.
     */
    public void setType(String t) {
        type = t;
    }

    public String getName() {
        return name;
    }

    public String getType() {
        return type;
    }

    /** Returns image pertaining to MetaCoordinate type
     */
    public Image getIcon() {
        Image icon = null;
        try {
            //System.out.println(this.getType());
            if (this.getType().toLowerCase().endsWith("water")) {
            // matching "Water", "Drinking Water", "N Water...."
                icon = Image.createImage("/img/water_icon_sm.png");
            } else if (this.getType().equalsIgnoreCase("beacon")) {
                icon = Image.createImage("/img/beacon_icon_sm.png");
            } else if (this.getType().equalsIgnoreCase("city")) {
                icon = Image.createImage("/img/generic_icon_sm.png");
            } else {
                icon = Image.createImage("/img/generic_icon_sm.png");
            }
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
        return icon;
    }
}
```

TBGpxParser.java

available at: http://sourceforge.net/p/walkingtoolsgpx/code/HEAD/tree/tbtool/src/edu/ucsd/calit2/TransBorderTool/TBGpxParser.java

```
/*  WalkingtoolsGpx: XML, APIs, and Apps for Walking Artists
    Copyright (C) 2007-2012 Walkingtools project/b.a.n.g. lab UCSD

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

package edu.ucsd.calit2.TransBorderTool;

/* KXML APIs - http://kobjects.org/kxml/index.php
 * USE of the Kxml packages require the distribution of the following license
 * Copyright (c) 2002-2007 Stefan Haustein, Oberhausen, Rhld., Germany
 *
 * 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.
 */

import org.kxml2.io.KXmlParser;
import javax.microedition.io.HttpConnection;
import javax.microedition.io.Connector;
import java.io.InputStreamReader;
import java.io.InputStream;
import java.io.IOException;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
/**
 * Objects of this class serve the narrow
 * purpose of reading a .gpx file containing only waypoints. It was updated a great deal
 * as of version B3 to make it more robust-it should now handle any gpx file containing
 * waypoints, regardless of other junk in the file.
 * @author Jason Najarro and Brett Stalbaum
 * @version 0.5.5
 */
public class TBGpxParser implements Runnable {

      /**
       * Load from resource
       */
      public static final int RES = 0;
      /**
       * Load from HTTP connection
       */
      public static final int HTTP = 1;
      private int type;
      private String filePath;
      private CoordinatesContainer coordsList = null;
      // parse states
      boolean open = false;
      boolean waypointOpen = false;
      boolean gotName = false;
      boolean gotSym = false;
```

```
String sym = "";
String name = "";
double lat = -1;
double lon = -1;
private boolean loading = true;

/** This constructor accepts a source type (see the static final values of this class) and a
    String
 * representing the file path or URL to the source location.
 * @param sourceType as of version beta 3, HTTP or RES
 * @param sourceLocation a file path or URL
 */
public TBGpxParser(int sourceType, String sourceLocation) {
      coordsList = new CoordinatesContainer();
      type = sourceType;
      filePath = sourceLocation;
      Thread t = new Thread(this);
      t.start();
}


/** This constructor accepts a source type (see the static final values of this class), a
    String
 * representing the file path or URL to the source location, and a CoordinatesContainer if
 * you have one.
 * @param sourceType as of version beta 3, HTTP or RES
 * @param sourceLocation a file path or URL
 * @param container a CoordinatesContainer, left out, this class will produce its own.
 */
public TBGpxParser(int sourceType, String sourceLocation, CoordinatesContainer container) {
      type = sourceType;
      filePath = sourceLocation;
      coordsList=container;
      Thread t = new Thread(this);
      t.start();
}

/** returns true if objects of this class are busy loading the file
 * @return true if still loading
 */
```

```
public boolean loading() {
      return loading;
}

public void run(){
      try {
            KXmlParser parser = new KXmlParser();

            if(type==HTTP){
                  //Open http connection
                  HttpConnection httpConnection = (HttpConnection) Connector.open(filePath);

                  //Initilialize XML parser
                  parser.setInput(new InputStreamReader(httpConnection.openInputStream()));
            }else if(type==RES){
                  InputStream GPXInStream = null;
                  //System.out.println(filePath);
                  GPXInStream = this.getClass().getResourceAsStream(filePath);
                  //Initilialize XML parser
                  parser.setInput(new InputStreamReader(GPXInStream));
            }
            //System.out.println("calling parse");
            parseGPXData(parser);

      }catch (Exception e) {
            //e.printStackTrace ();
            //System.out.println("Error:");
            //System.out.println(e.toString ());
            return;
      }
      loading = false;
}

/* Completely rebuilt this part on May 10, 2008, in order to make the class
* more robust. Before it assumed that certain elements would be contained in
* a particular order, without any intervening xml tags that might be added
* by various applications. (Such as G7towin, which produced gpx files that
* this class threw up on.) I tried to make it more robust by looking for tags
* and accepting their data if they contained "wpt" data, and just ignoring
* anything else it finds. Brett.
```

```
*/
private void parseGPXData(KXmlParser parser) throws IOException {
        try{
                parser.nextTag();
                if (parser.getEventType() != XmlPullParser.END_DOCUMENT) open = true;
                while (open) {
                        /*
                        // elegant testing!
                        int tabs = parser.getDepth();
                        StringBuffer buf = new StringBuffer();
                        for (int i = 0; i < tabs; i++ ) {
                                buf.append("\t*");
                        }
                        */
                        // parsing the Start tags has all the stuff we care about
                        if (parser.getEventType() == XmlPullParser.START_TAG) {
                                String currentName = parser.getName();
                                //System.out.println(buf.toString() + currentName);
                                if (currentName.equals("wpt")) {
                                        lat = Double.parseDouble(parser.getAttributeValue(0));
                                        lon = Double.parseDouble(parser.getAttributeValue(1));
                                        waypointOpen = true;
                                }
                                if (waypointOpen && currentName.equals("sym")) {
                                        parser.next();
                                        sym = new String(parser.getText());
                                        gotSym = true;
                                }
                                if (waypointOpen && currentName.equals("name")) {
                                        parser.next();
                                        name = new String(parser.getText());
                                        gotName = true;
                                }
                                if (waypointOpen && gotSym && gotName) {
                                        //System.out.println("*** " + name + " " + sym);
                                        String localName = new String(name.toCharArray());
                                        String localSym = new String(sym.toCharArray());
                                        //System.out.println("*** " + localName.hashCode() + " " + lo
                                          //calSym.hashCode());
                                        TBCoordinates wp = new TBCoordinates(lat, lon, 0, localName,
                                localSym);
                            waypointOpen = false;
                            gotSym = false;
                            gotName = false;// Add MetaCoordinate to CoordinatesContainer
                            //System.out.println(wp.getLatitude() + " "+ wp.getLongitude() + " "
                                //+ wp.getName() + " " + wp.getType());
                            coordsList.put(localName, wp);
                            //System.out.println(coordsList.hashCode() + " "
                                //+ coordsList.size());
                        }
                        //System.out.println(buf.toString() + waypointOpen + " " + gotName + " " +
                            //gotSym);
                        parseGPXData(parser);
                    }

                    parser.nextTag();
                }
                // we are at an end tag
                //System.out.println("got out");

        }catch(XmlPullParserException xe){
                //xe.printStackTrace();
                return; // not so great, but there seems to be no other choice
        }

    }

        /** Returns the CoordinantsContainter
         * @return the CoordinatesContainer representing the coordinates
         */
        public CoordinatesContainer getCoordsContainer(){
                //System.out.println("getCoordsContainer() " + coordsList.hashCode());
                return coordsList;
        }

}
```

TBMIDlet.java
available at: http://sourceforge.net/p/walkingtoolsgpx/code/HEAD/tree/tbtool/src/edu/ucsd/calit2/TransBorderTool/TBMIDlet.java

```
/*  WalkingtoolsGpx: XML, APIs, and Apps for Walking Artists
    Copyright (C) 2007-2012 Walkingtools project/b.a.n.g. lab UCSD

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.

*/

// note: good idea to extend the dowsing interface to include (or redirect users)
// to sites that are within 500 meters.

package edu.ucsd.calit2.TransBorderTool;

import edu.ucsd.calit2.TransBorderTool.international.*;
import javax.microedition.MIDlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.location.*;
import java.io.*;
import java.util.*;
import javax.microedition.media.*;
import net.walkingtools.javame.util.AudioArrayPlayer;

/**
 * @author Brett Stalbaum and Jason Najarro
 * @version 0.5.5
 */
```

```
public class TBMIDlet extends MIDlet implements DowsingCompassListener, CommandListener {

    private Display display = null;
    // current displayable will normally be the tbDowsingCompass, but if expired, an alert.
    private DowsingCompass tbDowsingCompass = null;
    private Vector nearbyWPList = null;
    private static final int SEARCH_DISTANCE = 10000; // 10K
    private List targetList = null;
    private Alert arrivedAlert = null;
    private Alert waypointAheadAlert = null;
    private Alert expired = null;
    private boolean isExpired = false;
    private Alert expirationWarning = null;
    private boolean expireWarning = false;
    private Alert startUpDisplay = null;
    private boolean startUpAlert = false;
    private Alert noNearbyWaypoints = null;
    private Alert minimalistInfoAlert = null;
    private Command exit = null;
    private Command ignore = null;
    private Command cancel = null;
    private Command listNearbyWPs = null;
    private Command setTargetManual = null;
    private Command setTargetAuto = null;
    private AudioTimer audioTimer = null;
    // private String URL = "http://internetjunkee.com/transborder/GPScourseFinal.gpx";
    private LocationProvider lp = null;
    private TBCoordinates aheadCoords = null;
    // using only one audio player for two kinds of sound. The first is the
    // poems, the second is the Audio UI elements
    private static final String[] audioStrings = {
      "1-GAIN_04-02.wav", "2-GAIN_02-03.wav", "2-GAIN_03-01.wav",
      "3-GAIN_01.wav", "3-GAIN_02-02.wav", "5-GAIN_02-07.wav",
      "6-GAIN_04-02.wav", "7-GAIN_02-03.wav", "7-GAIN_03-01.wav",
      "8-GAIN_01.wav", "8-GAIN_02-02.wav", "10-GAIN_02-07.wav",
      "11-GAIN_04-02.wav", "12-GAIN_02-03.wav", "12-GAIN_03-01.wav",
      "13-GAIN_01.wav", "13-GAIN_02-02.wav", "15-GAIN_02-07.wav",
      "16-GAIN_04-02.wav", "17-GAIN_02-03.wav", "17-GAIN_03-01.wav",
      "18-GAIN_01.wav", "18-GAIN_02-02.wav", "20-GAIN_02-07.wav", // 24 poems
      "arriving.wav", "expiration.wav", "expired.wav", "found.wav", // AudioUI
      "lowgps.wav", "move.wav", "nosites.wav", "pointing.wav", // AudioUI
      "read.wav", "searching.wav", "startup.wav", "beep.wav" // AudioUI
    };
    private static final int NUMBER_OF_POEMS = 24;
    private boolean running = false;
    private boolean navigating = false;
    // if the MIDlet is getting an update
    // interval which is adequate for
    // dymanic navigation, dynamicNavigation should be true
    private boolean dynamicNavigation = false;
    private Ticker minimalistTicker = null;
    private net.walkingtools.international.Translation translation = null;
    private AudioArrayPlayer audioPlayer = null;
    private byte moveWarningEnervator = 1;

    /**
     * Constructor for a TransBorderMIDlet
     */
    public TBMIDlet() {
        // load the translation
        translation = Translation.loadTranslation(getAppProperty("language"));

        // get the display
        if (display == null) {
            display = Display.getDisplay(this);
        }

        // test value for jad file... can delete
        //System.out.println(System.currentTimeMillis()+1000*60*60*24*8);

        // set up the test alert first (for debugging on phone)
        exit = new Command(translation.translate("Exit"), Command.EXIT, 0);
        // get the gpx file
        String gpxFile = this.getAppProperty("GPXFile");

        tbDowsingCompass = new DowsingCompass(gpxFile);

        int width = tbDowsingCompass.getWidth();
        Image errorImage = null;
        Image tbImage = null;
```

```
if (width < 150) {
    errorImage = loadImage("error_sm.png");
    tbImage = loadImage("tb_sm.png");
} else {
    errorImage = loadImage("error.png");
    tbImage = loadImage("tb.png");
}

// first, validate the expiration value
ignore = new Command(translation.translate("Ignore"), Command.CANCEL, 0);
String expirationDate = this.getAppProperty("Expiration-Date");
long exp = Long.parseLong(expirationDate);

if (exp <= System.currentTimeMillis()) {
    expired = new Alert(translation.translate("Data expired"),
            translation.translate("The data is expired,
              TBTool is not safe to use."),
            errorImage,
            AlertType.ERROR);
    expired.setTimeout(Alert.FOREVER);
    expired.addCommand(exit);
    expired.addCommand(ignore);
    expired.setCommandListener(this);
    isExpired = true;
} else if (exp <= System.currentTimeMillis() +
   (1000 * 60 * 60 * 24 * 7) ) { // 7 day warning
    Date date = new Date(exp);
    expirationWarning = new Alert(translation.translate("Expiration Warning"),
            translation.translate("\nThe data will expire on:\n") + date.toString() +
            translation.translate("\nTransborder Immigrant Tool\
              nEDT/BANGLAB/CRCA/CALIT2/VISARTS/UCSD\n\n"),
            errorImage,
            AlertType.WARNING);
    expirationWarning.addCommand(ignore);
    expirationWarning.setTimeout(15000);
    expirationWarning.setCommandListener(this);
    expireWarning = true;
} else {
    Date date = new Date(exp);
    startUpDisplay = new Alert(translation.translate
        ("Transborder Immigrant Tool"),
              translation.translate("\nExpires: ") + date.toString() +
              translation.translate("\nTransborder Immigrant Tool\
                nEDT/BANGLAB/CRCA/CALIT2/VISARTS/UCSD\n\n"),
              tbImage,
              AlertType.INFO);
      startUpDisplay.addCommand(ignore);
      startUpDisplay.setTimeout(10000);
      startUpDisplay.setCommandListener(this);
      startUpAlert = true;
   }

   noNearbyWaypoints = new Alert(translation.translate("No Nearby Points"),
           translation.translate("There are no sites within ") +
           (int)((SEARCH_DISTANCE / 1000.0) + .5)
           + translation.translate(" Kilometers."),
           tbImage,
           AlertType.WARNING);

   noNearbyWaypoints.addCommand(ignore);
   noNearbyWaypoints.setTimeout(10000);
   noNearbyWaypoints.setCommandListener(this);

   minimalistInfoAlert = new Alert(translation.translate("Minimal Mode"),
           "",
           tbImage,
           AlertType.WARNING);

   minimalistInfoAlert.addCommand(ignore);
   minimalistInfoAlert.setTimeout(Alert.FOREVER);
   minimalistInfoAlert.setCommandListener(this);

   nearbyWPList = new Vector();
   cancel = new Command(translation.translate("Cancel"), Command.CANCEL, 0);
   listNearbyWPs = new Command(translation.translate("Find"), Command.SCREEN, 1);
   setTargetManual = new Command(translation.translate("Select"), Command.SCREEN, 1);
   setTargetAuto = new Command(translation.translate("Set Target"), Command.SCREEN, 1);

   /* through a lot of tedious testing, I discovered that these
   constructors of the TextFields were throwing an IllegalArgumentException
```

```
when using TextField.DECIMAL or TextField.NUMERIC constraints. The
following is from the javadoc. It seems not to contradict the
use of TextField.DECIMAL or NUMERIC given that I was setting
the forms to a decimal/numeric value... hmmmmm... this must be an
issue in iden implementation.

"Some constraints, such as DECIMAL, require the implementation to
perform syntactic validation of the contents of the text object.
The syntax checking is performed on the actual contents of the text
object, which may differ from the displayed contents as described
above. Syntax checking is performed on the initial contents passed
to the constructors, and it is also enforced for all method calls
that affect the contents of the text object. The methods and
constructors throw IllegalArgumentException if they would result
in the contents of the text object not conforming to the required
syntax."
 */
tbDowsingCompass.addCommand(exit);
tbDowsingCompass.addCommand(listNearbyWPs);
tbDowsingCompass.setCommandListener(this);
tbDowsingCompass.addNavigatorListener(this);

targetList = new List(translation.translate("Select a Target"), List.IMPLICIT);
targetList.addCommand(cancel);
targetList.addCommand(setTargetManual);
targetList.setCommandListener(this);

waypointAheadAlert = new Alert(translation.translate("Site Ahead!"),
        translation.translate("Site Ahead!"),
        tbImage, AlertType.INFO);
waypointAheadAlert.setTimeout(Alert.FOREVER);
waypointAheadAlert.addCommand(ignore);
waypointAheadAlert.addCommand(setTargetAuto);
waypointAheadAlert.setCommandListener(this);

arrivedAlert = new Alert(translation.translate("Arrived at Site"),
        translation.translate("Arrived at Site"),
        tbImage, AlertType.INFO);
arrivedAlert.setTimeout(Alert.FOREVER);
arrivedAlert.addCommand(ignore);
arrivedAlert.setCommandListener(this);
minimalistTicker = new Ticker(
            translation.translate("Minimal or no GPS signal.
              Alert will give direction and distance information if possible.")
        );
dynamicNavigation = true;
  // assume active navigation at startup of gps to give it a chance to fix

// set up location provider
// Set criteria for selecting a location provider:
// accurate to 50 meters horizontally
try {
    Criteria cr = new Criteria();
    cr.setHorizontalAccuracy(50);
    // we can set other criteria that we require
    cr.setSpeedAndCourseRequired(true);
    cr.setPreferredResponseTime(2000);
    cr.setAltitudeRequired(true);
    try {
        // Get an instance of the provider
        lp = LocationProvider.getInstance(cr);
    } catch (LocationException e) { // if this happens, lp could not get a location
        display.setCurrent(new Alert(translation.translate
          ("Exception on getting location provider"),
                translation.translate("Exception on getting location provider")
                  + ':' + e.toString(),
                null,
                AlertType.INFO));
    }
    // register this with the location listener
    // the second argument is the interval. -1 is a flag that says,
    // "whatever works best for you"
    // the third arg is the timeout, or, how many seconds past the
    // interval defined in arg 2
    // the provider should wait before it returns an invalid Location
    // the fourth is the maxAge of a valid location.
    // The provider may provide a valid location
    // in lieu of a current location as long as it is not older than this.
    lp.setLocationListener(tbDowsingCompass, 2, 2, 2);
} catch (SecurityException se) {
        Alert noLocationService = new Alert(translation.translate
         ("TBtool requires location"),
                translation.translate("The Transborder Immigrant Tool
                  needs access to location services.") +
                translation.translate("Try answering \"Yes\"
                 on startup to grant TBtool access."),
                errorImage,
                AlertType.INFO);
        noLocationService.setTimeout(Alert.FOREVER);
        noLocationService.addCommand(exit);
        noLocationService.setCommandListener(this);
        display.setCurrent(noLocationService);
    }
}

protected void startApp() throws MIDletStateChangeException {
    // get the display
    if (display == null) {
        display = Display.getDisplay(this);
    }

    // this thread to randomly play audio file
    try {
        audioPlayer = new AudioArrayPlayer("audio", audioStrings, true);
        // true, in audio cueing mode
        //InputStream in = getClass().getResourceAsStream("/audio/beep.wav");
        audioTimer = new AudioTimer();
        running = true;
        audioTimer.start(); // start audio thread
    } catch (IOException e) {
        Alert bailOnAudioException = new Alert(translation.translate
         ("Could not load audio"),
                translation.translate("Could not load audio"),
                loadImage("error_sm.png"),
                AlertType.INFO);
        bailOnAudioException.setTimeout(Alert.FOREVER);
        bailOnAudioException.addCommand(exit);
        display.setCurrent(bailOnAudioException);
    } catch (MediaException e) {
        Alert bailOnAudioException = new Alert(translation.translate
            ("Could not play audio"),
                  translation.translate("Could not play audio"),
                  loadImage("error_sm.png"),
                  AlertType.INFO);
          bailOnAudioException.setTimeout(Alert.FOREVER);
          bailOnAudioException.addCommand(exit);
          display.setCurrent(bailOnAudioException);
      }

      // make sure the data is not expired
      if (isExpired) {
          display.setCurrent(expired);
          display.vibrate(1000);
          playAudioFile("expired.wav", true);
      } else if (expireWarning) {
          display.setCurrent(expirationWarning, tbDowsingCompass);
          display.vibrate(1000);
          playAudioFile("expiration.wav", true);
      } else if (startUpAlert) { //first time only
          startUpAlert = false;
          display.setCurrent(startUpDisplay, tbDowsingCompass);
          display.vibrate(1000);
          playAudioFile("startup.wav", true);
      } else { // we are good to go
          display.setCurrent(tbDowsingCompass);
      }
  }

  /**
   * edu.ucsd.calit2.TransBorderTool.CompassListener interface method
   * Called when user is facing a waypoint
   * Displays waypointAheadAlert pertaining to type of waypoint
   */
  public void witchingEvent(TBCoordinates mc) {
      aheadCoords = mc;
      if (display.getCurrent().equals(tbDowsingCompass)) {
          waypointAheadAlert.setString(tbDowsingCompass.getInfo(mc));
          waypointAheadAlert.setImage(aheadCoords.getIcon());
          double distance = tbDowsingCompass.distanceTo(mc);
          if (distance > SEARCH_DISTANCE) {
            display.vibrate(100);
        } else if (distance > 1000) {
            display.vibrate(300);
        } else if (distance > 500) {
            display.vibrate(500);
        } else if (distance > 100) {
            display.vibrate(800);
        }
        display.setCurrent(waypointAheadAlert);
        display.vibrate(1000);
        playAudioFile("found.wav", false);
    }
}

/**
 * NavigatorListener interface method
 * Displays alert when user arrives within range of target
 */
public void arrivedAtTarget(int distance) {
    navigating = false;
    // stop the compass from navigating
    tbDowsingCompass.stopNavigation();
    display.setCurrent(arrivedAlert);
    display.vibrate(1000);
    playAudioFile("arriving.wav", false);
}

// all of the UI audio files are played through this method
// the poems are not played through this method,
// see second arg in playFileName below
private void playAudioFile(String name, boolean interrupt) {
    try {
        audioPlayer.playFileName(name, interrupt); // true will interrupt a poem if playing
    } catch (MediaException e) {
        try {
            audioPlayer.playFileName("beep.wav", true);
        } catch (MediaException ex) {
            return;
        } catch (Exception eb) {
            eb.printStackTrace();
        }
    }
}

/**
 * NavigatorListener interface method
 * Called to populate nearby waypoint vector
 * once the CompassCanvas detects a valid location
 * so user may begin "dowsing" for waypoints
 * @param ready true for ready to navigate
 */
public void navigationReady(boolean ready) {
    if (ready) {
        nearbyWPList = tbDowsingCompass.getNearbyWaypoints(SEARCH_DISTANCE);
        if (navigating) {
            tbDowsingCompass.removeCommand(listNearbyWPs);
            tbDowsingCompass.addCommand(cancel);
        } else {
            tbDowsingCompass.addCommand(listNearbyWPs);
            tbDowsingCompass.removeCommand(cancel);
        }
    } else {
        if (!navigating) {
            tbDowsingCompass.removeCommand(listNearbyWPs); // can’t use
        }
    }
}

/**
 * NavigatorListener interface method tells the MIDlet the GPS refresh rate
 * of the Navigator (DowsingCompass...) If the MIDlet is getting an update
 * interval which is adequate for dymanic navigation then dynamic
 * (compass based) navigation should be on.
 * Otherwise the phone enters into a minimalist mode that can still provide
 * an occasional alert, useful with less capable phones or in place where
 * GPS coverage is poor. In these cases the user may still be able to navigate
 * with a magnetic compass.
 * @param milliseconds reported milliseconds since last update
 */
public void updateInterval(long milliseconds) {
```

```
// if the device is without update for 10 minutes, enter minimal mode
if (milliseconds > 1000*60*10) { // signal is not good
    if (dynamicNavigation) { // entering non dynamic mode from dynamic
        tbDowsingCompass.setTicker(minimalistTicker);
        if (!navigating) {
            tbDowsingCompass.removeCommand(listNearbyWPs); // can't use
        }
        display.vibrate(1000);
        playAudioFile("lowgps.wav", true);
    }
    dynamicNavigation = false;
 } else { // we have a good signal
    if (!dynamicNavigation) { // we are now returning from a bad signal
        // because dN is set to true in the constructor
        // we must be returning from non-dynamic to dynamic, not just starting
        // restore interface to last state
        tbDowsingCompass.setTicker(null);
        // Offer any available help to user
        // get closest point data into alert string if available
        nearbyWPList = tbDowsingCompass.getNearbyWaypoints(SEARCH_DISTANCE);
        if (nearbyWPList != null && !nearbyWPList.isEmpty()) {
            TBCoordinates target = (TBCoordinates)nearbyWPList.elementAt(0);
            Coordinates current = tbDowsingCompass.getCurrentCoords();
            float distance = current.distance(target);
            String distanceStr = null;
            if (distance >= 1000) {
                distanceStr = (int)(distance/1000) + translation.translate(" Kilometers");
            } else {
                distanceStr = distance + translation.translate(" Meters");
            }
            // create minimalist info alert (if it is just an intermittent single report
            // then at least this info will be left on screen as the system goes
            // back into non dynamic navigation mode.
            minimalistInfoAlert.setString(
                    translation.translate("Nearest Site: Distance ") + distanceStr + ", " +
                    translation.translate("Azimuth ") + (int)current.azimuthTo(target)
                    + translation.translate(" degrees, General Direction ") +
                    tbDowsingCompass.directionTo(target)
                );
            display.setCurrent(minimalistInfoAlert, tbDowsingCompass);
```

```
                display.vibrate(1000);
                playAudioFile("read.wav", false);
            } else {
                display.setCurrent(noNearbyWaypoints);
                display.vibrate(1000);
                playAudioFile("nosites.wav", false);
            }
        }
        dynamicNavigation = true;
    }
}

/** (non-Javadoc)
 * @param arg0
 * @throws MIDletStateChangeException
 * @see javax.microedition.MIDlet.MIDlet#destroyApp(boolean)
 */
protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
    // TODO Auto-generated method stub
}

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

private Image loadImage(String str) {
    Image image = null;
    try {
        image = Image.createImage("/img/" + str);
    } catch (IOException e) {

        image = null;
    }
    //System.out.println(image);

    return image;
}
```

```
public void commandAction(Command c, Displayable d) {
    if (c == exit) { // exit
        running = false;
        notifyDestroyed();
    } else if (c == cancel) { // stop navigation and reset softkey commands
        navigating = false;
        tbDowsingCompass.stopNavigation();
        tbDowsingCompass.removeCommand(cancel);
        if (dynamicNavigation) {
            tbDowsingCompass.addCommand(listNearbyWPs);
        }
        display.setCurrent(tbDowsingCompass);
    } else if (c == ignore) {        // Returns to compass interface if user chooses not
                                     // to set a dowsingEvent as a target
        if (navigating) {
            tbDowsingCompass.removeCommand(listNearbyWPs);
            tbDowsingCompass.addCommand(cancel);
        } else {
            tbDowsingCompass.removeCommand(cancel);
            if (dynamicNavigation) {
                tbDowsingCompass.addCommand(listNearbyWPs);
            }
        }
        display.setCurrent(tbDowsingCompass);
    } else if (c == listNearbyWPs) {    // Display a List of waypoints within range
                                        // from which user can manually choose a target
                                        // Update nearby waypoint vector
        nearbyWPList = tbDowsingCompass.getNearbyWaypoints(SEARCH_DISTANCE);
        if (nearbyWPList != null && !nearbyWPList.isEmpty()) {
            targetList.deleteAll();
            // Loop through waypoint vector adding waypoint
            // image and information to list
            for (int i = 0; i < nearbyWPList.size(); i++) {
                TBCoordinates mc = (TBCoordinates) nearbyWPList.elementAt(i);
                targetList.append(tbDowsingCompass.getInfo(mc), mc.getIcon());
            }
            display.setCurrent(targetList);
        } else {
            display.setCurrent(noNearbyWaypoints);
```

```
                playAudioFile("nosites.wav", true);
            }
        } else if (c == setTargetAuto) {
        // Set a waypoint detected by a dowsingEvent as the target
            navigating = true;
            tbDowsingCompass.setTarget(aheadCoords);
            // Change commands on tbDowsingCanvas
            tbDowsingCompass.removeCommand(listNearbyWPs);
            tbDowsingCompass.addCommand(cancel);
            display.setCurrent(tbDowsingCompass);
            // Set a waypoint selected from nearby waypoint List as the target
        } else if (c == setTargetManual) {
            navigating = true;
            int index = targetList.getSelectedIndex();
            tbDowsingCompass.setTarget((TBCoordinates) nearbyWPList.elementAt(index));
            //Change Commands on tbDowsingCanvas
            tbDowsingCompass.removeCommand(listNearbyWPs);
            tbDowsingCompass.addCommand(cancel);
            display.setCurrent(tbDowsingCompass);
        }
    }

    public void motionStatusUpdate(boolean isMoving) {
        if (isMoving) { // updated to moving
            nearbyWPList = tbDowsingCompass.getNearbyWaypoints(SEARCH_DISTANCE);
            // so update nearby points
        } else { // updated not moving
            display.vibrate(200);
            if (moveWarningEnervator % 5 == 0) { // only play this file ~ every 5th time
                playAudioFile("move.wav", false);
                // the "move for compass" message can be too frequent
            }
            moveWarningEnervator++;
        }
    }

    // inner class to control audio
    class AudioTimer extends Thread {

        Random rand = new Random();
        public void run() {
            while (running) {
                try {
                    Thread.sleep(1000 * 60 * (rand.nextInt(19) + 1)); // sleep random min-
utes
                    //Thread.sleep(1000 * 60); // sleep one min (test)
                } catch (InterruptedException e) {
                    running = false;
                }
                try {
                    int randIndex = rand.nextInt(NUMBER_OF_POEMS);
                    // poems at the top of the audio array
                    audioPlayer.play(randIndex, false);
                    // false means to cue the audio if something else is playing
                } catch (MediaException ex) {
                    Alert bailOnAudioException =
                      w     new Alert(translation.translate("media exception"),
                            ex.getMessage(),
                            null,
                            AlertType.INFO);
                    bailOnAudioException.setTimeout(Alert.FOREVER);
                    display.setCurrent(bailOnAudioException);
                }
            }
        }

        public void finalize() {
            running = false;
        }
    }
}
```