```
    1: package com.droidplanner.activitys;
    2:
    3: import android.app.ActionBar;
    4: import android.content.Intent;
    5: import android.os.Bundle;
    6: import android.support.v13.app.FragmentTabHost;
    7: import android.support.v4.app.NavUtils;
    8: import android.view.MenuItem;
    9:
   10: import com.droidplanner.R;
   11: import com.droidplanner.activitys.helpers.SuperUI;
   12: import com.droidplanner.fragments.ParametersTableFragment;
   13: import com.droidplanner.fragments.RcSetupFragment;
   14: import com.droidplanner.fragments.SettingsFragment;
   15: import com.droidplanner.fragments.TuningFragment;
   16:
   17: public class ConfigurationActivity extends SuperUI{
   18:
   19:         public static final String SCREEN_INTENT = "screen";
   20:         public static final String TUNING = "tuning";
   21:         public static final String PARAMETERS = "parameters";
   22:     public static final String SETTINGS = "settings";
   23:         private static final String RC_SETUP = "rc_setup";
   24:         private FragmentTabHost mTabHost;
   25:
   26:         @Override
   27:         public void onCreate(Bundle savedInstanceState) {
   28:                 super.onCreate(savedInstanceState);
   29:                 setContentView(R.layout.activity_configuration);
   30:
   31:                 ActionBar actionBar = getActionBar();
   32:                 actionBar.setDisplayHomeAsUpEnabled(true);
   33:
   34:                 mTabHost = (FragmentTabHost)findViewById(R.id.configurationTabHost
);
   35:             mTabHost.setup(this, getFragmentManager(), R.id.realtabcontent);
   36:             mTabHost.addTab(mTabHost.newTabSpec(TUNING).setIndicator("Tuning"),
   37:                     TuningFragment.class, null);
   38:             mTabHost.addTab(mTabHost.newTabSpec(RC_SETUP).setIndicator("RC"),
   39:                     RcSetupFragment.class, null);
   40:             mTabHost.addTab(mTabHost.newTabSpec(PARAMETERS).setIndicator("Paramete
rs"),
   41:                     ParametersTableFragment.class, null);
   42:             mTabHost.addTab(mTabHost.newTabSpec(SETTINGS).setIndicator("Settings")
,
   43:                     SettingsFragment.class, null);
   44:
   45:             Intent intent = getIntent();
   46:             String stringExtra = intent.getStringExtra(SCREEN_INTENT);
   47:                 if(SETTINGS.equalsIgnoreCase(stringExtra)){
   48:                 mTabHost.setCurrentTabByTag(SETTINGS);
   49:             }
   50:         }
   51:
   52:         @Override
   53:         public boolean onOptionsItemSelected(MenuItem item) {
   54:                 switch (item.getItemId()) {
   55:                 case android.R.id.home:
   56:                         NavUtils.navigateUpFromSameTask(this);
   57:                         return true;
   58:                 }
   59:                 return super.onOptionsItemSelected(item);
   60:         }
   61:
   62: }
```

```
    1: package com.droidplanner.activitys;
    2:
    3: import java.util.List;
    4:
    5: import android.app.ActionBar;
    6: import android.app.FragmentManager;
    7: import android.graphics.Point;
    8: import android.os.Bundle;
    9: import android.support.v4.app.NavUtils;
   10: import android.view.MenuItem;
   11:
   12: import com.droidplanner.R;
   13: import com.droidplanner.activitys.helpers.SuperUI;
   14: import com.droidplanner.drone.variables.mission.Mission;
   15: import com.droidplanner.drone.variables.mission.MissionItem;
   16: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
   17: import com.droidplanner.fragments.EditorToolsFragment;
   18: import com.droidplanner.fragments.EditorToolsFragment.EditorTools;
   19: import com.droidplanner.fragments.EditorToolsFragment.OnEditorToolSelected;
   20: import com.droidplanner.fragments.PlanningMapFragment;
   21: import com.droidplanner.fragments.helpers.GestureMapFragment;
   22: import com.droidplanner.fragments.helpers.GestureMapFragment.OnPathFinishedListner
;
   23: import com.droidplanner.fragments.helpers.MapProjection;
   24: import com.droidplanner.fragments.helpers.OnMapInteractionListener;
   25: import com.droidplanner.fragments.mission.MissionDetailFragment;
   26: import com.droidplanner.fragments.mission.MissionDetailFragment.OnWayPointTypeChan
geListener;
   27: import com.droidplanner.polygon.PolygonPoint;
   28: import com.google.android.gms.maps.model.LatLng;
   29:
   30: public class EditorActivity extends SuperUI implements
   31:                 OnMapInteractionListener, OnPathFinishedListner, OnEditorToolSelec
ted,
   32:                 OnWayPointTypeChangeListener {
   33:
   34:         private PlanningMapFragment planningMapFragment;
   35:         private GestureMapFragment gestureMapFragment;
   36:         private Mission mission;
   37:         private EditorToolsFragment editorToolsFragment;
   38:         private MissionDetailFragment itemDetailFragment;
   39:         private FragmentManager fragmentManager;
   40:
   41:         @Override
   42:         public void onCreate(Bundle savedInstanceState) {
   43:                 super.onCreate(savedInstanceState);
   44:                 setContentView(R.layout.activity_editor);
   45:
   46:                 ActionBar actionBar = getActionBar();
   47:                 actionBar.setDisplayHomeAsUpEnabled(true);
   48:
   49:                 fragmentManager = getFragmentManager();
   50:
   51:                 planningMapFragment = ((PlanningMapFragment) fragmentManager
   52:                                 .findFragmentById(R.id.mapFragment));
   53:                 gestureMapFragment = ((GestureMapFragment) fragmentManager
   54:                                 .findFragmentById(R.id.gestureMapFragment));
   55:                 editorToolsFragment = (EditorToolsFragment) fragmentManager
   56:                                 .findFragmentById(R.id.editorToolsFragment);
   57:
   58:                 mission = drone.mission;
   59:                 gestureMapFragment.setOnPathFinishedListner(this);
   60:                 mission.onMissionUpdate();
   61:
   62:         }
   63:
   64:         @Override
   65:         protected void onDestroy() {
   66:                 super.onDestroy();
   67:                 mission.removeOnMissionUpdateListner(planningMapFragment);
   68:         }
   69:
   70:         @Override
   71:         public boolean onOptionsItemSelected(MenuItem item) {
   72:                 switch (item.getItemId()) {
   73:                 case android.R.id.home:
   74:                         NavUtils.navigateUpFromSameTask(this);
   75:                         return true;
   76:                 }
   77:                 return super.onOptionsItemSelected(item);
   78:         }
   79:
   80:         @Override
   81:         public boolean onMarkerClick(MissionItem wp) {
   82:                 showItemDetail(wp);
   83:                 return true;
   84:         }
   85:
   86:         @Override
   87:         public void onAddPoint(LatLng point) {
   88:                 // TODO Auto-generated method stub
   89:
   90:         }
   91:
   92:         @Override
   93:         public void onMoveHome(LatLng coord) {
   94:                 // TODO Auto-generated method stub
   95:
   96:         }
   97:
   98:         @Override
   99:         public void onMoveWaypoint(SpatialCoordItem waypoint, LatLng latLng) {
  100:                 // TODO Auto-generated method stub
  101:
  102:         }
  103:
  104:         @Override
  105:         public void onMovingWaypoint(SpatialCoordItem source, LatLng latLng) {
  106:                 // TODO Auto-generated method stub
  107:
  108:         }
  109:
  110:         @Override
  111:         public void onMovePolygonPoint(PolygonPoint source, LatLng newCoord) {
  112:                 // TODO Auto-generated method stub
  113:
  114:         }
  115:
  116:         @Override
  117:         public void onMapClick(LatLng point) {
  118:                 switch (editorToolsFragment.getTool()) {
  119:                 case MARKER:
  120:                         mission.addWaypoint(point, mission.getDefaultAlt());
  121:                         break;
  122:                 case DRAW:
  123:                         break;
  124:                 case POLY:
  125:                         break;
  126:                 case TRASH:
  127:                         break;
  128:                 }
  129:         }
  130:
  131:         @Override
```

```
  132:         public void editorToolChanged(EditorTools tools) {
  133:                 removeItemDetail(); // TODO remove this, used for debbuging
  134:                 switch (tools) {
  135:                 case DRAW:
  136:                 case POLY:
  137:                         gestureMapFragment.enableGestureDetection();
  138:                         break;
  139:                 case MARKER:
  140:                 case TRASH:
  141:                         gestureMapFragment.disableGestureDetection();
  142:                         break;
  143:                 }
  144:         }
  145:
  146:         private void showItemDetail(MissionItem item) {
  147:                 if (itemDetailFragment == null) {
  148:                         addItemDetail(item);
  149:                 } else {
  150:                         switchItemDetail(item);
  151:                 }
  152:         }
  153:
  154:         private void addItemDetail(MissionItem item) {
  155:                 itemDetailFragment = item.getDetailFragment();
  156:                 fragmentManager.beginTransaction()
  157:                                 .add(R.id.containerItemDetail, itemDetailFragment)
.commit();
  158:         }
  159:
  160:         private void switchItemDetail(MissionItem item) {
  161:                 itemDetailFragment = item.getDetailFragment();
  162:                 fragmentManager.beginTransaction()
  163:                                 .replace(R.id.containerItemDetail, itemDetailFragm
ent).commit();
  164:         }
  165:
  166:         private void removeItemDetail() {
  167:                 if (itemDetailFragment != null) {
  168:                         fragmentManager.beginTransaction().remove(itemDetailFragme
nt)
  169:                                         .commit();
  170:                         itemDetailFragment = null;
  171:                 }
  172:         }
  173:
  174:         @Override
  175:         public void onPathFinished(List<Point> path) {
  176:                 List<LatLng> points = MapProjection.projectPathIntoMap(path,
  177:                                 planningMapFragment.mMap);
  178:                 switch (editorToolsFragment.getTool()) {
  179:                 case DRAW:
  180:                         drone.mission.addWaypointsWithDefaultAltitude(points);
  181:                         break;
  182:                 case POLY:
  183:                         drone.mission.addSurveyPolygon(points);
  184:                         break;
  185:                 default:
  186:                         break;
  187:                 }
  188:                 editorToolsFragment.setTool(EditorTools.MARKER);
  189:         }
  190:
  191:         @Override
  192:         public void onWaypointTypeChanged(MissionItem newItem, MissionItem oldItem
) {
  193:                 mission.replace(oldItem, newItem);
  194:                 showItemDetail(newItem);
  195:         }
  196:
  197: }
```

```
    1: package com.droidplanner.activitys;
    2:
    3: import android.app.FragmentManager;
    4: import android.content.Intent;
    5: import android.os.Bundle;
    6: import android.view.View;
    7:
    8: import com.droidplanner.R;
    9: import com.droidplanner.activitys.helpers.SuperUI;
   10: import com.droidplanner.drone.DroneInterfaces.OnStateListner;
   11: import com.droidplanner.drone.variables.mission.MissionItem;
   12: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
   13: import com.droidplanner.fragments.MissionControlFragment.OnMissionControlInteracti
on;
   14: import com.droidplanner.fragments.RCFragment;
   15: import com.droidplanner.fragments.helpers.OnMapInteractionListener;
   16: import com.droidplanner.polygon.PolygonPoint;
   17: import com.google.android.gms.maps.model.LatLng;
   18:
   19: public class FlightActivity extends SuperUI implements
   20:                 OnMapInteractionListener, OnMissionControlInteraction, OnStateList
ner {
   21:         private FragmentManager fragmentManager;
   22:         private RCFragment rcFragment;
   23:         private View failsafeTextView;
   24:
   25:         @Override
   26:         public void onCreate(Bundle savedInstanceState) {
   27:                 super.onCreate(savedInstanceState);
   28:                 setContentView(R.layout.activity_flight);
   29:                 fragmentManager = getFragmentManager();
   30:
   31:                 failsafeTextView = findViewById(R.id.failsafeTextView);
   32:                 drone.state.addFlightStateListner(this);
   33:
   34:         }
   35:
   36:         @Override
   37:         protected void onDestroy() {
   38:                 super.onDestroy();
   39:                 drone.state.removeFlightStateListner(this);
   40:         }
   41:
   42:         @Override
   43:         public void onAddPoint(LatLng point) {
   44:                 // TODO Auto-generated method stub
   45:
   46:         }
   47:
   48:         @Override
   49:         public void onMoveHome(LatLng coord) {
   50:                 // TODO Auto-generated method stub
   51:
   52:         }
   53:
   54:         @Override
   55:         public void onMoveWaypoint(SpatialCoordItem waypoint, LatLng latLng) {
   56:                 // TODO Auto-generated method stub
   57:
   58:         }
   59:
   60:         @Override
   61:         public void onMovePolygonPoint(PolygonPoint source, LatLng newCoord) {
   62:                 // TODO Auto-generated method stub
   63:
   64:         }
   65:
   66:         @Override
   67:         public void onMapClick(LatLng point) {
   68:                 // TODO Auto-generated method stub
   69:
   70:         }
   71:
   72:         @Override
   73:         public boolean onMarkerClick(MissionItem wp) {
   74:                 // TODO Auto-generated method stub
   75:                 return false;
   76:         }
   77:
   78:         @Override
   79:         public void onJoystickSelected() {
   80:                 toggleRCFragment();
   81:         }
   82:
   83:         @Override
   84:         public void onPlanningSelected() {
   85:                 Intent navigationIntent;
   86:                 navigationIntent = new Intent(this, EditorActivity.class);
   87:                 startActivity(navigationIntent);
   88:         }
   89:
   90:         private void toggleRCFragment() {
   91:                 if (rcFragment == null) {
   92:                         rcFragment = new RCFragment();
   93:                         fragmentManager.beginTransaction()
   94:                                         .add(R.id.containerRC, rcFragment).commit(
);
   95:                 } else {
   96:                         fragmentManager.beginTransaction().remove(rcFragment).comm
it();
   97:                         rcFragment = null;
   98:                 }
   99:         }
  100:
  101:         @Override
  102:         public void onMovingWaypoint(SpatialCoordItem source, LatLng latLng) {
  103:                 // TODO Auto-generated method stub
  104:
  105:         }
  106:
  107:         @Override
  108:         public void onFlightStateChanged() {
  109:
  110:         }
  111:
  112:         @Override
  113:         public void onArmChanged() {
  114:
  115:         }
  116:
  117:         @Override
  118:         public void onFailsafeChanged() {
  119:                 if (drone.state.isFailsafe()) {
  120:                         failsafeTextView.setVisibility(View.VISIBLE);
  121:                 } else {
  122:                         failsafeTextView.setVisibility(View.GONE);
  123:                 }
  124:         }
  125:
  126: }
```

```
    1: package com.droidplanner.activitys.helpers;
    2:
    3: import android.content.Context;
    4: import android.view.Menu;
    5: import android.view.MenuInflater;
    6: import android.view.MenuItem;
    7:
    8: import com.droidplanner.R;
    9: import com.droidplanner.drone.Drone;
   10: import com.droidplanner.drone.DroneInterfaces.HomeDistanceChangedListner;
   11: import com.droidplanner.drone.DroneInterfaces.InfoListner;
   12: import com.droidplanner.drone.DroneInterfaces.OnStateListner;
   13: import com.droidplanner.widgets.TimerView;
   14: import com.droidplanner.widgets.spinners.SelectModeSpinner;
   15:
   16: public class InfoMenu implements InfoListner, HomeDistanceChangedListner,
   17:                 OnStateListner {
   18:         private Drone drone;
   19:         private MenuItem battery;
   20:         private MenuItem gps;
   21:         private MenuItem propeler;
   22:         private MenuItem home;
   23:         private MenuItem signal;
   24:         public SelectModeSpinner mode;
   25:
   26:         private TimerView timer;
   27:
   28:         public InfoMenu(Drone drone) {
   29:                 this.drone = drone;
   30:         }
   31:
   32:         public void inflateMenu(Menu menu, MenuInflater menuInflater) {
   33:                 if (drone.MavClient.isConnected()) {
   34:                         menuInflater.inflate(R.menu.menu_newui_connected, menu);
   35:                         battery = menu.findItem(R.id.bar_battery);
   36:                         gps = menu.findItem(R.id.bar_gps);
   37:                         propeler = menu.findItem(R.id.bar_propeller);
   38:                         home = menu.findItem(R.id.bar_home);
   39:                         signal = menu.findItem(R.id.bar_signal);
   40:                         mode = (SelectModeSpinner) menu.findItem(R.id.bar_mode)
   41:                                         .getActionView();
   42:
   43:                         timer = new TimerView(propeler);
   44:                         drone.setHomeChangedListner(this);
   45:                         drone.setInfoListner(this);
   46:                         drone.state.addFlightStateListner(this);
   47:
   48:                 } else {
   49:                         menuInflater.inflate(R.menu.menu_newui_disconnected, menu)
;
   50:                 }
   51:         }
   52:
   53:         public void onOptionsItemSelected(MenuItem item) {
   54:                 switch (item.getItemId()) {
   55:                 case R.id.bar_home:
   56:                         drone.waypointMananger.getWaypoints();
   57:                         break;
   58:                 }
   59:         }
   60:
   61:         @Override
   62:         public void onInfoUpdate() {
   63:                 battery.setTitle(String.format("%2.1fv, %2.0f%%",
   64:                                 drone.battery.getBattVolt(), drone.battery.getBatt
Remain()));
   65:                 gps.setTitle(String.format("%d, %s", drone.GPS.getSatCount(),
   66:                                 drone.GPS.getFixType()));
   67:         }
   68:
   69:         @Override
   70:         public void onDistanceToHomeHasChanged() {
   71:                 home.setTitle(drone.home.getDroneDistanceToHome().toString());
   72:         }
   73:
   74:         public void setupModeSpinner(Context context) {
   75:                 if (mode != null) {
   76:                         mode.buildSpinner(context, drone);
   77:                 }
   78:         }
   79:
   80:         @Override
   81:         public void onFlightStateChanged() {
   82:                 if (drone.state.isFlying()) {
   83:                         timer.reStart();
   84:                 }else{
   85:                         timer.stop();
   86:                 }
   87:         }
   88:
   89:         @Override
   90:         public void onArmChanged() {
   91:                 // TODO Auto-generated method stub
   92:
   93:         }
   94:
   95:         @Override
   96:         public void onFailsafeChanged() {
   97:                 // TODO Auto-generated method stub
   98:
   99:         }
  100:
  101:
  102: }
```

```
    1: package com.droidplanner.activitys.helpers;
    2:
    3: import android.app.Activity;
    4: import android.content.Context;
    5: import android.content.pm.ActivityInfo;
    6: import android.content.res.Configuration;
    7: import android.preference.PreferenceManager;
    8: import android.view.Surface;
    9: import android.view.WindowManager;
   10:
   11: public class ScreenOrientation {
   12:         public int screenRequestedOrientation;
   13:         private Activity activity;
   14:
   15:         public ScreenOrientation(Activity activity) {
   16:                 this.activity = activity;
   17:         }
   18:
   19:         public void requestLock() {
   20:                 if (isPrefLockOrientationSet()) {
   21:                         lockOrientation();
   22:                 }
   23:         }
   24:
   25:         public void unlock() {
   26:                 if (screenRequestedOrientation != ActivityInfo.SCREEN_ORIENTATION_
UNSPECIFIED) {
   27:                         screenRequestedOrientation = ActivityInfo.SCREEN_ORIENTATI
ON_UNSPECIFIED;
   28:                         setOrientation();
   29:                 }
   30:         }
   31:
   32:         private void setOrientation() {
   33:                 activity.setRequestedOrientation(screenRequestedOrientation);
   34:         }
   35:
   36:         private void lockOrientation() {
   37:                 int rotation = ((WindowManager) activity
   38:                                 .getSystemService(Context.WINDOW_SERVICE)).getDefa
ultDisplay()
   39:                                 .getRotation();
   40:                 int actualOrientation = activity.getResources().getConfiguration()
.orientation;
   41:                 boolean naturalOrientationLandscape = (((rotation == Surface.ROTAT
ION_0 || rotation == Surface.ROTATION_180) && actualOrientation == Configuration.ORIENTAT
ION_LANDSCAPE) || ((rotation == Surface.ROTATION_90 || rotation == Surface.ROTATION_270)
&& actualOrientation == Configuration.ORIENTATION_PORTRAIT));
   42:                 if (naturalOrientationLandscape) {
   43:                         switch (rotation) {
   44:                         case Surface.ROTATION_0:
   45:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_LANDSCAPE;
   46:                                 break;
   47:                         case Surface.ROTATION_90:
   48:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_REVERSE_PORTRAIT;
   49:                                 break;
   50:                         case Surface.ROTATION_180:
   51:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_REVERSE_LANDSCAPE;
   52:                                 break;
   53:                         case Surface.ROTATION_270:
   54:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_PORTRAIT;
   55:                                 break;
   56:                         }
   57:                 } else {
   58:                         switch (rotation) {
   59:                         case Surface.ROTATION_0:
   60:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_PORTRAIT;
   61:                                 break;
   62:                         case Surface.ROTATION_90:
   63:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_LANDSCAPE;
   64:                                 break;
   65:                         case Surface.ROTATION_180:
   66:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_REVERSE_PORTRAIT;
   67:                                 break;
   68:                         case Surface.ROTATION_270:
   69:                                 screenRequestedOrientation = ActivityInfo.SCREEN_O
RIENTATION_REVERSE_LANDSCAPE;
   70:                                 break;
   71:                         }
   72:                 }
   73:                 setOrientation();
   74:         }
   75:
   76:         private boolean isPrefLockOrientationSet() {
   77:                 return PreferenceManager.getDefaultSharedPreferences(
   78:                                 activity.getApplicationContext()).getBoolean(
   79:                                 "pref_lock_screen_orientation", false);
   80:         }
   81: }
```

```
    1: package com.droidplanner.activitys.helpers;
    2:
    3: import android.app.Activity;
    4: import android.content.Intent;
    5: import android.media.AudioManager;
    6: import android.os.Bundle;
    7: import android.preference.PreferenceManager;
    8: import android.view.MenuItem;
    9:
   10: import com.droidplanner.DroidPlannerApp;
   11: import com.droidplanner.DroidPlannerApp.OnSystemArmListener;
   12: import com.droidplanner.R;
   13: import com.droidplanner.activitys.ConfigurationActivity;
   14: import com.droidplanner.dialogs.AltitudeDialog;
   15: import com.droidplanner.dialogs.AltitudeDialog.OnAltitudeChangedListner;
   16: import com.droidplanner.dialogs.checklist.PreflightDialog;
   17: import com.droidplanner.drone.Drone;
   18: import com.droidplanner.fragments.helpers.OfflineMapFragment;
   19: import com.droidplanner.helpers.units.Altitude;
   20:
   21: public abstract class SuperActivity extends Activity implements
   22:                 OnAltitudeChangedListner, OnSystemArmListener {
   23:
   24:         public DroidPlannerApp app;
   25:         public Drone drone;
   26:         private MenuItem armButton;
   27:
   28:         public SuperActivity() {
   29:                 super();
   30:         }
   31:
   32:         @Override
   33:         public void onCreate(Bundle savedInstanceState) {
   34:                 super.onCreate(savedInstanceState);
   35:
   36:                 PreferenceManager.setDefaultValues(this, R.xml.preferences, false)
;
   37:
   38:                 app = (DroidPlannerApp) getApplication();
   39:                 app.onSystemArmListener = this;
   40:                 this.drone = app.drone;
   41:
   42:                 setVolumeControlStream(AudioManager.STREAM_MUSIC);
   43:         }
   44:
   45:         @Override
   46:         public boolean onMenuItemSelected(int featureId, MenuItem item) {
   47:                 switch (item.getItemId()) {
   48:                 case R.id.menu_configuration:
   49:                         startActivity(new Intent(this, ConfigurationActivity.class
));
   50:                         return true;
   51:                 case R.id.menu_settings:
   52:                         Intent intent = new Intent(this, ConfigurationActivity.cla
ss);
   53:                         intent.putExtra(ConfigurationActivity.SCREEN_INTENT,
   54:                                         ConfigurationActivity.SETTINGS);
   55:                         startActivity(intent);
   56:                         return true;
   57:                 case R.id.menu_connect:
   58:                         drone.MavClient.toggleConnectionState();
   59:                         return true;
   60:                 case R.id.menu_load_from_apm:
   61:                         drone.waypointMananger.getWaypoints();
   62:                         return true;
   63:                 case R.id.menu_default_alt:
   64:                         changeDefaultAlt();
   65:                         return true;
   66:                 case R.id.menu_preflight_calibration:
   67:                         drone.calibrationSetup.startCalibration(this);
   68:                         return true;
   69:                 case R.id.menu_record_me:
   70:                         app.recordMe.toogleRecordMeState();
   71:                         return true;
   72:                 case R.id.menu_follow_me:
   73:                         app.followMe.toogleFollowMeState();
   74:                         return true;
   75:                 case R.id.menu_preflight_checklist:
   76:                         showCheckList();
   77:                         return true;
   78:                 case R.id.menu_map_type_hybrid:
   79:                 case R.id.menu_map_type_normal:
   80:                 case R.id.menu_map_type_terrain:
   81:                 case R.id.menu_map_type_satellite:
   82:                         setMapTypeFromItemId(item.getItemId());
   83:                         return true;
   84:                 default:
   85:                         return super.onMenuItemSelected(featureId, item);
   86:                 }
   87:         }
   88:
   89:         private void showCheckList() {
   90:                 PreflightDialog dialog = new PreflightDialog();
   91:                 dialog.build(this, drone, false);
   92:
   93:         }
   94:
   95:         private void setMapTypeFromItemId(int itemId) {
   96:                 final String mapType;
   97:                 switch (itemId) {
   98:                 case R.id.menu_map_type_hybrid:
   99:                         mapType = OfflineMapFragment.MAP_TYPE_HYBRID;
  100:                         break;
  101:                 case R.id.menu_map_type_normal:
  102:                         mapType = OfflineMapFragment.MAP_TYPE_NORMAL;
  103:                         break;
  104:                 case R.id.menu_map_type_terrain:
  105:                         mapType = OfflineMapFragment.MAP_TYPE_TERRAIN;
  106:                         break;
  107:                 default:
  108:                         mapType = OfflineMapFragment.MAP_TYPE_SATELLITE;
  109:                         break;
  110:                 }
  111:
  112:                 PreferenceManager.getDefaultSharedPreferences(this).edit()
  113:                                 .putString(OfflineMapFragment.PREF_MAP_TYPE, mapTy
pe).commit();
  114:
  115:                 drone.notifyMapTypeChanged();
  116:         }
  117:
  118:         public void notifyArmed() {
  119:                 if (armButton != null) {
  120:                         armButton.setTitle(getResources().getString(R.string.menu_
disarm));
  121:                 }
  122:         }
  123:
  124:         public void notifyDisarmed() {
  125:                 if (armButton != null) {
  126:                         armButton.setTitle(getResources().getString(R.string.menu_
arm));
  127:                 }
  128:         }
```

```
129:
130:         public void changeDefaultAlt() {
131:                 AltitudeDialog dialog = new AltitudeDialog(this);
132:                 dialog.build(drone.mission.getDefaultAlt(), this);
133:         }
134:
135:         @Override
136:         public void onAltitudeChanged(Altitude newAltitude) {
137:                 drone.mission.setDefaultAlt(newAltitude);
138:         }
139: }
```

```
    1: package com.droidplanner.activitys.helpers;
    2:
    3: import android.os.Bundle;
    4: import android.view.Menu;
    5: import android.view.MenuItem;
    6:
    7: import com.droidplanner.DroidPlannerApp.ConnectionStateListner;
    8:
    9: public abstract class SuperUI extends SuperActivity implements ConnectionStateList
ner {
   10:         private ScreenOrientation screenOrientation = new ScreenOrientation(this);
   11:         private InfoMenu infoMenu;
   12:
   13:         public SuperUI() {
   14:                 super();
   15:         }
   16:
   17:         @Override
   18:         public void onCreate(Bundle savedInstanceState) {
   19:                 super.onCreate(savedInstanceState);
   20:                 screenOrientation.unlock();
   21:                 infoMenu = new InfoMenu(drone);
   22:         }
   23:
   24:         @Override
   25:         protected void onStart() {
   26:                 super.onStart();
   27:                 app.conectionListner = this;
   28:                 drone.MavClient.queryConnectionState();
   29:         }
   30:
   31:         @Override
   32:         public boolean onCreateOptionsMenu(Menu menu) {
   33:                 infoMenu.inflateMenu(menu, getMenuInflater());
   34:                 infoMenu.setupModeSpinner(this);
   35:                 return super.onCreateOptionsMenu(menu);
   36:         }
   37:
   38:         @Override
   39:         public boolean onOptionsItemSelected(MenuItem item) {
   40:                 infoMenu.onOptionsItemSelected(item);
   41:                 return super.onOptionsItemSelected(item);
   42:         }
   43:
   44:         public void notifyDisconnected() {
   45:                 invalidateOptionsMenu();
   46:                 /*
   47:                 if(armButton != null){
   48:                         armButton.setEnabled(false);
   49:                 }*/
   50:                 screenOrientation.unlock();
   51:         }
   52:
   53:         public void notifyConnected() {
   54:                 invalidateOptionsMenu();
   55:
   56:                 /*
   57:                 if(armButton != null){
   58:                         armButton.setEnabled(true);
   59:                 }
   60:                 */
   61:                 screenOrientation.requestLock();
   62:         }
   63:
   64: }
```

```
    1: package com.droidplanner.activitys;
    2:
    3: import java.util.List;
    4:
    5: import android.app.AlertDialog;
    6: import android.content.DialogInterface;
    7: import android.os.Bundle;
    8: import android.widget.Toast;
    9:
   10: import com.droidplanner.R;
   11: import com.droidplanner.activitys.helpers.SuperActivity;
   12: import com.droidplanner.drone.DroneInterfaces;
   13: import com.droidplanner.parameters.Parameter;
   14: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
   15:
   16: public class PidActivity extends SuperActivity implements
   17:                 DroneInterfaces.OnParameterManagerListner,
   18:                 DialogInterface.OnClickListener {
   19:
   20:         private SeekBarWithText rollPSeekBar;
   21:         private SeekBarWithText rollDSeekBar;
   22:         private SeekBarWithText yawPSeekBar;
   23:         private SeekBarWithText thrAclSeekBar;
   24:         private SeekBarWithText thrMidSeekBar;
   25:
   26:         private Parameter rollP;
   27:         private Parameter rollD;
   28:         private Parameter yawP;
   29:         private Parameter thrAcl;
   30:         private Parameter thrMid;
   31:
   32:         private DroneInterfaces.OnParameterManagerListner parameterListener;
   33:
   34:         @Override
   35:         public void onCreate(Bundle savedInstanceState) {
   36:                 super.onCreate(savedInstanceState);
   37:                 setContentView(R.layout.pid);
   38:                 findLocalViews();
   39:                 setupLocalViews();
   40:         }
   41:
   42:         private void setupLocalViews() {
   43:                 rollPSeekBar = (SeekBarWithText) findViewById(R.id.SeekBarRollPitc
hControl);
   44:                 rollDSeekBar = (SeekBarWithText) findViewById(R.id.SeekBarRollPitc
hDampenning);
   45:                 yawPSeekBar = (SeekBarWithText) findViewById(R.id.SeekBarYawContro
l);
   46:                 thrAclSeekBar = (SeekBarWithText) findViewById(R.id.SeekBarThrottl
eAccel);
   47:                 thrMidSeekBar = (SeekBarWithText) findViewById(R.id.seekBarThrottl
eHover);
   48:         }
   49:
   50:         private void findLocalViews() {
   51:                 // TODO Auto-generated method stub
   52:
   53:         }
   54:
   55:         private void refreshPIDValues() {
   56:                 if (drone.MavClient.isConnected()) {
   57:                         Toast.makeText(this, "Retreiving PID Values", Toast.LENGTH
_SHORT)
   58:                                         .show();
   59:                         drone.parameters.getAllParameters();
   60:                 } else {
   61:                         Toast.makeText(this, "Please connect first", Toast.LENGTH_
SHORT)
   62:                                         .show();
   63:                 }
   64:                 // drone.parameters.ReadParameter("RATE_RLL_P");
   65:                 // drone.parameters.ReadParameter("RATE_RLL_D");
   66:                 // drone.parameters.ReadParameter("RATE_YAW_P");
   67:                 // drone.parameters.ReadParameter("THR_ACCEL_P");
   68:                 // drone.parameters.ReadParameter("THR_MID");
   69:         }
   70:
   71:         private void updatePIDValues() {
   72:                 if(rollP==null || rollD==null || yawP==null || thrAcl==null || thr
Mid==null)
   73:                         return;
   74:
   75:                 //This is to check if data has changed - but it does not work here
 - Need to find out why
   76:                 if(rollP.value!=rollPSeekBar.getValue() ||
   77:                         rollD.value!=rollDSeekBar.getValue()||
   78:                         yawP.value!=yawPSeekBar.getValue()||
   79:                         thrAcl.value!=thrAclSeekBar.getValue()||
   80:                         thrMid.value!=thrMidSeekBar.getValue()){
   81:
   82:                         AlertDialog.Builder builder = new AlertDialog.Builder(this
);
   83:
   84:                 builder.setMessage(
   85:                         "Update changes to vehicle?")
   86:                         .setCancelable(false)
   87:                         .setPositiveButton("Ok",this)
   88:                         .setNegativeButton("Cancel",this);
   89:                 AlertDialog alert = builder.create();
   90:                 alert.show();
   91:                 }
   92:     }
   93:
   94:         private void doUpdatePIDValues() {
   95:                 rollP.value=rollPSeekBar.getValue();
   96:                 rollD.value=rollDSeekBar.getValue();
   97:                 yawP.value=yawPSeekBar.getValue();
   98:                 thrAcl.value=thrAclSeekBar.getValue();
   99:                 thrMid.value=thrMidSeekBar.getValue();
  100:
  101:                 drone.parameters.sendParameter(rollP);
  102:                 drone.parameters.sendParameter(rollD);
  103:                 drone.parameters.sendParameter(yawP);
  104:                 drone.parameters.sendParameter(thrAcl);
  105:                 drone.parameters.sendParameter(thrMid);
  106:                 Toast.makeText(this, "PID values updated", Toast.LENGTH_SHORT).sho
w();
  107:         }
  108:
  109:         public boolean onParameterReceived(Parameter parameter) {
  110:                 if (parameter.name.equals("RATE_RLL_P")) {
  111:                         Toast.makeText(this, "Rate Roll/Pitch control received",
  112:                                         Toast.LENGTH_SHORT).show();
  113:                         rollP = parameter;
  114:                         rollPSeekBar.setValue(rollP.value);
  115:                 }
  116:                 if (parameter.name.equals("RATE_RLL_D")) {
  117:                         Toast.makeText(this, "Rate Roll/Pitch dampening received",
  118:                                         Toast.LENGTH_SHORT).show();
  119:                         rollD = parameter;
  120:                         rollDSeekBar.setValue(rollD.value);
  121:                 }
  122:                 if (parameter.name.equals("RATE_YAW_P")) {
  123:                         Toast.makeText(this, "Rate Yaw control received",
```

```
  124:                                    Toast.LENGTH_SHORT).show();
  125:                        yawP = parameter;
  126:                        yawPSeekBar.setValue(yawP.value);
  127:                }
  128:                if (parameter.name.equals("THR_ACCEL_P")) {
  129:                        Toast.makeText(this, "Rate Throttle accelration received",
  130:                                        Toast.LENGTH_SHORT).show();
  131:                        thrAcl = parameter;
  132:                        thrAclSeekBar.setValue(thrAcl.value);
  133:                }
  134:                if (parameter.name.equals("THR_MID")) {
  135:                        Toast.makeText(this, "Throttle hover received", Toast.LENG
TH_SHORT)
  136:                                        .show();
  137:                        thrMid = parameter;
  138:                        thrMidSeekBar.setValue(thrMid.value);
  139:                }
  140:                return true;
  141:        }
  142:
  143:        @Override
  144:        public void onResume() {
  145:                parameterListener = drone.parameters.parameterListner;
  146:                drone.parameters.parameterListner = this;
  147:                super.onResume();
  148:
  149:                refreshPIDValues();
  150:        }
  151:
  152:        @Override
  153:        public void onPause() {
  154:                drone.parameters.parameterListner = parameterListener;
  155:                super.onPause();
  156:        }
  157:
  158:        @Override
  159:        public void onClick(DialogInterface arg0, int arg1) {
  160:                switch (arg1) {
  161:                case -1:
  162:                        doUpdatePIDValues();
  163:                        break;
  164:                case -2:
  165:                        break;
  166:                }
  167:
  168:        }
  169:
  170:        @Override
  171:        public void onBeginReceivingParameters() {
  172:                // TODO Auto-generated method stub
  173:
  174:        }
  175:
  176:        @Override
  177:        public void onParameterReceived(Parameter parameter, int index, int count)
 {
  178:                // TODO Auto-generated method stub
  179:
  180:        }
  181:
  182:        @Override
  183:        public void onEndReceivingParameters(List<Parameter> parameter) {
  184:                // TODO Auto-generated method stub
  185:
  186:        }
  187:
  188:        @Override
  189:        public void onParamterMetaDataChanged() {
  190:                // TODO Auto-generated method stub
  191:
  192:        }
  193:
  194: }
```

```
    1: package com.droidplanner.checklist;
    2:
    3: import java.util.ArrayList;
    4: import java.util.HashMap;
    5: import java.util.List;
    6:
    7: import com.droidplanner.R;
    8: import com.droidplanner.checklist.listadapter.ListXmlAdapter;
    9: import com.droidplanner.checklist.row.ListRow;
   10: import com.droidplanner.checklist.row.ListRow_CheckBox;
   11: import com.droidplanner.checklist.row.ListRow_Interface;
   12: import com.droidplanner.checklist.row.ListRow_Interface.OnRowItemChangeListener;
   13: import com.droidplanner.checklist.row.ListRow_Level;
   14: import com.droidplanner.checklist.row.ListRow_Note;
   15: import com.droidplanner.checklist.row.ListRow_Radio;
   16: import com.droidplanner.checklist.row.ListRow_Select;
   17: import com.droidplanner.checklist.row.ListRow_Switch;
   18: import com.droidplanner.checklist.row.ListRow_Toggle;
   19: import com.droidplanner.checklist.row.ListRow_Type;
   20: import com.droidplanner.checklist.row.ListRow_Value;
   21: import com.droidplanner.drone.Drone;
   22:
   23: import android.graphics.Typeface;
   24: import android.view.LayoutInflater;
   25: import android.view.View;
   26: import android.widget.TextView;
   27:
   28: public class CheckListAdapter extends ListXmlAdapter implements
   29:                 OnRowItemChangeListener {
   30:
   31:         public interface OnCheckListItemUpdateListener {
   32:                 public void onRadioGroupUpdate(CheckListItem checkListItem, int ch
eckId);
   33:
   34:                 public void onSelectUpdate(CheckListItem checkListItem, int select
Id);
   35:
   36:                 public void onCheckBoxUpdate(CheckListItem checkListItem,
   37:                                 boolean isChecked);
   38:
   39:                 public void onSwitchUpdate(CheckListItem checkListItem,
   40:                                 boolean isSwitched);
   41:
   42:                 public void onToggleUpdate(CheckListItem checkListItem,
   43:                                 boolean isToggled);
   44:
   45:                 public void onValueUpdate(CheckListItem checkListItem, String newV
alue);
   46:         }
   47:
   48:         private OnCheckListItemUpdateListener listener;
   49:
   50:         public CheckListAdapter(Drone drone, LayoutInflater inflater,
   51:                         List<String> listHeader,
   52:                         HashMap<String, List<CheckListItem>> listDataChild) {
   53:                 super(inflater, listHeader);
   54:
   55:                 setHeaderLayout(R.layout.list_group_header);
   56:
   57:                 for (String dataHeader : listHeader) {
   58:                         List<ListRow_Interface> xmlRows = new ArrayList<ListRow_In
terface>();
   59:                         for (CheckListItem listItem : listDataChild.get(dataHeader
)) {
   60:                                 if (listItem.getTagName().equalsIgnoreCase("check_
item")) {
   61:                                         ListRow_CheckBox row = new ListRow_CheckBo
x(this.inflater,
   62:                                                         listItem);
   63:                                         row.setOnRowItemChangeListener(this);
   64:                                         xmlRows.add(row);
   65:
   66:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"value_item")) {
   67:                                         ListRow_Value row = new ListRow_Value(this
.inflater,
   68:                                                         listItem);
   69:                                         row.setOnRowItemChangeListener(this);
   70:                                         xmlRows.add(row);
   71:
   72:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"radio_item")) {
   73:                                         ListRow_Radio row = new ListRow_Radio(this
.inflater,
   74:                                                         listItem);
   75:                                         row.setOnRowItemChangeListener(this);
   76:                                         xmlRows.add(row);
   77:
   78:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"select_item")) {
   79:                                         ListRow_Select row = new ListRow_Select(th
is.inflater,listItem);
   80:                                         row.setOnRowItemChangeListener(this);
   81:                                         xmlRows.add(row);
   82:
   83:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"toggle_item")) {
   84:                                         ListRow_Toggle row = new ListRow_Toggle(dr
one, this.inflater, listItem);
   85:                                         row.setOnRowItemChangeListener(this);
   86:                                         xmlRows.add(row);
   87:
   88:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"switch_item")) {
   89:                                         ListRow_Switch row = new ListRow_Switch(dr
one, this.inflater, listItem);
   90:                                         row.setOnRowItemChangeListener(this);
   91:                                         xmlRows.add(row);
   92:
   93:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"level_item")) {
   94:                                         ListRow_Level row = new ListRow_Level(dron
e, this.inflater, listItem);
   95:                                         xmlRows.add(row);
   96:                                 } else if (listItem.getTagName().equalsIgnoreCase(
"note_item")) {
   97:                                         ListRow_Note row = new ListRow_Note(this.i
nflater, listItem);
   98:                                         xmlRows.add(row);
   99:
  100:                                 }
  101:                         }
  102:                         listItems.put(dataHeader, xmlRows);
  103:                 }
  104:         }
  105:
  106:         public void setOnCheckListItemUpdateListener(
  107:                         OnCheckListItemUpdateListener listener) {
  108:                 this.listener = listener;
  109:         }
  110:
  111:         @Override
  112:         public void updateRatioValue(TextView lblChkRatio, int groupPosition){
  113:                 int childCount = getChildrenCount(groupPosition);
```

```
  114:                int childVerified = getChildrenVerified(groupPosition);
  115:                int childMandatory = getChildrenMandatory(groupPosition);
  116:
  117:                if(childVerified!=childMandatory)
  118:                        lblChkRatio.setTextColor(0xfff9093d);
  119:                else
  120:                        lblChkRatio.setTextColor(0xff09f93d);
  121:
  122:                lblChkRatio.setTypeface(null, Typeface.BOLD);
  123:                lblChkRatio.setText(String.format("%d/%d [%d]", childVerified, chi
ldCount, childMandatory));
  124:        }
  125:
  126:        private int getChildrenVerified(int groupPosition) {
  127:                int verified = 0;
  128:                for(int c=0;c<getChildrenCount(groupPosition);c++){
  129:                        ListRow row = (ListRow) getChild(groupPosition,c);
  130:                        CheckListItem listItem = row.getCheckListItem();
  131:                        if(listItem.isVerified())
  132:                                verified++;
  133:                }
  134:                return verified;
  135:        }
  136:
  137:        private int getChildrenMandatory(int groupPosition) {
  138:                int count = 0;
  139:                for(int c=0;c<getChildrenCount(groupPosition);c++){
  140:                        ListRow row = (ListRow) getChild(groupPosition,c);
  141:                        CheckListItem listItem = row.getCheckListItem();
  142:                        if(listItem.isMandatory())
  143:                                count++;
  144:                }
  145:                return count;
  146:        }
  147:
  148:        @Override
  149:        public int getChildTypeCount() {
  150:                return ListRow_Type.values().length;
  151:        }
  152:
  153:        @Override
  154:        public boolean hasStableIds() {
  155:                return false;
  156:        }
  157:
  158:        @Override
  159:        public boolean isChildSelectable(int groupPosition, int childPosition) {
  160:                return true;
  161:        }
  162:
  163:        @Override
  164:        public void onRowItemChanged(View mView, CheckListItem listItem, boolean i
sChecked) {
  165:                if(this.listener==null)
  166:                        return;
  167:
  168:                if(listItem.getTagName().equalsIgnoreCase("check_item")){
  169:                        this.listener.onCheckBoxUpdate(listItem, isChecked);
  170:                }else if(listItem.getTagName().equalsIgnoreCase("switch_item")){
  171:                        this.listener.onSwitchUpdate(listItem, listItem.isSys_acti
vated());
  172:                }else if(listItem.getTagName().equalsIgnoreCase("toggle_item")){
  173:                        this.listener.onToggleUpdate(listItem, listItem.isSys_acti
vated());
  174:                }else if(listItem.getTagName().equalsIgnoreCase("select_item")){
  175:                        this.listener.onSelectUpdate(listItem, listItem.getSelecte
dIndex());
  176:                }if(listItem.getTagName().equalsIgnoreCase("radio_item")){
  177:                        this.listener.onRadioGroupUpdate(listItem, listItem.getSel
ectedIndex());
  178:                }if(listItem.getTagName().equalsIgnoreCase("value_item")){
  179:                        this.listener.onValueUpdate(listItem, listItem.getValue())
;
  180:                }
  181:        }
  182: }
```

```java
package com.droidplanner.checklist;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.droidplanner.checklist.xml.ListXmlData;

public class CheckListItem extends ListXmlData {
        private int categoryIndex;
        private String title;
        private String desc;
        private String unit;
        private String on_label;
        private String off_label;
        private String sys_tag;
        private String value;
        private boolean editable;
        private boolean mandatory;
        private int selectedIndex;
        private float min_val;
        private float nom_val;
        private float max_val;
        private double sys_value;
        private boolean verified;
        private boolean sys_activated;
        private List<String> optionLists;

        public CheckListItem(String mTagName) {
                super(mTagName);
        }

        public CheckListItem(int mcategoryIndex, String mTagName, String mTitle,
                        String mDescription, String mSysTag, String mMandatory,
                        String mEditable) {
                super(mTagName);
                this.setCategoryIndex(mcategoryIndex);
                this.setTitle(mTitle);
                this.setDesc(mDescription);
                this.setSys_tag(mSysTag);
                this.setMandatory(mMandatory);
                this.setEditable(mEditable);
        }

        public String getTitle() {
                if (title == null)
                        return "No Title";
                if (isMandatory())
                        return "* " +title;
                else
                        return title;
        }

        public void setTitle(String title) {
                this.title = title;
        }

        public String getDesc() {
                if (desc == null)
                        return "";
                return desc;
        }

        public void setDesc(String desc) {
                this.desc = desc;
        }

        public String getSys_tag() {
                return sys_tag;
        }

        public void setSys_tag(String sys_tag) {
                this.sys_tag = sys_tag;
        }

        public boolean isEditable() {
                return editable;
        }

        public void setEditable(boolean editable) {
                this.editable = editable;
        }

        public void setEditable(String editable) {
                if (editable != null) {
                        this.editable = editable.equalsIgnoreCase("yes") ? true :
false;
                }
        }

        public boolean isMandatory() {
                return mandatory;
        }

        public void setMandatory(boolean mandatory) {
                this.mandatory = mandatory;
        }

        public void setMandatory(String mandatory) {
                if (mandatory != null) {
                        this.mandatory = mandatory.equalsIgnoreCase("yes") ? true
: false;
                }
        }

        public float getMin_val() {
                return min_val;
        }

        public void setMin_val(float min_val) {
                this.min_val = min_val;
        }

        public void setMin_val(String min_val) {
                if (min_val != null) {
                        try {
                                this.min_val = Float.parseFloat(min_val);
                        } catch (NumberFormatException e) {
                                this.min_val = 0;
                                e.printStackTrace();
                        }

                }
        }

        public float getNom_val() {
                return nom_val;
        }

        public void setNom_val(float nom_val) {
                this.nom_val = nom_val;
        }

        public void setNom_val(String nom_val) {
```

```
133:                 if (nom_val != null) {
134:                         try {
135:                                 this.nom_val = Float.parseFloat(nom_val);
136:                         } catch (NumberFormatException e) {
137:                                 this.nom_val = 0;
138:                                 e.printStackTrace();
139:                         }
140:
141:                 }
142:         }
143:
144:         public float getMax_val() {
145:                 return max_val;
146:         }
147:
148:         public void setMax_val(float max_val) {
149:                 this.max_val = max_val;
150:         }
151:
152:         public void setMax_val(String max_val) {
153:                 if (max_val != null) {
154:                         try {
155:                                 this.max_val = Float.parseFloat(max_val);
156:                         } catch (NumberFormatException e) {
157:                                 this.max_val = 0;
158:                                 e.printStackTrace();
159:                         }
160:
161:                 }
162:         }
163:
164:         public int getSelectedIndex() {
165:                 return selectedIndex;
166:         }
167:
168:         public void setSelectedIndex(int selectedIndex) {
169:                 this.selectedIndex = selectedIndex;
170:         }
171:
172:         public void setSelectedIndex(String selectedIndex) {
173:                 if (selectedIndex != null) {
174:                         try {
175:                                 this.selectedIndex = Integer.parseInt(selectedInde
x);
176:                         } catch (NumberFormatException e) {
177:                                 this.selectedIndex = -1;
178:                                 e.printStackTrace();
179:                         }
180:                 }
181:         }
182:
183:         public List<String> getOptionLists() {
184:                 return optionLists;
185:         }
186:
187:         public void setOptionLists(String optionListStr) {
188:                 this.optionLists = null;
189:
190:                 if (optionListStr != null) {
191:                         this.optionLists = new ArrayList<String>(
192:                                         Arrays.asList(optionListStr.split("\\s*,\\
s*")));
193:
194:                         for (int i = 0; i < optionLists.size(); i++)
195:                                 System.out.println("option : " + optionLists.get(i
));
196:
197:                 }
198:         }
199:
200:         public int getCategoryIndex() {
201:                 return categoryIndex;
202:         }
203:
204:         public void setCategoryIndex(int categoryIndex) {
205:                 this.categoryIndex = categoryIndex;
206:         }
207:
208:         public String getUnit() {
209:                 return unit;
210:         }
211:
212:         public void setUnit(String unit) {
213:                 if (unit != null) {
214:                         this.unit = unit;
215:                 } else {
216:                         this.unit = "";
217:                 }
218:         }
219:
220:         public double getSys_value() {
221:                 return sys_value;
222:         }
223:
224:         public void setSys_value(double d) {
225:                 this.sys_value = d;
226:         }
227:
228:         public boolean isSys_activated() {
229:                 return sys_activated;
230:         }
231:
232:         public void setSys_activated(boolean sys_activated) {
233:                 this.sys_activated = sys_activated;
234:         }
235:
236:         public String getOn_label() {
237:                 if (on_label == null)
238:                         return "";
239:                 return on_label;
240:         }
241:
242:         public void setOn_label(String on_label) {
243:                 this.on_label = on_label;
244:         }
245:
246:         public String getOff_label() {
247:                 if (off_label == null)
248:                         return "";
249:                 return off_label;
250:         }
251:
252:         public void setOff_label(String off_label) {
253:                 this.off_label = off_label;
254:         }
255:
256:         public boolean isVerified() {
257:                 return verified;
258:         }
259:
260:         public void setVerified(boolean verified) {
261:                 this.verified = verified;
262:         }
263:
```

```
264:         public String getValue() {
265:                 if (value == null)
266:                         return "";
267:                 return value;
268:         }
269:
270:         public float getFloatValue() {
271:                 float fValue = (float) 0.0;
272:                 if (value != null) {
273:                         try {
274:                                 fValue = Float.parseFloat(value);
275:                         } catch (NumberFormatException e) {
276:                                 e.printStackTrace();
277:                         }
278:                 }
279:                 return fValue;
280:
281:         }
282:
283:         public void setValue(String value) {
284:                 this.value = value;
285:         }
286:
287: }
```

```java
package com.droidplanner.checklist;

import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;

import com.droidplanner.checklist.xml.ListXmlParser;

import android.content.Context;

public class CheckListXmlParser extends ListXmlParser {

	private List<String> categories;
	private List<CheckListItem> checkListItems;
	private String checkListTitle;
	private String checkListType;
	private String checkListVersion;

	public void setOnXMLParserError(OnXmlParserError listener) {
		errorListener = listener;
	}

	public CheckListXmlParser() {
		categories = new ArrayList<String>();
		checkListItems = new ArrayList<CheckListItem>();
	}

	public CheckListXmlParser(Context context, int resourceId) {
		categories = new ArrayList<String>();
		checkListItems = new ArrayList<CheckListItem>();
		getListItemsFromResource(context, resourceId);
	}

	public CheckListXmlParser(String ioFile) {
		categories = new ArrayList<String>();
		checkListItems = new ArrayList<CheckListItem>();
		try {
			getListItemsFromFile(ioFile);
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (XmlPullParserException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	private void process_category(XmlPullParser xpp) {
		String lbl = xpp.getAttributeValue(null, "label");
		if (lbl == null)
			lbl = "Unknown";
		categories.add(lbl);
		System.out.println("Category - " + lbl);

	}

	private void process_checkitems(XmlPullParser xpp) {

		String itemType = xpp.getName();

		if (categories.size() == 0)
			return;

		if (itemType != null) {
			checkListItems.add(new CheckListItem(xpp.getName()));
			CheckListItem checkListItem = checkListItems.get(checkListItems
					.size() - 1);
			checkListItem.setDepth(xpp.getDepth());
			checkListItem.setCategoryIndex(categories.size() - 1);
			checkListItem.setTitle(xpp.getAttributeValue(null, "title"));
			checkListItem.setDesc(xpp.getAttributeValue(null, "description"));
			checkListItem.setUnit(xpp.getAttributeValue(null, "unit"));
			checkListItem.setOn_label(xpp.getAttributeValue(null, "on_label"));
			checkListItem.setOff_label(xpp.getAttributeValue(null, "off_label"));
			checkListItem.setSys_tag(xpp.getAttributeValue(null, "system_tag"));
			checkListItem.setEditable(xpp.getAttributeValue(null, "editable"));
			checkListItem.setMandatory(xpp.getAttributeValue(null, "mandatory"));
			checkListItem.setNom_val(xpp.getAttributeValue(null, "nominal_val"));
			checkListItem.setMin_val(xpp.getAttributeValue(null, "minimum_val"));
			checkListItem.setMax_val(xpp.getAttributeValue(null, "maximum_val"));
			checkListItem.setValue(xpp.getAttributeValue(null, "value"));
			checkListItem.setSelectedIndex(xpp.getAttributeValue(null, "selectindex"));
			checkListItem.setOptionLists(xpp.getAttributeValue(null,"optionlist"));
		}
	}

	public List<String> getCategories() {
		return categories;
	}

	public List<CheckListItem> getCheckListItems() {
		return checkListItems;
	}

	@Override
	public void process_StartDocument(XmlPullParser xpp) {
		// TODO Auto-generated method stub

	}

	@Override
	public void process_EndDocument(XmlPullParser xpp) {
		// TODO Auto-generated method stub

	}

	@Override
	public void process_StartTag(XmlPullParser xpp) {
		if (xpp.getName().equalsIgnoreCase("category")) {
			process_category(xpp);
		} else if (xpp.getName().contains("_item")) {
			process_checkitems(xpp);
		} else if (xpp.getDepth()==1){
			this.checkListTitle = xpp.getAttributeValue(null, "title");
			this.checkListType = xpp.getAttributeValue(null, "type");
```

```
  119:                         this.checkListVersion = xpp.getAttributeValue(null, "versi
on");
  120:                 }
  121:         }
  122:
  123:         @Override
  124:         public void process_EndTag(XmlPullParser xpp) {
  125:                 // TODO Auto-generated method stub
  126:
  127:         }
  128:
  129:         @Override
  130:         public void process_Text(XmlPullParser xpp) {
  131:                 // TODO Auto-generated method stub
  132:
  133:         }
  134:
  135:         public String getCheckListTitle() {
  136:                 return checkListTitle;
  137:         }
  138:
  139:         public String getCheckListType() {
  140:                 return checkListType;
  141:         }
  142:
  143:         public String getCheckListVersion() {
  144:                 return checkListVersion;
  145:         }
  146: }
```

```
    1: package com.droidplanner.checklist.listadapter;
    2:
    3: import java.util.HashMap;
    4: import java.util.List;
    5:
    6: import com.droidplanner.R;
    7: import com.droidplanner.checklist.row.ListRow_Interface;
    8:
    9: import android.graphics.Typeface;
   10: import android.view.LayoutInflater;
   11: import android.view.View;
   12: import android.view.ViewGroup;
   13: import android.widget.BaseExpandableListAdapter;
   14: import android.widget.TextView;
   15:
   16: public abstract class ListXmlAdapter extends BaseExpandableListAdapter{
   17:
   18:         protected LayoutInflater inflater;
   19:         protected List<String> listHeader;
   20:         protected ListRow_Interface rowHeader;
   21:         protected HashMap<String, List<ListRow_Interface>> listItems;
   22:         protected int layoutId;
   23:
   24:         public ListXmlAdapter(LayoutInflater inflater,
   25:                         List<String> listHeader) {
   26:                 this.inflater = inflater;
   27:                 this.listHeader = listHeader;
   28:                 this.listItems = new HashMap<String, List<ListRow_Interface>>();
   29:         }
   30:
   31:         public void addRowItem(int groupPosition, int childPostion, ListRow_Interf
ace rowItem){
   32:
   33:         }
   34:
   35:         public void setHeaderLayout(int mLayoutId){
   36:                 this.layoutId = mLayoutId;
   37:         }
   38:
   39:         public void setRowHeader(ListRow_Interface mRowHeader){
   40:                 this.rowHeader = mRowHeader;
   41:         }
   42:
   43:         @Override
   44:         public Object getChild(int groupPosition, int childPosititon) {
   45:                 return this.listItems.get(
   46:                                 this.listHeader.get(groupPosition))
   47:                                 .get(childPosititon);
   48:         }
   49:
   50:         @Override
   51:         public long getChildId(int groupPosition, int childPosition) {
   52:                 return childPosition;
   53:         }
   54:
   55:         @Override
   56:         public View getChildView(int groupPosition, final int childPosition,
   57:                         boolean isLastChild, View convertView, ViewGroup parent) {
   58:                 ListRow_Interface row = (ListRow_Interface) getChild(groupPosition
, childPosition);
   59:                 return row.getView(convertView);
   60:         }
   61:
   62:         @Override
   63:         public int getChildrenCount(int groupPosition) {
   64:                 return this.listItems.get(
   65:                                 this.listHeader.get(groupPosition)).size();
   66:         }
   67:
   68:         @Override
   69:         public int getChildTypeCount() {
   70:                 return 0;
   71:         }
   72:
   73:         @Override
   74:         public int getChildType(int groupPosition, int childPosition) {
   75:                 return ((ListRow_Interface) getChild(groupPosition, childPosition)
).getViewType();
   76:         }
   77:
   78:         @Override
   79:         public Object getGroup(int groupPosition) {
   80:                 return this.listHeader.get(groupPosition);
   81:         }
   82:
   83:         @Override
   84:         public int getGroupCount() {
   85:                 return this.listHeader.size();
   86:         }
   87:
   88:         @Override
   89:         public long getGroupId(int groupPosition) {
   90:                 return groupPosition;
   91:         }
   92:
   93:         @Override
   94:         public View getGroupView(int groupPosition, boolean isExpanded,
   95:                         View convertView, ViewGroup parent) {
   96:                 String headerTitle = (String) getGroup(groupPosition);
   97:                 if (convertView == null) {
   98:                         LayoutInflater infalInflater = this.inflater;
   99:                         convertView = infalInflater.inflate(layoutId,
  100:                                         null);
  101:                 }
  102:
  103:                 TextView lblListHeader = (TextView) convertView
  104:                                 .findViewById(R.id.lblListHeader);
  105:                 lblListHeader.setTypeface(null, Typeface.BOLD);
  106:                 lblListHeader.setText(headerTitle);
  107:
  108:                 TextView lblChkRatio = (TextView) convertView.findViewById(R.id.lb
lChkRatio);
  109:                 updateRatioValue(lblChkRatio, groupPosition);
  110:
  111:                 return convertView;
  112:         }
  113:
  114:         public void updateRatioValue(TextView lblChkRatio, int groupPosition) {
  115:                 return;
  116:         }
  117:
  118:         @Override
  119:         public boolean hasStableIds() {
  120:                 return false;
  121:         }
  122:
  123:         @Override
  124:         public boolean isChildSelectable(int groupPosition, int childPosition) {
  125:                 return true;
  126:         }
  127: }
```

```
    1: package com.droidplanner.checklist.row;
    2:
    3: import com.droidplanner.R;
    4: import com.droidplanner.checklist.CheckListItem;
    5:
    6: import android.view.ViewGroup;
    7: import android.widget.CheckBox;
    8: import android.widget.LinearLayout;
    9:
   10: public class BaseViewHolder {
   11:         protected LinearLayout layoutView;
   12:         protected CheckBox checkBox;
   13:
   14:         public BaseViewHolder(ViewGroup viewGroup, CheckListItem checkListItem) {
   15:                 this.layoutView = (LinearLayout) viewGroup
   16:                                 .findViewById(R.id.lst_layout);
   17:                 this.checkBox = (CheckBox) viewGroup.findViewById(R.id.lst_check);
   18:                 setupViewItems(viewGroup, checkListItem);
   19:         }
   20:
   21:         protected void setupViewItems(ViewGroup viewGroup, CheckListItem checkList
Item){
   22:
   23:         }
   24: }
```

```
    1: package com.droidplanner.checklist.row;
    2:
    3: import com.droidplanner.checklist.CheckListItem;
    4: import com.droidplanner.drone.Drone;
    5:
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.View.OnClickListener;
    9: import android.view.View.OnLongClickListener;
   10: import android.widget.CheckBox;
   11: import android.widget.Toast;
   12:
   13: public class ListRow implements ListRow_Interface, OnClickListener, OnLongClickLis
tener {
   14:         protected final CheckListItem checkListItem;
   15:         protected final LayoutInflater inflater;
   16:         protected  Drone drone;
   17:
   18:         public OnRowItemChangeListener listener;
   19:         public BaseViewHolder holder;
   20:
   21:         public ListRow(LayoutInflater mInflater, CheckListItem mCheckListItem) {
   22:                 this.checkListItem = mCheckListItem;
   23:                 this.inflater = mInflater;
   24:         }
   25:         public ListRow(Drone mDrone,LayoutInflater mInflater, CheckListItem mCheck
ListItem) {
   26:                 this.checkListItem = mCheckListItem;
   27:                 this.inflater = mInflater;
   28:                 this.drone = mDrone;
   29:         }
   30:
   31:         protected void updateCheckBox(boolean mState) {
   32:
   33:                 // Common display update
   34:                 holder.layoutView.setOnLongClickListener(this);
   35:                 holder.checkBox.setOnClickListener(this);
   36:                 holder.checkBox.setText(checkListItem.getTitle());
   37:                 holder.checkBox.setChecked(mState);
   38:                 holder.checkBox
   39:                 .setClickable(checkListItem.getSys_tag() == null ? checkListItem
   40:                                 .isEditable() : !checkListItem.getSys_tag().contai
ns(
   41:                                 "SYS"));
   42:
   43:                 checkListItem.setVerified(mState);
   44:         }
   45:
   46:         public void updateRowChanged(View mView, CheckListItem mListItem) {
   47:                 if (listener != null)
   48:                         listener.onRowItemChanged(mView, this.checkListItem,
   49:                                         this.checkListItem.isVerified());
   50:         }
   51:         public CheckListItem getCheckListItem(){
   52:                 return checkListItem;
   53:         }
   54:
   55:         @Override
   56:         public View getView(View convertView) {
   57:                 // TODO Auto-generated method stub
   58:                 return null;
   59:         }
   60:
   61:         @Override
   62:         public int getViewType() {
   63:                 // TODO Auto-generated method stub
   64:                 return 0;
   65:         }
   66:
   67:         public void setOnRowItemChangeListener(OnRowItemChangeListener mListener)
{
   68:                 listener = mListener;
   69:         }
   70:
   71:         protected void getDroneVariable(Drone mDrone, CheckListItem mListItem) {
   72:                 String sys_tag = mListItem.getSys_tag();
   73:
   74:                 if (sys_tag.equalsIgnoreCase("SYS_BATTREM_LVL")) {
   75:                         mListItem.setSys_value(mDrone.battery.getBattRemain());
   76:                 } else if (sys_tag.equalsIgnoreCase("SYS_BATTVOL_LVL")) {
   77:                         mListItem.setSys_value(mDrone.battery.getBattVolt());
   78:                 } else if (sys_tag.equalsIgnoreCase("SYS_BATTCUR_LVL")) {
   79:                         mListItem.setSys_value(mDrone.battery.getBattCurrent());
   80:                 } else if (sys_tag.equalsIgnoreCase("SYS_GPS3D_LVL")) {
   81:                         mListItem.setSys_value(mDrone.GPS.getSatCount());
   82:                 } else if (sys_tag.equalsIgnoreCase("SYS_DEF_ALT")) {
   83:                         mListItem.setSys_value(mDrone.mission.getDefaultAlt().valu
eInMeters());
   84:                 } else if (sys_tag.equalsIgnoreCase("SYS_ARM_STATE")) {
   85:                         mListItem.setSys_activated(mDrone.state.isArmed());
   86:                 } else if (sys_tag.equalsIgnoreCase("SYS_FAILSAFE_STATE")) {
   87:                         mListItem.setSys_activated(mDrone.state.isFailsafe());
   88:                 } else if (sys_tag.equalsIgnoreCase("SYS_CONNECTION_STATE")) {
   89:                         mListItem.setSys_activated(mDrone.MavClient.isConnected())
;
   90:                 }
   91:         }
   92:
   93:         @Override
   94:         public void onClick(View v) {
   95:                 this.checkListItem.setVerified(((CheckBox) v).isChecked());
   96:                 updateRowChanged(v, this.checkListItem);
   97:         }
   98:
   99:         @Override
  100:         public boolean onLongClick(View arg0) {
  101:                 if(arg0.equals(this.holder.layoutView)){
  102:                         Toast.makeText(
  103:                                         arg0.getContext(),
  104:                                         checkListItem.getDesc(),
  105:                                         Toast.LENGTH_SHORT).show();
  106:
  107:                 }
  108:                 return false;
  109:         }
  110:
  111: }
```

```
    1: package com.droidplanner.checklist.row;
    2:
    3: import com.droidplanner.R;
    4: import com.droidplanner.checklist.CheckListItem;
    5:
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.ViewGroup;
    9:
   10: public class ListRow_CheckBox extends ListRow {
   11:
   12:         public ListRow_CheckBox(LayoutInflater inflater, CheckListItem checkListIt
em) {
   13:                 super(inflater, checkListItem);
   14:         }
   15:
   16:         public View getView(View convertView) {
   17:                 View view;
   18:                 if (convertView == null) {
   19:                         ViewGroup viewGroup = (ViewGroup) inflater.inflate(
   20:                                         R.layout.list_check_item, null);
   21:                         holder = new ViewHolder(viewGroup, checkListItem);
   22:                         viewGroup.setTag(holder);
   23:                         view = viewGroup;
   24:                 } else {
   25:                         holder = (ViewHolder) convertView.getTag();
   26:                         view = convertView;
   27:                 }
   28:
   29:                 updateDisplay(view, (ViewHolder)holder,checkListItem);
   30:                 return view;
   31:         }
   32:
   33:         private void updateDisplay(View view, ViewHolder holder,
   34:                         CheckListItem mListItem) {
   35:
   36:                 updateCheckBox(checkListItem.isVerified());
   37:         }
   38:
   39:         public int getViewType() {
   40:                 return ListRow_Type.CHECKBOX_ROW.ordinal();
   41:         }
   42:
   43:         private static class ViewHolder extends BaseViewHolder {
   44:
   45:                 private ViewHolder(ViewGroup viewGroup, CheckListItem checkListIte
m) {
   46:                         super(viewGroup, checkListItem);
   47:                 }
   48:         }
   49: }
```

```
    1: package com.droidplanner.checklist.row;
    2:
    3: import com.droidplanner.checklist.CheckListItem;
    4:
    5: import android.view.View;
    6:
    7: public interface ListRow_Interface {
    8:         public interface OnRowItemChangeListener{
    9:                 public void onRowItemChanged(View mView, CheckListItem listItem ,b
oolean isChecked);
   10:         }
   11:     public View getView(View convertView);
   12:     public int getViewType();
   13: }
```

```java
package com.droidplanner.checklist.row;

import com.droidplanner.R;
import com.droidplanner.checklist.CheckListItem;
import com.droidplanner.drone.Drone;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.CheckBox;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.TextView;

public class ListRow_Level extends ListRow {

	public ListRow_Level(Drone drone, LayoutInflater inflater,
			CheckListItem checkListItem) {
		super(drone,inflater, checkListItem);
	}

	public View getView(View convertView) {
		View view;
		if (convertView == null) {
			ViewGroup viewGroup = (ViewGroup) inflater.inflate(
					R.layout.list_level_item, null);
			holder = new ViewHolder(viewGroup, checkListItem);
			viewGroup.setTag(holder);
			view = viewGroup;
		} else {
			view = convertView;
			holder = (ViewHolder) convertView.getTag();
		}

		updateDisplay(view, (ViewHolder)holder, checkListItem);
		return view;
	}

	private void updateDisplay(View view, ViewHolder holder,
			CheckListItem mListItem) {
		int drawableId;
		double minVal = mListItem.getMin_val();
		double nomVal = mListItem.getNom_val();
		double sysValue = mListItem.getSys_value();
		String unit = mListItem.getUnit();
		boolean failMandatory = false;

		getDroneVariable(drone, mListItem);
		failMandatory = sysValue <= minVal;

		if (sysValue <= minVal)
			drawableId = R.drawable.pstate_poor;
		else if (sysValue > minVal && sysValue <= nomVal)
			drawableId = R.drawable.pstate_warning;
		else
			drawableId = R.drawable.pstate_good;

		holder.progressBar.setMax((int) mListItem.getMax_val());
		holder.progressBar.setProgressDrawable(view.getResources().getDrawable(
				drawableId));
		holder.progressBar.setProgress((int) sysValue);

		try {
			holder.unitView.setText(String.format(unit, sysValue));
		} catch (Exception e) {
			holder.unitView.setText("Error");
			e.printStackTrace();
		}

		updateCheckBox(checkListItem.isMandatory()&&!failMandatory);
/*
		if(holder.checkBox.isChecked())
			holder.layoutView.setBackgroundColor(getViewType());
		else
			holder.layoutView.setBackgroundColor(Color.parseColor("#4f0f00"));
*/

	}

	public int getViewType() {
		return ListRow_Type.LEVEL_ROW.ordinal();
	}

	private static class ViewHolder extends BaseViewHolder {
		private ProgressBar progressBar;
		private TextView unitView;

		private ViewHolder(ViewGroup viewGroup, CheckListItem checkListItem) {
			super(viewGroup,checkListItem);
		}

		@Override
		protected void setupViewItems(ViewGroup viewGroup, CheckListItem checkListItem){
			this.layoutView = (LinearLayout) viewGroup
					.findViewById(R.id.lst_layout);
			this.checkBox = (CheckBox) viewGroup.findViewById(R.id.lst_check);
			this.progressBar = (ProgressBar) viewGroup
					.findViewById(R.id.lst_level);
			this.unitView = (TextView) viewGroup.findViewById(R.id.lst_unit);
			this.progressBar.setMax((int) checkListItem.getMax_val());

		}
	}
}
```

```
 1: package com.droidplanner.checklist.row;
 2:
 3: import com.droidplanner.R;
 4: import com.droidplanner.checklist.CheckListItem;
 5:
 6: import android.view.LayoutInflater;
 7: import android.view.View;
 8: import android.view.View.OnFocusChangeListener;
 9: import android.view.ViewGroup;
10: import android.widget.EditText;
11:
12: public class ListRow_Note extends ListRow implements OnFocusChangeListener {
13:
14:         public ListRow_Note(LayoutInflater inflater,
15:                         final CheckListItem checkListItem) {
16:                 super(inflater, checkListItem);
17:         }
18:
19:         public View getView(View convertView) {
20:                 View view;
21:                 if (convertView == null) {
22:                         ViewGroup viewGroup = (ViewGroup) inflater.inflate(
23:                                         R.layout.list_note_item, null);
24:                         holder = new ViewHolder(viewGroup, checkListItem);
25:                         viewGroup.setTag(holder);
26:                         view = viewGroup;
27:                 } else {
28:                         view = convertView;
29:                         holder = (ViewHolder) convertView.getTag();
30:                 }
31:
32:                 updateDisplay(view, (ViewHolder)holder, checkListItem);
33:                 return view;
34:         }
35:
36:         private void updateDisplay(View view, ViewHolder holder,
37:                         CheckListItem mListItem) {
38:                 holder.editTextView.setOnFocusChangeListener(this);
39:                 holder.editTextView.setText(checkListItem.getValue());
40:
41:                 updateCheckBox(checkListItem.isVerified());
42:         }
43:
44:         public int getViewType() {
45:                 return ListRow_Type.NOTE_ROW.ordinal();
46:         }
47:
48:         private static class ViewHolder extends BaseViewHolder {
49:                 private EditText editTextView;
50:
51:                 private ViewHolder(ViewGroup viewGroup, CheckListItem checkListIte
m) {
52:                         super(viewGroup, checkListItem);
53:                 }
54:
55:                 @Override
56:                 protected void setupViewItems(ViewGroup viewGroup,
57:                                 CheckListItem checkListItem) {
58:                         this.editTextView = (EditText) viewGroup
59:                                         .findViewById(R.id.lst_note);
60:                 }
61:         }
62:
63:         @Override
64:         public void onFocusChange(View v, boolean hasFocus) {
65:                 if (!v.isFocused() && this.listener != null) {
66:                         checkListItem.setValue(((ViewHolder)this.holder).editTextV
iew.getText().toString());
67:                         updateRowChanged(v,this.checkListItem);
68:                 }
69:
70:         }
71: }
```

```
   1: package com.droidplanner.checklist.row;
   2:
   3: import java.util.List;
   4:
   5: import com.droidplanner.R;
   6: import com.droidplanner.checklist.CheckListItem;
   7:
   8: import android.view.LayoutInflater;
   9: import android.view.View;
  10: import android.view.ViewGroup;
  11: import android.view.ViewGroup.LayoutParams;
  12: import android.widget.RadioButton;
  13: import android.widget.RadioGroup;
  14: import android.widget.RadioGroup.OnCheckedChangeListener;
  15:
  16: public class ListRow_Radio extends ListRow implements OnCheckedChangeListener {
  17:
  18:         public ListRow_Radio(LayoutInflater inflater, CheckListItem checkListItem)
 {
  19:                 super(inflater, checkListItem);
  20:         }
  21:
  22:         public View getView(View convertView) {
  23:                 View view;
  24:                 if (convertView == null) {
  25:                         ViewGroup viewGroup = (ViewGroup) inflater.inflate(
  26:                                         R.layout.list_radio_item, null);
  27:                         holder = new ViewHolder(viewGroup, checkListItem);
  28:                         viewGroup.setTag(holder);
  29:                         view = viewGroup;
  30:                 } else {
  31:                         view = convertView;
  32:                         holder = (ViewHolder) convertView.getTag();
  33:                 }
  34:                 updateDisplay(view, (ViewHolder)holder, checkListItem);
  35:                 return view;
  36:         }
  37:
  38:         private void updateDisplay(View view, ViewHolder holder,
  39:                         CheckListItem mListItem) {
  40:                 holder.radioGroupView.setOnCheckedChangeListener(this);
  41:
  42:                 updateCheckBox(checkListItem.isVerified());
  43:
  44:         }
  45:
  46:         public int getViewType() {
  47:                 return ListRow_Type.RADIO_ROW.ordinal();
  48:         }
  49:
  50:         private static class ViewHolder extends BaseViewHolder {
  51:                 private RadioGroup radioGroupView;
  52:
  53:                 public ViewHolder(ViewGroup viewGroup, CheckListItem checkListItem
) {
  54:                         super(viewGroup,checkListItem);
  55:                 }
  56:
  57:                 @Override
  58:                 protected void setupViewItems(ViewGroup viewGroup,
  59:                                 CheckListItem checkListItem) {
  60:                         this.radioGroupView = (RadioGroup) viewGroup
  61:                                         .findViewById(R.id.lst_radioGroup);
  62:
  63:                         this.radioGroupView.removeAllViews();
  64:
  65:                         List<String> optionLists = checkListItem.getOptionLists();
  66:                         for (String optionlist : optionLists) {
  67:                                 RadioButton rButton = new RadioButton(viewGroup.ge
tContext());
  68:                                 rButton.setLayoutParams(new LayoutParams(
  69:                                                 LayoutParams.WRAP_CONTENT, LayoutP
arams.WRAP_CONTENT));
  70:                                 rButton.setText(optionlist);
  71:                                 rButton.setId(optionLists.indexOf(optionlist));
  72:                                 this.radioGroupView.addView(rButton);
  73:                         }
  74:                         this.radioGroupView.check(checkListItem.getSelectedIndex()
);
  75:
  76:                 }
  77:         }
  78:
  79:         @Override
  80:         public void onCheckedChanged(RadioGroup arg0, int arg1) {
  81:                 checkListItem.setSelectedIndex(arg1);
  82:                 updateRowChanged((View)arg0,this.checkListItem);
  83:
  84:         }
  85: }
```

```
    1: package com.droidplanner.checklist.row;
    2:
    3: import com.droidplanner.R;
    4: import com.droidplanner.checklist.CheckListItem;
    5:
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.ViewGroup;
    9: import android.widget.AdapterView;
   10: import android.widget.AdapterView.OnItemSelectedListener;
   11: import android.widget.ArrayAdapter;
   12: import android.widget.CheckBox;
   13: import android.widget.LinearLayout;
   14: import android.widget.Spinner;
   15:
   16: public class ListRow_Select extends ListRow implements OnItemSelectedListener{
   17:
   18:         public ListRow_Select(LayoutInflater inflater, CheckListItem checkListItem
) {
   19:                 super(inflater, checkListItem);
   20:         }
   21:
   22:         public View getView(View convertView) {
   23:                 View view;
   24:                 if (convertView == null) {
   25:                         ViewGroup viewGroup = (ViewGroup) inflater.inflate(
   26:                                         R.layout.list_select_item, null);
   27:                         holder = new ViewHolder(viewGroup, checkListItem);
   28:                         viewGroup.setTag(holder);
   29:                         view = viewGroup;
   30:                 } else {
   31:                         view = convertView;
   32:                         holder = (ViewHolder) convertView.getTag();
   33:                 }
   34:
   35:                 updateDisplay(view, (ViewHolder)holder, checkListItem);
   36:                 return view;
   37:         }
   38:
   39:         private void updateDisplay(View view, ViewHolder holder,
   40:                         CheckListItem mListItem) {
   41:                 holder.selectView.setOnItemSelectedListener(this);
   42:                 updateCheckBox(checkListItem.isVerified());
   43:         }
   44:
   45:         public int getViewType() {
   46:                 return ListRow_Type.SELECT_ROW.ordinal();
   47:         }
   48:
   49:         private static class ViewHolder extends BaseViewHolder {
   50:                 private Spinner selectView;
   51:                 @SuppressWarnings("unused")
   52:                 private CheckListItem checkListItem;
   53:
   54:                 private ArrayAdapter<String> adapter;
   55:
   56:                 private ViewHolder(ViewGroup viewGroup, CheckListItem checkListIte
m) {
   57:                         super(viewGroup, checkListItem);
   58:                         this.checkListItem = checkListItem;
   59:                 }
   60:
   61:                 @Override
   62:                 protected void setupViewItems(ViewGroup viewGroup, CheckListItem c
heckListItem){
   63:                         this.layoutView = (LinearLayout) viewGroup
   64:                                         .findViewById(R.id.lst_layout);
   65:                         this.selectView = (Spinner) viewGroup.findViewById(R.id.ls
t_select);
   66:                         this.checkBox = (CheckBox) viewGroup.findViewById(R.id.lst
_check);
   67:
   68:                         setupSpinner(viewGroup, checkListItem);
   69:                 }
   70:
   71:                 private void setupSpinner(ViewGroup viewGroup,
   72:                                 CheckListItem checkListItem) {
   73:                         adapter = new ArrayAdapter<String>(viewGroup.getContext(),
   74:                                         android.R.layout.simple_spinner_item,
   75:                                         checkListItem.getOptionLists());
   76:                         adapter.setDropDownViewResource(android.R.layout.simple_sp
inner_dropdown_item);
   77:
   78:                         selectView.setAdapter(adapter);
   79:
   80:                 }
   81:         }
   82:
   83:         @Override
   84:         public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2,
   85:                         long arg3) {
   86:                 checkListItem.setSelectedIndex(arg2);
   87:                 updateRowChanged(arg1,this.checkListItem);
   88:         }
   89:
   90:         @Override
   91:         public void onNothingSelected(AdapterView<?> arg0) {
   92:                 // TODO Auto-generated method stub
   93:         }
   94:
   95: }
```

```
 1: package com.droidplanner.checklist.row;
 2:
 3: import com.droidplanner.R;
 4: import com.droidplanner.checklist.CheckListItem;
 5: import com.droidplanner.drone.Drone;
 6:
 7: import android.view.LayoutInflater;
 8: import android.view.View;
 9: import android.view.ViewGroup;
10: import android.widget.CheckBox;
11: import android.widget.CompoundButton;
12: import android.widget.CompoundButton.OnCheckedChangeListener;
13: import android.widget.LinearLayout;
14: import android.widget.Switch;
15:
16: public class ListRow_Switch extends ListRow implements OnCheckedChangeListener {
17:
18:         public ListRow_Switch(Drone drone, LayoutInflater inflater,
19:                         CheckListItem checkListItem) {
20:                 super(drone,inflater, checkListItem);
21:         }
22:
23:         public View getView(View convertView) {
24:                 View view;
25:                 if (convertView == null) {
26:                         ViewGroup viewGroup = (ViewGroup) inflater.inflate(
27:                                         R.layout.list_switch_item, null);
28:                         holder = new ViewHolder(viewGroup, checkListItem);
29:
30:                         viewGroup.setTag(holder);
31:                         view = viewGroup;
32:                 } else {
33:                         view = convertView;
34:                         holder = (ViewHolder) convertView.getTag();
35:                 }
36:
37:                 // TODO - Add spinner items
38:                 updateDisplay(view, (ViewHolder)holder, checkListItem);
39:                 return view;
40:         }
41:
42:         private void updateDisplay(View view, ViewHolder holder,
43:                         CheckListItem mListItem) {
44:                 boolean failMandatory = false;
45:
46:                 getDroneVariable(this.drone, mListItem);
47:                 failMandatory = !checkListItem.isSys_activated();
48:
49:                 holder.switchView.setOnCheckedChangeListener(this);
50:                 holder.switchView.setClickable(checkListItem.isEditable());
51:                 holder.switchView.setChecked(mListItem.isSys_activated());
52:
53:                 updateCheckBox(checkListItem.isMandatory() && !failMandatory);
54:         }
55:
56:         public int getViewType() {
57:                 return ListRow_Type.SWITCH_ROW.ordinal();
58:         }
59:
60:         private static class ViewHolder extends BaseViewHolder {
61:                 private Switch switchView;
62:
63:                 private ViewHolder(ViewGroup viewGroup, CheckListItem checkListIte
m) {
64:                         super(viewGroup, checkListItem);
65:                 }
66:
67:                 @Override
68:                 protected void setupViewItems(ViewGroup viewGroup,
69:                                 CheckListItem checkListItem) {
70:                         this.layoutView = (LinearLayout) viewGroup
71:                                         .findViewById(R.id.lst_layout);
72:                         this.switchView = (Switch) viewGroup.findViewById(R.id.lst
_switch);
73:                         this.checkBox = (CheckBox) viewGroup.findViewById(R.id.lst
_check);
74:                 }
75:         }
76:
77:         @Override
78:         public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
79:                 this.checkListItem.setSys_activated(arg1);
80:                 updateRowChanged((View)arg0,this.checkListItem);
81:
82:         }
83: }
```

```
    1: package com.droidplanner.checklist.row;
    2:
    3: import com.droidplanner.R;
    4: import com.droidplanner.checklist.CheckListItem;
    5: import com.droidplanner.drone.Drone;
    6:
    7: import android.view.LayoutInflater;
    8: import android.view.View;
    9: import android.view.ViewGroup;
   10: import android.widget.CheckBox;
   11: import android.widget.CompoundButton;
   12: import android.widget.LinearLayout;
   13: import android.widget.ToggleButton;
   14: import android.widget.CompoundButton.OnCheckedChangeListener;
   15:
   16: public class ListRow_Toggle extends ListRow implements OnCheckedChangeListener{
   17:
   18:         public ListRow_Toggle(Drone drone, LayoutInflater inflater, CheckListItem
checkListItem) {
   19:                 super(inflater,checkListItem);
   20:                 getDroneVariable(drone, checkListItem);
   21:         }
   22:
   23:         public View getView(View convertView) {
   24:                 View view;
   25:                 if (convertView == null) {
   26:                         ViewGroup viewGroup = (ViewGroup) inflater.inflate(
   27:                                         R.layout.list_toggle_item, null);
   28:                         holder = new ViewHolder(viewGroup, checkListItem);
   29:
   30:                         viewGroup.setTag(holder);
   31:                         view = viewGroup;
   32:                 } else {
   33:                         view = convertView;
   34:                         holder = (ViewHolder) convertView.getTag();
   35:                 }
   36:
   37:                 updateDisplay(view, (ViewHolder)holder, checkListItem);
   38:                 return view;
   39:         }
   40:
   41:         private void updateDisplay(View view, ViewHolder holder,
   42:                         CheckListItem mListItem) {
   43:                 boolean failMandatory = !checkListItem.isSys_activated();
   44:
   45:                 holder.toggleButton.setOnCheckedChangeListener(this);
   46:                 holder.toggleButton.setChecked(checkListItem.isSys_activated());
   47:                 holder.toggleButton.setClickable(checkListItem.isEditable());
   48:
   49:                 updateCheckBox(checkListItem.isMandatory() && !failMandatory);
   50:         }
   51:
   52:         public int getViewType() {
   53:                 return ListRow_Type.TOGGLE_ROW.ordinal();
   54:         }
   55:
   56:         private static class ViewHolder extends BaseViewHolder{
   57:                 private ToggleButton toggleButton;
   58:
   59:                 private ViewHolder(ViewGroup viewGroup, CheckListItem checkListIte
m) {
   60:                                 super(viewGroup, checkListItem);
   61:                 }
   62:
   63:                 @Override
   64:                 protected void setupViewItems(ViewGroup viewGroup, CheckListItem c
heckListItem)
   65:                 {
   66:                         this.layoutView = (LinearLayout) viewGroup
   67:                                         .findViewById(R.id.lst_layout);
   68:                         this.toggleButton = (ToggleButton) viewGroup
   69:                                         .findViewById(R.id.lst_toggle);
   70:                         this.checkBox = (CheckBox) viewGroup.findViewById(R.id.lst
_check);
   71:                 }
   72:         }
   73:
   74:         @Override
   75:         public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
 {
   76:                 this.checkListItem.setSys_activated(isChecked);
   77:                 updateRowChanged((View)(buttonView),this.checkListItem);
   78:         }
   79: }
```

```
 1: package com.droidplanner.checklist.row;
 2:
 3: public enum ListRow_Type {
 4:     CHECKBOX_ROW,
 5:     VALUE_ROW,
 6:     TOGGLE_ROW,
 7:     SWITCH_ROW,
 8:     RADIO_ROW,
 9:     SELECT_ROW,
10:     LEVEL_ROW,
11:     NOTE_ROW
12: }
```

```java
package com.droidplanner.checklist.row;

import com.droidplanner.R;
import com.droidplanner.checklist.CheckListItem;

import android.text.InputType;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnFocusChangeListener;
import android.view.ViewGroup;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.LinearLayout;

public class ListRow_Value extends ListRow implements OnFocusChangeListener {
	@SuppressWarnings("unused")
	private EditText editText;
	private boolean lastFocusState;
	private float lastValue;

	public ListRow_Value(LayoutInflater inflater,
			final CheckListItem checkListItem) {
		super(inflater, checkListItem);
	}

	public View getView(View convertView) {
		View view;

		if (convertView == null) {
			ViewGroup viewGroup = (ViewGroup) inflater.inflate(
					R.layout.list_value_item, null);
			holder = new ViewHolder(viewGroup, checkListItem);
			viewGroup.setTag(holder);
			view = viewGroup;

			lastValue = checkListItem.getFloatValue();

		} else {
			view = convertView;
			holder = (ViewHolder) convertView.getTag();

		}

		updateDisplay(view, (ViewHolder)holder, checkListItem);
		return view;
	}

	@SuppressWarnings("unused")
	private void updateDisplay(View view, ViewHolder holder,
			CheckListItem mListItem) {
		double minVal = mListItem.getMin_val();
		double nomVal = mListItem.getNom_val();
		double sysValue = mListItem.getSys_value();
		String unit = mListItem.getUnit();
		boolean failMandatory = sysValue <= minVal;

		editText = holder.editTextView;
		if (holder.editTextView.getText().toString() == null)
			holder.editTextView.setText("0.0");
		holder.editTextView.setOnFocusChangeListener(this);
		holder.editTextView.setText(String.valueOf(checkListItem
				.getFloatValue()));

		updateCheckBox(checkListItem.isMandatory() && !failMandatory);
	}

	public int getViewType() {
		return ListRow_Type.VALUE_ROW.ordinal();
	}

	private static class ViewHolder extends BaseViewHolder {
		private EditText editTextView;

		private ViewHolder(ViewGroup viewGroup, CheckListItem checkListItem) {
			super(viewGroup, checkListItem);
		}

		@Override
		protected void setupViewItems(ViewGroup viewGroup,
				CheckListItem checkListItem) {
			this.layoutView = (LinearLayout) viewGroup
					.findViewById(R.id.lst_layout);
			this.editTextView = (EditText) viewGroup
					.findViewById(R.id.lst_editText);
			this.editTextView.setInputType(InputType.TYPE_NUMBER_FLAG_DECIMAL
					| InputType.TYPE_NUMBER_FLAG_SIGNED
					| InputType.TYPE_CLASS_NUMBER);
			// this.editTextView.setInputType(InputType.TYPE_CLASS_PHONE);

			this.checkBox = (CheckBox) viewGroup.findViewById(R.id.lst_check);
		}
	}

	@Override
	public void onFocusChange(View v, boolean hasFocus) {
		if (lastFocusState != hasFocus) {
			lastFocusState = hasFocus;
			float a = (float) 0.0;

			try {
				a = Float.parseFloat(((EditText) v).getText().toString());
			} catch (NumberFormatException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

			if (a != lastValue) {
				lastValue = a;
				this.checkListItem
						.setValue(((EditText) v).getText().toString());
			}

			if (listener != null)
				listener.onRowItemChanged(v, this.checkListItem,
						this.checkListItem.isVerified());

		}
	}
}
```

```
 1: package com.droidplanner.checklist.xml;
 2:
 3: public class ListXmlData implements ListXmlData_Interface {
 4:         private String tagName;
 5:         private int depth;
 6:
 7:         public ListXmlData(String mTagName){
 8:                 this.tagName = mTagName;
 9:         }
10:
11:         public void setDepth(int mDepth){
12:                 this.depth = mDepth;
13:         }
14:
15:         @Override
16:         public String getTagName() {
17:                 return tagName;
18:         }
19:
20:         @Override
21:         public int getDepth() {
22:                 // TODO Auto-generated method stub
23:                 return depth;
24:         }
25: }
```

```
1: package com.droidplanner.checklist.xml;
2:
3: public interface ListXmlData_Interface {
4:         public String getTagName();
5:         public int getDepth();
6:
7: }
```

```
    1: package com.droidplanner.checklist.xml;
    2:
    3: import java.io.File;
    4: import java.io.FileInputStream;
    5: import java.io.FileNotFoundException;
    6: import java.io.IOException;
    7: import java.io.InputStreamReader;
    8: import java.io.StringReader;
    9: import org.xmlpull.v1.XmlPullParser;
   10: import org.xmlpull.v1.XmlPullParserException;
   11: import org.xmlpull.v1.XmlPullParserFactory;
   12:
   13: import android.content.Context;
   14: import android.content.res.XmlResourceParser;
   15: import android.os.Environment;
   16:
   17: public abstract class ListXmlParser {
   18:
   19:         public interface OnXmlParserError {
   20:                 public void onError(XmlPullParser parser);
   21:         }
   22:
   23:         protected OnXmlParserError errorListener;
   24:         protected XmlPullParser _xpp;
   25:
   26:
   27:         public void setOnXMLParserError(OnXmlParserError listener) {
   28:                 this.errorListener = listener;
   29:         }
   30:
   31:         public ListXmlParser() {
   32:         }
   33:
   34:         public ListXmlParser(Context context, int resourceId) {
   35:                 getListItemsFromResource(context, resourceId);
   36:         }
   37:
   38:         public ListXmlParser(String ioFile) throws FileNotFoundException, XmlPullP
arserException {
   39:                 getListItemsFromFile(ioFile);
   40:         }
   41:
   42:         public void getListItemsFromFile(String ioFile) throws FileNotFoundExcepti
on, XmlPullParserException {
   43:                 ioFile = Environment.getExternalStorageDirectory()+"/DroidPlanner/
Checklists/"+ioFile;
   44:                 File file = new File(ioFile);
   45:                 FileInputStream fis = new FileInputStream(file);
   46:                 XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
   47:                 factory.setNamespaceAware(true);
   48:                 _xpp = factory.newPullParser();
   49:                 _xpp.setInput(new InputStreamReader(fis));
   50:                 do_parse(_xpp);
   51:                 }
   52:
   53:         public void getListItemsFromResource(Context context, int resourceId) {
   54:                 XmlResourceParser is = context.getResources().getXml(resourceId);
   55:                 parse(is);
   56:         }
   57:
   58:         public void parse(String xmlStr) throws XmlPullParserException, IOExceptio
n {
   59:
   60:                 XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
   61:                 factory.setNamespaceAware(true);
   62:                 _xpp = factory.newPullParser();
   63:                 _xpp.setInput(new StringReader(xmlStr));
   64:                 do_parse(_xpp);
   65:         }
   66:
   67:         public void parse(XmlResourceParser xmlpp) {
   68:                 _xpp = xmlpp;
   69:                 do_parse(_xpp);
   70:         }
   71:
   72:         public void next() {
   73:                 if (_xpp != null)
   74:                         try {
   75:                                 _xpp.next();
   76:                         } catch (XmlPullParserException e) {
   77:                                 if (errorListener != null)
   78:                                         errorListener.onError(_xpp);
   79:
   80:                                 e.printStackTrace();
   81:                         } catch (IOException e) {
   82:                                 if (errorListener != null)
   83:                                         errorListener.onError(_xpp);
   84:
   85:                                 e.printStackTrace();
   86:                         }
   87:         }
   88:
   89:         public int getDepth() {
   90:                 if (_xpp != null)
   91:                         return _xpp.getDepth();
   92:                 return -1;
   93:         }
   94:
   95:         private void do_parse(XmlPullParser _xpp) {
   96:
   97:                 int eventType = 0;
   98:                 try {
   99:                         eventType = _xpp.getEventType();
  100:                 } catch (XmlPullParserException e) {
  101:                         if (errorListener != null)
  102:                                 errorListener.onError(_xpp);
  103:                         e.printStackTrace();
  104:                 }
  105:
  106:                 while (eventType != XmlPullParser.END_DOCUMENT) {
  107:                         if (eventType == XmlPullParser.START_DOCUMENT) {
  108:                                 process_StartDocument(_xpp);
  109:                         } else if (eventType == XmlPullParser.END_DOCUMENT) {
  110:                                 process_EndDocument(_xpp);
  111:                         } else if (eventType == XmlPullParser.START_TAG) {
  112:                                 process_StartTag(_xpp);
  113:                         } else if (eventType == XmlPullParser.END_TAG) {
  114:                                 process_EndTag(_xpp);
  115:                         } else if (eventType == XmlPullParser.TEXT) {
  116:                                 process_Text(_xpp);
  117:                         }
  118:                         try {
  119:                                 eventType = _xpp.next();
  120:                         } catch (XmlPullParserException e) {
  121:                                 if (errorListener != null)
  122:                                         errorListener.onError(_xpp);
  123:                                 e.printStackTrace();
  124:                         } catch (IOException e) {
  125:                                 if (errorListener != null)
  126:                                         errorListener.onError(_xpp);
  127:                                 e.printStackTrace();
  128:                         }
  129:                 }
  130:         }
```

```
131:
132:         public abstract void process_StartDocument(XmlPullParser xpp);
133:
134:         public abstract void process_EndDocument(XmlPullParser xpp);
135:
136:         public abstract void process_StartTag(XmlPullParser xpp);
137:
138:         public abstract void process_EndTag(XmlPullParser xpp);
139:
140:         public abstract void process_Text(XmlPullParser xpp);
141: }
```

```
    1: package com.droidplanner.connection;
    2:
    3: import java.io.IOException;
    4: import java.io.InputStream;
    5: import java.io.OutputStream;
    6: import java.net.UnknownHostException;
    7: import java.util.Set;
    8: import java.util.UUID;
    9:
   10: import android.annotation.SuppressLint;
   11: import android.bluetooth.BluetoothAdapter;
   12: import android.bluetooth.BluetoothDevice;
   13: import android.bluetooth.BluetoothSocket;
   14: import android.content.Context;
   15: import android.content.SharedPreferences;
   16: import android.os.ParcelUuid;
   17: import android.util.Log;
   18:
   19: public class BluetoothConnection extends MAVLinkConnection {
   20:         private static final String BLUE = "BLUETOOTH";
   21:         private static final String UUID_SPP_DEVICE = "00001101-0000-1000-8000-008
05F9B34FB";
   22:         private BluetoothAdapter mBluetoothAdapter;
   23:         private OutputStream out;
   24:         private InputStream in;
   25:         private BluetoothSocket bluetoothSocket;
   26:
   27:         public BluetoothConnection(Context parentContext) {
   28:                 super(parentContext);
   29:                 mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
   30:                 if (mBluetoothAdapter == null) {
   31:                         Log.d(BLUE, "Null adapters");
   32:                 }
   33:         }
   34:
   35:         @Override
   36:         protected void openConnection() throws UnknownHostException, IOException {
   37:                 Log.d(BLUE, "Conenct");
   38:                 BluetoothDevice device = findBluetoothDevice();
   39:
   40:                 bluetoothSocket = device.createRfcommSocketToServiceRecord(UUID
   41:                                 .fromString(UUID_SPP_DEVICE)); // TODO May need wo
rk
   42:                 mBluetoothAdapter.cancelDiscovery();
   43:                 bluetoothSocket.connect();
   44:
   45:                 out = bluetoothSocket.getOutputStream();
   46:                 in = bluetoothSocket.getInputStream();
   47:         }
   48:
   49:         @SuppressLint("NewApi")
   50:         private BluetoothDevice findBluetoothDevice() throws UnknownHostException
{
   51:                 Set<BluetoothDevice> pairedDevices = mBluetoothAdapter
   52:                                 .getBondedDevices();
   53:                 // If there are paired devices
   54:                 if (pairedDevices.size() > 0) {
   55:                         // Loop through paired devices
   56:                         for (BluetoothDevice device : pairedDevices) {
   57:                                 // Add the name and address to an array adapter to
 show in a
   58:                                 // ListView
   59:                                 Log.d(BLUE, device.getName() + " #" + device.getAd
dress() + "#");
   60:                                 for (ParcelUuid id : device.getUuids()) {
   61:                                         // TODO maybe this will not work on newer
devices
   62:                                         Log.d(BLUE, "id:" + id.toString());
   63:                                         if (id.toString().equalsIgnoreCase(UUID_SP
P_DEVICE)) {
   64:                                                 return device;
   65:                                         }
   66:                                 }
   67:                         }
   68:                 }
   69:                 throw new UnknownHostException("No Bluetooth Device found");
   70:         }
   71:
   72:         @Override
   73:         protected void readDataBlock() throws IOException {
   74:                 iavailable = in.read(readData);
   75:
   76:         }
   77:
   78:         @Override
   79:         protected void sendBuffer(byte[] buffer) throws IOException {
   80:                 if (out != null) {
   81:                         out.write(buffer);
   82:                 }
   83:         }
   84:
   85:         @Override
   86:         protected void closeConnection() throws IOException {
   87:                 bluetoothSocket.close();
   88:         }
   89:
   90:         @Override
   91:         protected void getPreferences(SharedPreferences prefs) {
   92:                 // TODO Auto-generated method stub
   93:         }
   94:
   95:         /*
   96:          * private getUUID(device: BluetoothDevice) = { val uuids =
   97:          * Option(device.getUuids).getOrElse(Array()).map(_.getUuid) uuids.find {
   98:          * uuid => uuid.toString.startsWith(serialUUIDprefix) } }
   99:          */
  100:
  101: }
```

```
 1: package com.droidplanner.connection;
 2:
 3: import java.io.BufferedOutputStream;
 4: import java.io.FileNotFoundException;
 5: import java.io.IOException;
 6: import java.net.UnknownHostException;
 7: import java.nio.ByteBuffer;
 8: import java.nio.ByteOrder;
 9:
10: import android.content.Context;
11: import android.content.SharedPreferences;
12: import android.preference.PreferenceManager;
13:
14: import com.MAVLink.Parser;
15: import com.MAVLink.Messages.MAVLinkMessage;
16: import com.MAVLink.Messages.MAVLinkPacket;
17: import com.droidplanner.file.FileStream;
18:
19: public abstract class MAVLinkConnection extends Thread {
20:
21:         protected abstract void openConnection() throws UnknownHostException,
22:                         IOException;
23:
24:         protected abstract void readDataBlock() throws IOException;
25:
26:         protected abstract void sendBuffer(byte[] buffer) throws IOException;
27:
28:         protected abstract void closeConnection() throws IOException;
29:
30:         protected abstract void getPreferences(SharedPreferences prefs);
31:
32:         public interface MavLinkConnectionListner {
33:                 public void onReceiveMessage(MAVLinkMessage msg);
34:
35:                 public void onDisconnect();
36:         }
37:
38:         protected Context parentContext;
39:         private MavLinkConnectionListner listner;
40:         private boolean logEnabled;
41:         private BufferedOutputStream logWriter;
42:
43:         protected MAVLinkPacket receivedPacket;
44:         protected Parser parser = new Parser();
45:         protected byte[] readData = new byte[4096];
46:         protected int iavailable, i;
47:         protected boolean connected = true;
48:
49:         private ByteBuffer logBuffer;
50:
51:         public MAVLinkConnection(Context parentContext) {
52:                 this.parentContext = parentContext;
53:                 this.listner = (MavLinkConnectionListner) parentContext;
54:
55:                 SharedPreferences prefs = PreferenceManager
56:                                 .getDefaultSharedPreferences(parentContext);
57:                 logEnabled = prefs.getBoolean("pref_mavlink_log_enabled", false);
58:                 getPreferences(prefs);
59:         }
60:
61:         @Override
62:         public void run() {
63:                 super.run();
64:                 try {
65:                         parser.stats.mavlinkResetStats();
66:                         openConnection();
67:                         if (logEnabled) {
68:                                 logWriter = FileStream.getTLogFileStream();
69:                                 logBuffer = ByteBuffer.allocate(Long.SIZE / Byte.S
IZE);
70:                                 logBuffer.order(ByteOrder.BIG_ENDIAN);
71:                         }
72:
73:                         while (connected) {
74:                                 readDataBlock();
75:                                 handleData();
76:                         }
77:                         closeConnection();
78:                 } catch (FileNotFoundException e) {
79:                         e.printStackTrace();
80:                 } catch (IOException e) {
81:                         e.printStackTrace();
82:                 }
83:                 listner.onDisconnect();
84:         }
85:
86:         private void handleData() throws IOException {
87:                 if (iavailable < 1) {
88:                         return;
89:                 }
90:                 for (i = 0; i < iavailable; i++) {
91:                         receivedPacket = parser.mavlink_parse_char(readData[i] & 0
x00ff);
92:                         if (receivedPacket != null) {
93:                                 saveToLog(receivedPacket);
94:                                 MAVLinkMessage msg = receivedPacket.unpack();
95:                                 listner.onReceiveMessage(msg);
96:                         }
97:                 }
98:
99:         }
100:
101:        private void saveToLog(MAVLinkPacket receivedPacket) throws IOException {
102:                if (logEnabled) {
103:                        try {
104:                                logBuffer.clear();
105:                                long time = System.currentTimeMillis() * 1000;
106:                                logBuffer.putLong(time);
107:                                logWriter.write(logBuffer.array());
108:                                logWriter.write(receivedPacket.encodePacket());
109:                        } catch (Exception e) {
110:                                // There was a null pointer error for some users o
n
111:                                // logBuffer.clear();
112:                        }
113:                }
114:        }
115:
116:        /**
117:         * Format and send a Mavlink packet via the MAVlink stream
118:         *
119:         * @param packet
120:         *            MavLink packet to be transmitted
121:         */
122:        public void sendMavPacket(MAVLinkPacket packet) {
123:                byte[] buffer = packet.encodePacket();
124:                try {
125:                        sendBuffer(buffer);
126:                        saveToLog(packet);
127:                } catch (IOException e) {
128:                        e.printStackTrace();
129:                }
130:        }
131:
```

```
132:         public void disconnect() {
133:                 connected = false;
134:         }
135:
136: }
```

```
    1: package com.droidplanner.connection;
    2:
    3: import java.io.BufferedInputStream;
    4: import java.io.BufferedOutputStream;
    5: import java.io.IOException;
    6: import java.net.InetAddress;
    7: import java.net.Socket;
    8: import java.net.UnknownHostException;
    9:
   10: import android.content.Context;
   11: import android.content.SharedPreferences;
   12:
   13: public class TcpConnection extends MAVLinkConnection {
   14:         private Socket socket;
   15:         private BufferedOutputStream mavOut;
   16:         private BufferedInputStream mavIn;
   17:
   18:         private String serverIP;
   19:         private int serverPort;
   20:
   21:         public TcpConnection(Context context) {
   22:                 super(context);
   23:         }
   24:
   25:         @Override
   26:         protected void openConnection() throws UnknownHostException, IOException {
   27:                 getTCPStream();
   28:         }
   29:
   30:         @Override
   31:         protected void readDataBlock() throws IOException {
   32:                 iavailable = mavIn.read(readData);
   33:         }
   34:
   35:         @Override
   36:         protected void sendBuffer(byte[] buffer) throws IOException {
   37:                 if (mavOut != null) {
   38:                         mavOut.write(buffer);
   39:                         mavOut.flush();
   40:                 }
   41:         }
   42:
   43:         @Override
   44:         protected void closeConnection() throws IOException {
   45:                 socket.close();
   46:         }
   47:
   48:         @Override
   49:         protected void getPreferences(SharedPreferences prefs) {
   50:                 serverIP = prefs.getString("pref_server_ip", "");
   51:                 serverPort = Integer.parseInt(prefs.getString("pref_server_port",
"0"));
   52:
   53:         }
   54:
   55:         private void getTCPStream() throws UnknownHostException, IOException {
   56:                 InetAddress serverAddr = InetAddress.getByName(serverIP);
   57:                 socket = new Socket(serverAddr, serverPort);
   58:                 mavOut = new BufferedOutputStream((socket.getOutputStream()));
   59:                 mavIn = new BufferedInputStream(socket.getInputStream());
   60:         }
   61:
   62: }
```

```
    1: package com.droidplanner.connection;
    2:
    3: import java.io.IOException;
    4: import java.net.DatagramPacket;
    5: import java.net.DatagramSocket;
    6: import java.net.InetAddress;
    7: import java.net.UnknownHostException;
    8:
    9: import android.content.Context;
   10: import android.content.SharedPreferences;
   11: import android.os.AsyncTask;
   12:
   13: public class UdpConnection extends MAVLinkConnection {
   14:
   15:         private DatagramSocket socket;
   16:         private int serverPort;
   17:
   18:         private int hostPort;
   19:         private InetAddress hostAdd;
   20:
   21:         public UdpConnection(Context context) {
   22:                 super(context);
   23:         }
   24:
   25:         @Override
   26:         protected void openConnection() throws UnknownHostException, IOException {
   27:                 getUdpStream();
   28:         }
   29:
   30:         @Override
   31:         protected void readDataBlock() throws IOException {
   32:                 DatagramPacket packet = new DatagramPacket(readData, readData.leng
th);
   33:                 socket.receive(packet);
   34:                 hostAdd=packet.getAddress();
   35:                 hostPort = packet.getPort();
   36:                 iavailable = packet.getLength();
   37:         }
   38:
   39:         @Override
   40:         protected void sendBuffer(byte[] buffer) throws IOException {
   41:                 new UdpSender().execute(buffer);
   42:         }
   43:
   44:         private class UdpSender extends AsyncTask<byte[], Integer, Integer> {
   45:
   46:                 @Override
   47:                 protected Integer doInBackground(byte[]... params) {
   48:                         try{
   49:                                 byte[] buffer = params[0];
   50:                                 DatagramPacket packet = new DatagramPacket(buffer,
 buffer.length,
   51:                                                 hostAdd, hostPort);
   52:                                 socket.send(packet);
   53:
   54:                         }catch (Exception e){
   55:                                 e.printStackTrace();
   56:                         }
   57:                         return null;
   58:                 }
   59:         }
   60:
   61:
   62:         @Override
   63:         protected void closeConnection() throws IOException {
   64:                 socket.close();
   65:         }
   66:
   67:         @Override
   68:         protected void getPreferences(SharedPreferences prefs) {
   69:                 serverPort = Integer.parseInt(prefs.getString("pref_udp_server_por
t",
   70:                 "14550"));
   71:         }
   72:
   73:         private void getUdpStream() throws UnknownHostException, IOException {
   74:                 socket = new DatagramSocket(serverPort);
   75:                 socket.setBroadcast(true);
   76:                 socket.setReuseAddress(true);
   77:         }
   78:
   79: }
```

```
    1: package com.droidplanner.connection;
    2:
    3: import java.io.IOException;
    4: import java.net.UnknownHostException;
    5:
    6: //This version is modified by Helibot to use the "USB Serial for Android Library"
    7: //See https://code.google.com/p/usb-serial-for-android/
    8: // It should allow support of FDTI and other Serial to USB converters.
    9: // It should allow APM v2.0 and 2.5 to connect via USB cable straight to APM.
   10: // Be sure to set the Telementry speed in the setting menu to
   11: //    115200 when connecting directly with USB cable.
   12: import com.hoho.android.usbserial.driver.UsbSerialDriver;
   13: import com.hoho.android.usbserial.driver.UsbSerialProber;
   14: import android.hardware.usb.UsbManager;
   15:
   16: import android.content.Context;
   17: import android.content.SharedPreferences;
   18: import android.util.Log;
   19:
   20: public class UsbConnection extends MAVLinkConnection {
   21:         private static int baud_rate = 57600;
   22:     private static UsbSerialDriver sDriver = null;
   23:
   24:         public UsbConnection(Context parentContext) {
   25:                 super(parentContext);
   26:         }
   27:
   28:         @Override
   29:         protected void openConnection() throws UnknownHostException, IOException {
   30:                 openCOM();
   31:         }
   32:
   33:         @Override
   34:         protected void readDataBlock() throws IOException {
   35:                 //Read data from driver. This call will return upto readData.lengt
h bytes.
   36:                 //If no data is received it will timeout after 200ms (as set by pa
rameter 2)
   37:                 iavailable = sDriver.read(readData,200);
   38:                 if (iavailable == 0) iavailable = -1;
   39:                 //Log.d("USB", "Bytes read" + iavailable);
   40:         }
   41:
   42:         @Override
   43:         protected void sendBuffer(byte[] buffer) {
   44:                 //Write data to driver. This call should write buffer.length bytes

   45:                 //if data cant be sent , then it will timeout in 500ms (as set by
parameter 2)
   46:                 if (connected && sDriver != null) {
   47:                         try{
   48:                                 sDriver.write(buffer,500);
   49:                         } catch (IOException e) {
   50:                                 Log.e("USB", "Error Sending: " + e.getMessage(), e
);
   51:                         }
   52:                 }
   53:         }
   54:
   55:         @Override
   56:         protected void closeConnection() throws IOException {
   57:       if (sDriver != null) {
   58:          try {
   59:               sDriver.close();
   60:          } catch (IOException e) {
   61:              // Ignore.
   62:          }
   63:           sDriver = null;
   64:       }
   65:         }
   66:
   67:         @Override
   68:         protected void getPreferences(SharedPreferences prefs) {
   69:                 String baud_type = prefs.getString("pref_baud_type", "57600");
   70:                 if (baud_type.equals("57600"))
   71:                         baud_rate = 57600;
   72:                 else
   73:                         baud_rate = 115200;
   74:         }
   75:
   76:         private void openCOM() throws IOException {
   77:                 // Get UsbManager from Android.
   78:                 UsbManager manager = (UsbManager) parentContext.getSystemService(C
ontext.USB_SERVICE);
   79:
   80:                 // Find the first available driver.
   81:                 //**TODO: We should probably step through all available USB Device
s
   82:                 //...but its unlikely to happen on a Phone/tablet running DroidPla
nner.
   83:                 sDriver = UsbSerialProber.findFirstDevice(manager);
   84:
   85:                 if (sDriver == null) {
   86:                         Log.d("USB", "No Devices found");
   87:                         throw new IOException();
   88:                 }
   89:                 else
   90:                 {
   91:                 Log.d("USB", "Opening using Baud rate " + baud_rate);
   92:                 try {
   93:                         sDriver.open();
   94:                         sDriver.setParameters(baud_rate, 8, UsbSerialDriver.STOPBI
TS_1, UsbSerialDriver.PARITY_NONE);
   95:                     } catch (IOException e) {
   96:                         Log.e("USB", "Error setting up device: " + e.getMessage(),
 e);
   97:                         try {
   98:                             sDriver.close();
   99:                         } catch (IOException e2) {
  100:                             // Ignore.
  101:                         }
  102:                         sDriver = null;
  103:                         return;
  104:                 }
  105:                 }
  106:         }
  107: }
```

```
    1: package com.droidplanner.dialogs;
    2:
    3: import com.droidplanner.helpers.units.Altitude;
    4:
    5: import android.app.AlertDialog;
    6: import android.app.Dialog;
    7: import android.content.Context;
    8: import android.content.DialogInterface;
    9: import android.view.View;
   10: import android.widget.LinearLayout;
   11: import android.widget.LinearLayout.LayoutParams;
   12: import android.widget.NumberPicker;
   13:
   14: public class AltitudeDialog implements DialogInterface.OnClickListener {
   15:         private NumberPicker thousandPicker;
   16:         private NumberPicker hundredPicker;
   17:         private NumberPicker decadePicker;
   18:         private NumberPicker unitPicker;
   19:         private OnAltitudeChangedListner listner;
   20:
   21:         public interface OnAltitudeChangedListner {
   22:                 public void onAltitudeChanged(Altitude newAltitude);
   23:         }
   24:
   25:         public AltitudeDialog(OnAltitudeChangedListner listner) {
   26:                 this.listner = listner;
   27:         }
   28:
   29:         public void build(Altitude altitude, Context context) {
   30:                 AlertDialog dialog = buildDialog(context);
   31:                 setValue(altitude.valueInMeters());
   32:                 dialog.show();
   33:         }
   34:
   35:         private AlertDialog buildDialog(Context context) {
   36:                 AlertDialog.Builder builder = new AlertDialog.Builder(context);
   37:                 builder.setTitle("Altitude");
   38:                 builder.setView(buildAltitudePicker(context));
   39:                 builder.setNegativeButton("Cancel", this).setPositiveButton("Ok",
this);
   40:                 AlertDialog dialog = builder.create();
   41:                 return dialog;
   42:         }
   43:
   44:         private View buildAltitudePicker(Context context) {
   45:                 LayoutParams layoutStyle = new LayoutParams(LayoutParams.MATCH_PAR
ENT,
   46:                                 LayoutParams.WRAP_CONTENT);
   47:                 layoutStyle.weight = 1;
   48:                 LinearLayout layout = new LinearLayout(context);
   49:                 layout.setLayoutParams(layoutStyle);
   50:
   51:                 unitPicker = buildDigitPicker(context, layoutStyle);
   52:                 decadePicker = buildDigitPicker(context, layoutStyle);
   53:                 hundredPicker = buildDigitPicker(context, layoutStyle);
   54:                 thousandPicker = buildDigitPicker(context, layoutStyle);
   55:
   56:                 layout.addView(thousandPicker);
   57:                 layout.addView(hundredPicker);
   58:                 layout.addView(decadePicker);
   59:                 layout.addView(unitPicker);
   60:                 return layout;
   61:         }
   62:
   63:         private NumberPicker buildDigitPicker(Context context,
   64:                         LayoutParams layoutStyle) {
   65:                 NumberPicker digitPicker = new NumberPicker(context);
   66:                 digitPicker.setMaxValue(9);
   67:                 digitPicker.setMinValue(0);
   68:                 digitPicker
   69:                                 .setDescendantFocusability(NumberPicker.FOCUS_BLOC
K_DESCENDANTS);
   70:                 digitPicker.setLayoutParams(layoutStyle);
   71:                 digitPicker.setWrapSelectorWheel(false);
   72:                 return digitPicker;
   73:         }
   74:
   75:         @Override
   76:         public void onClick(DialogInterface arg0, int which) {
   77:                 if (which == Dialog.BUTTON_POSITIVE) {
   78:                         listner.onAltitudeChanged(new Altitude(getValue()));
   79:                 }
   80:         }
   81:
   82:         private void setValue(double value) {
   83:                 thousandPicker.setValue((int) (value / 1000));
   84:                 value -= thousandPicker.getValue() * 1000;
   85:                 hundredPicker.setValue((int) (value / 100));
   86:                 value -= hundredPicker.getValue() * 100;
   87:                 decadePicker.setValue((int) (value / 10));
   88:                 value -= decadePicker.getValue() * 10;
   89:                 unitPicker.setValue((int) (value));
   90:         }
   91:
   92:         private double getValue() {
   93:                 return (thousandPicker.getValue() * 1000 + hundredPicker.getValue(
)
   94:                                 * 100 + decadePicker.getValue() * 10 + unitPicker.
getValue());
   95:         }
   96:
   97: }
```

```java
package com.droidplanner.dialogs.checklist;

import java.util.HashMap;
import java.util.List;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.view.LayoutInflater;
import android.view.View;
import android.view.WindowManager;
import java.util.ArrayList;

import org.xmlpull.v1.XmlPullParser;

import com.droidplanner.R;
import com.droidplanner.MAVLink.MavLinkArm;
import com.droidplanner.checklist.CheckListAdapter;
import com.droidplanner.checklist.CheckListAdapter.OnCheckListItemUpdateListener;
import com.droidplanner.checklist.CheckListItem;
import com.droidplanner.checklist.CheckListXmlParser;
import com.droidplanner.checklist.xml.ListXmlParser.OnXmlParserError;
import com.droidplanner.drone.Drone;
import android.widget.ExpandableListView;
import android.widget.Toast;

public class PreflightDialog implements DialogInterface.OnClickListener,
                OnXmlParserError, OnCheckListItemUpdateListener {

        private Context context;
        private View view;
        private Drone drone;
        private List<String> listDataHeader;
        private List<CheckListItem> checkItemList;
        private HashMap<String, List<CheckListItem>> listDataChild;
        private CheckListAdapter listAdapter;
        private ExpandableListView expListView;
        private AlertDialog dialog;

        public PreflightDialog() {
                // TODO Auto-generated constructor stub
        }

        // public void build(Drone mdrone, Context mcontext, boolean mpreflight) {
        public void build(Context mcontext, Drone mdrone, boolean mpreflight) {
                context = mcontext;
                drone = mdrone;
                // TODO Read System checklist here
                CheckListXmlParser xml = new CheckListXmlParser(mcontext,
                                R.xml.checklist_default);
//              CheckListXmlParser xml = new CheckListXmlParser("checklist_ext.xml");

                xml.setOnXMLParserError(this);
                listDataHeader = xml.getCategories();
                checkItemList = xml.getCheckListItems();

                dialog = buildDialog(mpreflight);
                dialog.show();
        }

        private AlertDialog buildDialog(boolean mpreflight) {
                AlertDialog.Builder builder = new AlertDialog.Builder(context);
                builder.setTitle("Pre-Flight Check");
                builder.setView(buildView());
                builder.setPositiveButton("Ok", this);
                if (mpreflight) {
                        builder.setNegativeButton("Cancel", this);
                }
                AlertDialog dialog = builder.create();
                return dialog;
        }

        protected View buildView() {
                LayoutInflater inflater = (LayoutInflater) context
                                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                view = inflater.inflate(R.layout.layout_checklist, null);
                // get the listview
                expListView = (ExpandableListView) view.findViewById(R.id.expListView);

                // preparing list data
                prepareListData();

                listAdapter = new CheckListAdapter(drone, inflater, listDataHeader,
                                listDataChild);
                listAdapter.setHeaderLayout(R.layout.list_group_header);
                listAdapter.setOnCheckListItemUpdateListener(this);
                // setting list adapter

                expListView.post(new Runnable() {

                        @Override
                        public void run() {
                                dialog.getWindow()
                                                .clearFlags(
                                                                WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE
                                                                | WindowManager.LayoutParams.FLAG_ALT_FOCUSABLE_IM);
                                dialog.getWindow().setSoftInputMode(
                                                WindowManager.LayoutParams.SOFT_INPUT_ADJUST_PAN);
                        }
                });
                expListView.setAdapter(listAdapter);

                expListView.expandGroup(0);
                expListView.expandGroup(1);

                return view;
        }

        private void prepareListData() {
                listDataChild = new HashMap<String, List<CheckListItem>>();
                List<CheckListItem> cli;

                for (int h = 0; h < listDataHeader.size(); h++) {
                        cli = new ArrayList<CheckListItem>();
                        for (int i = 0; i < checkItemList.size(); i++) {
                                CheckListItem c = checkItemList.get(i);
                                if (c.getCategoryIndex() == h)
                                        cli.add(c);
                        }
                        listDataChild.put(listDataHeader.get(h), cli);
                }
        }


        private void updateSystem(CheckListItem checkListItem) {
                if(checkListItem.getSys_tag().isEmpty())
                        return;
```

```
  128:                 if(checkListItem.getSys_tag().equalsIgnoreCase("SYS_CONNECTION_STA
TE")){
  129:                         drone.MavClient.toggleConnectionState();
  130:
  131:                 }else if(checkListItem.getSys_tag().equalsIgnoreCase("SYS_ARM_STAT
E")){
  132:                         if (drone.MavClient.isConnected()) {
  133:                                 if (!drone.state.isArmed())
  134:                                         drone.tts.speak("Arming the vehicle, pleas
e standby");
  135:                                 MavLinkArm.sendArmMessage(drone, !drone.state.isAr
med());
  136:                         }
  137:                 }
  138:         }
  139:         @Override
  140:         public void onClick(DialogInterface arg0, int arg1) {
  141:
  142:         }
  143:
  144:         @Override
  145:         public void onError(XmlPullParser parser) {
  146:                 // TODO Auto-generated method stub
  147:
  148:         }
  149:
  150:         @Override
  151:         public void onRadioGroupUpdate(CheckListItem checkListItem, int checkId) {
  152:                 Toast.makeText(
  153:                                 context,
  154:                                 checkListItem.getTitle() + " : "
  155:                                                 + checkListItem.getOptionLists().g
et(checkId),
  156:                                 Toast.LENGTH_SHORT).show();
  157:
  158:         }
  159:
  160:         @Override
  161:         public void onSelectUpdate(CheckListItem checkListItem, int selectId) {
  162:                 Toast.makeText(
  163:                                 context,
  164:                                 checkListItem.getTitle() + " : "
  165:                                                 + checkListItem.getOptionLists().g
et(selectId),
  166:                                 Toast.LENGTH_SHORT).show();
  167:
  168:         }
  169:
  170:         @Override
  171:         public void onCheckBoxUpdate(CheckListItem checkListItem, boolean isChecke
d) {
  172:                 Toast.makeText(
  173:                                 context,
  174:                                 checkListItem.getTitle() + " : " + checkListItem.g
etTitle()
  175:                                                 + (isChecked ? " checked" : " unch
ecked"),
  176:                                 Toast.LENGTH_SHORT).show();
  177:
  178:         }
  179:
  180:         @Override
  181:         public void onSwitchUpdate(CheckListItem checkListItem, boolean isSwitched
) {
  182:                 updateSystem(checkListItem);
  183:                 Toast.makeText(
  184:                                 context,
  185:                                 checkListItem.getTitle() + " : " + checkListItem.g
etTitle()
  186:                                                 + (isSwitched ? " switched ON" : "
 switched OFF"),
  187:                                 Toast.LENGTH_SHORT).show();
  188:
  189:         }
  190:
  191:         @Override
  192:         public void onToggleUpdate(CheckListItem checkListItem, boolean isToggled)
 {
  193:                 Toast.makeText(
  194:                                 context,
  195:                                 checkListItem.getTitle() + " : " + checkListItem.g
etTitle()
  196:                                                 + (isToggled ? " toggled ON" : " t
oggled OFF"),
  197:                                 Toast.LENGTH_SHORT).show();
  198:
  199:         }
  200:
  201:         @Override
  202:         public void onValueUpdate(CheckListItem checkListItem, String newValue) {
  203:                 // TODO Auto-generated method stub
  204:
  205:         }
  206: }
```

```
    1: package com.droidplanner.dialogs.openfile;
    2:
    3: import android.app.AlertDialog;
    4: import android.content.Context;
    5: import android.content.DialogInterface;
    6: import android.content.DialogInterface.OnClickListener;
    7: import android.widget.Toast;
    8:
    9: import com.droidplanner.R;
   10:
   11: public abstract class OpenFileDialog implements OnClickListener {
   12:
   13:         public interface FileReader {
   14:                 public String getPath();
   15:
   16:                 public String[] getFileList();
   17:
   18:                 public boolean openFile(String file);
   19:         }
   20:
   21:         protected abstract FileReader createReader();
   22:
   23:         protected abstract void onDataLoaded(FileReader reader);
   24:
   25:         private String[] itemList;
   26:         private Context context;
   27:         private FileReader reader;
   28:
   29:         public void openDialog(Context context) {
   30:                 this.context = context;
   31:                 reader = createReader();
   32:
   33:                 itemList = reader.getFileList();
   34:                 if (itemList.length == 0) {
   35:                         Toast.makeText(context, R.string.no_files, Toast.LENGTH_SH
ORT)
   36:                                         .show();
   37:                         return;
   38:                 }
   39:                 AlertDialog.Builder dialog = new AlertDialog.Builder(context);
   40:                 dialog.setTitle(R.string.select_file_to_open);
   41:                 dialog.setItems(itemList, this);
   42:                 dialog.create().show();
   43:         }
   44:
   45:         @Override
   46:         public void onClick(DialogInterface dialog, int which) {
   47:                 boolean isFileOpen = reader
   48:                                 .openFile(reader.getPath() + itemList[which]);
   49:
   50:                 if (isFileOpen) {
   51:                         Toast.makeText(context, itemList[which], Toast.LENGTH_LONG
).show();
   52:                 } else {
   53:                         Toast.makeText(context, R.string.error_when_opening_file,
   54:                                         Toast.LENGTH_SHORT).show();
   55:                 }
   56:
   57:                 onDataLoaded(reader);
   58:         }
   59:
   60: }
```

```
 1: package com.droidplanner.dialogs.openfile;
 2:
 3: import java.util.List;
 4:
 5: import com.droidplanner.file.IO.GcpReader;
 6: import com.droidplanner.gcp.Gcp;
 7:
 8: public abstract class OpenGcpFileDialog extends OpenFileDialog {
 9:         public abstract void onGcpFileLoaded(List<Gcp> gcpList);
10:
11:         @Override
12:         protected FileReader createReader() {
13:                 return new GcpReader();
14:         }
15:
16:         @Override
17:         protected void onDataLoaded(FileReader reader) {
18:                 onGcpFileLoaded(((GcpReader) reader).gcpList);
19:         }
20: }
```

```
 1: package com.droidplanner.dialogs.openfile;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.file.IO.MissionReader;
 5:
 6: public abstract class OpenMissionDialog extends OpenFileDialog {
 7:         public abstract void waypointFileLoaded(MissionReader reader);
 8:
 9:         Drone drone;
10:
11:         public OpenMissionDialog(Drone drone) {
12:                 super();
13:                 this.drone = drone;
14:         }
15:
16:         @Override
17:         protected FileReader createReader() {
18:                 return new MissionReader();
19:         }
20:
21:         @Override
22:         protected void onDataLoaded(FileReader reader) {
23:                 waypointFileLoaded((MissionReader) reader);
24:         }
25: }
```

```
 1: package com.droidplanner.dialogs.openfile;
 2:
 3: import java.util.List;
 4:
 5: import com.droidplanner.file.IO.ParameterReader;
 6: import com.droidplanner.parameters.Parameter;
 7:
 8: public abstract class OpenParameterDialog extends OpenFileDialog {
 9:         public abstract void parameterFileLoaded(List<Parameter> parameters);
10:
11:         @Override
12:         protected FileReader createReader() {
13:                 return new ParameterReader();
14:         }
15:
16:         @Override
17:         protected void onDataLoaded(FileReader reader) {
18:                 parameterFileLoaded(((ParameterReader) reader).getParameters());
19:         }
20: }
```

```
    1: package com.droidplanner.dialogs.parameters;
    2:
    3: import android.app.AlertDialog;
    4: import android.content.Context;
    5: import android.view.LayoutInflater;
    6: import android.view.View;
    7: import android.widget.TextView;
    8:
    9: import com.droidplanner.R;
   10: import com.droidplanner.parameters.ParameterMetadata;
   11:
   12:
   13: public class DialogParameterInfo {
   14:
   15:     public static AlertDialog.Builder build(ParameterMetadata metadata, Context co
ntext) {
   16:         return new AlertDialog.Builder(context)
   17:                 .setTitle(metadata.getName())
   18:                 .setView(buildView(metadata, context));
   19:     }
   20:
   21:     private static View buildView(ParameterMetadata metadata, Context context) {
   22:         final LayoutInflater inflater = LayoutInflater.from(context);
   23:         final View view = inflater.inflate(R.layout.dialog_parameters_info, null);
   24:
   25:         setTextView(view, R.id.nameView, metadata.getDisplayName());
   26:         setTextView(view, R.id.descView, metadata.getDescription());
   27:
   28:         setTextLayout(view, R.id.unitsLayout, R.id.unitsView, metadata.getUnits())
;
   29:         setTextLayout(view, R.id.rangeLayout, R.id.rangeView, formatRange(metadata
.getRange()));
   30:         setTextLayout(view, R.id.valuesLayout, R.id.valuesView, metadata.getValues
());
   31:
   32:         return view;
   33:     }
   34:
   35:     private static String formatRange(String range) {
   36:         if(range == null || range.isEmpty())
   37:             return null;
   38:
   39:         final String[] part = range.split(" ");
   40:         if(part.length == 2)
   41:             return part[0] + "  /  " + part[1];
   42:         else
   43:             return range;
   44:     }
   45:
   46:     private static void setTextView(View view, int ridTextView, String text) {
   47:         final TextView textView = (TextView) view.findViewById(ridTextView);
   48:         if(text != null) {
   49:             textView.setText(text);
   50:         } else {
   51:             textView.setVisibility(View.GONE);
   52:         }
   53:     }
   54:
   55:     private static void setTextLayout(View view, int ridLayout, int ridTextView, S
tring text) {
   56:         TextView textView;
   57:         if(text != null) {
   58:             textView = (TextView) view.findViewById(ridTextView);
   59:             textView.setText(text);
   60:         } else {
   61:             view.findViewById(ridLayout).setVisibility(View.GONE);
   62:         }
   63:     }
   64:
   65: }
```

```
    1: package com.droidplanner.dialogs.parameters;
    2:
    3: import java.text.ParseException;
    4: import java.util.Map;
    5:
    6: import android.app.AlertDialog;
    7: import android.content.Context;
    8: import android.content.DialogInterface;
    9:
   10: import com.droidplanner.parameters.Parameter;
   11: import com.droidplanner.parameters.ParameterMetadata;
   12:
   13:
   14: public class DialogParameterValues {
   15:
   16:     public static AlertDialog.Builder build(String name, ParameterMetadata metadat
a, double value, DialogInterface.OnClickListener listener, Context context) {
   17:
   18:         final AlertDialog.Builder builder = new AlertDialog.Builder(context)
   19:                 .setTitle(name);
   20:
   21:         try {
   22:             if(metadata == null || metadata.getValues() == null)
   23:                 throw new IllegalArgumentException();
   24:
   25:             final Map<Double,String> values = metadata.parseValues();
   26:             if(values.isEmpty())
   27:                 throw new IllegalArgumentException();
   28:
   29:             int i = 0, checkedItem = -1;
   30:             final String[] items = new String[values.size()];
   31:             for (Map.Entry<Double, String> entry : values.entrySet()) {
   32:                 if(entry.getKey() == value)
   33:                     checkedItem = i;
   34:                 items[i++] = entry.getValue();
   35:             }
   36:
   37:             builder.setSingleChoiceItems(items, checkedItem, listener)
   38:                 .setNegativeButton(android.R.string.cancel, null);
   39:
   40:         } catch (Throwable ex) {
   41:             builder.setIcon(android.R.drawable.ic_dialog_alert)
   42:                     .setMessage("Allowed values for this parameter are not availab
le.\nPlease edit the value directly.");
   43:         }
   44:
   45:         return builder;
   46:     }
   47:
   48:     public static AlertDialog.Builder build(String name, ParameterMetadata metadat
a, String value, DialogInterface.OnClickListener listener, Context context) {
   49:         try {
   50:             final double dval = Parameter.getFormat().parse(value).doubleValue();
   51:             return build(name, metadata, dval, listener, context);
   52:
   53:         } catch (ParseException ex) {
   54:             return new AlertDialog.Builder(context)
   55:                     .setIcon(android.R.drawable.ic_dialog_alert)
   56:                     .setTitle(name)
   57:                     .setMessage("Value missing or invalid");
   58:         }
   59:     }
   60: }
```

```
  1: package com.droidplanner;
  2:
  3: import android.app.Application;
  4: import android.os.Handler;
  5:
  6: import com.MAVLink.Messages.MAVLinkMessage;
  7: import com.MAVLink.Messages.ardupilotmega.msg_heartbeat;
  8: import com.MAVLink.Messages.enums.MAV_MODE_FLAG;
  9: import com.droidplanner.MAVLink.MavLinkMsgHandler;
 10: import com.droidplanner.MAVLink.MavLinkStreamRates;
 11: import com.droidplanner.drone.Drone;
 12: import com.droidplanner.helpers.FollowMe;
 13: import com.droidplanner.helpers.RecordMe;
 14: import com.droidplanner.helpers.TTS;
 15: import com.droidplanner.service.MAVLinkClient;
 16: import com.droidplanner.service.MAVLinkClient.OnMavlinkClientListner;
 17:
 18: public class DroidPlannerApp extends Application implements
 19:                 OnMavlinkClientListner {
 20:
 21:         private static long HEARTBEAT_NORMAL_TIMEOUT = 5000;
 22:         private static long HEARTBEAT_LOST_TIMEOUT = 15000;
 23:
 24:         public Drone drone;
 25:         private MavLinkMsgHandler mavLinkMsgHandler;
 26:         public FollowMe followMe;
 27:         public RecordMe recordMe;
 28:         public ConnectionStateListner conectionListner;
 29:         public OnSystemArmListener onSystemArmListener;
 30:         private TTS tts;
 31:
 32:     enum HeartbeatState {
 33:         FIRST_HEARTBEAT, LOST_HEARTBEAT, NORMAL_HEARTBEAT
 34:     }
 35:
 36:     private HeartbeatState heartbeatState;
 37:     private Handler watchdog = new Handler();
 38:     private Runnable watchdogCallback = new Runnable()
 39:     {
 40:         @Override
 41:         public void run()
 42:         {
 43:             onHeartbeatTimeout();
 44:         }
 45:     };
 46:
 47:         public interface OnWaypointChangedListner {
 48:                 public void onMissionUpdate();
 49:         }
 50:
 51:         public interface ConnectionStateListner {
 52:                 public void notifyConnected();
 53:
 54:                 public void notifyDisconnected();
 55:         }
 56:
 57:         public interface OnSystemArmListener {
 58:                 public void notifyArmed();
 59:
 60:                 public void notifyDisarmed();
 61:         }
 62:
 63:         @Override
 64:         public void onCreate() {
 65:                 super.onCreate();
 66:
 67:                 tts = new TTS(this);
 68:                 MAVLinkClient MAVClient = new MAVLinkClient(this, this);
 69:                 drone = new Drone(tts, MAVClient, getApplicationContext());
 70:                 followMe = new FollowMe(this, drone);
 71:                 recordMe = new RecordMe(this, drone);
 72:                 mavLinkMsgHandler = new com.droidplanner.MAVLink.MavLinkMsgHandler
(
 73:                                 drone);
 74:         }
 75:
 76:         @Override
 77:         public void notifyReceivedData(MAVLinkMessage msg) {
 78:                 if(msg.msgid == msg_heartbeat.MAVLINK_MSG_ID_HEARTBEAT){
 79:                         msg_heartbeat msg_heart = (msg_heartbeat) msg;
 80:                         if((msg_heart.base_mode & (byte) MAV_MODE_FLAG.MAV_MODE_FL
AG_SAFETY_ARMED) == (byte) MAV_MODE_FLAG.MAV_MODE_FLAG_SAFETY_ARMED){
 81:                                 notifyArmed();
 82:                         }
 83:                         else {
 84:                                 notifyDisarmed();
 85:                         }
 86:                         onHeartbeat();
 87:                 }
 88:                 mavLinkMsgHandler.receiveData(msg);
 89:         }
 90:
 91:         @Override
 92:         public void notifyDisconnected() {
 93:                 conectionListner.notifyDisconnected();
 94:                 tts.speak("Disconnected");
 95:
 96:                 // stop watchdog
 97:                 watchdog.removeCallbacks(watchdogCallback);
 98:         }
 99:
100:         @Override
101:         public void notifyConnected() {
102:                 MavLinkStreamRates.setupStreamRatesFromPref(this);
103:                 conectionListner.notifyConnected();
104:                 // don't announce 'connected' until first heartbeat received
105:
106:                 // start watchdog
107:                 heartbeatState = HeartbeatState.FIRST_HEARTBEAT;
108:                 restartWatchdog(HEARTBEAT_NORMAL_TIMEOUT);
109:         }
110:
111:         @Override
112:         public void notifyArmed() {
113:                 onSystemArmListener.notifyArmed();
114:         }
115:
116:         @Override
117:         public void notifyDisarmed() {
118:                 onSystemArmListener.notifyDisarmed();
119:         }
120:
121:         private void onHeartbeat() {
122:
123:                 switch(heartbeatState) {
124:                         case FIRST_HEARTBEAT:
125:                                 tts.speak("Connected");
126:                                 break;
127:
128:                         case LOST_HEARTBEAT:
129:                                 tts.speak("Data link restored");
130:                                 break;
131:                 case NORMAL_HEARTBEAT:
132:                         break;
```

```
133:                }
134:
135:                heartbeatState = HeartbeatState.NORMAL_HEARTBEAT;
136:                restartWatchdog(HEARTBEAT_NORMAL_TIMEOUT);
137:        }
138:
139:        private void onHeartbeatTimeout() {
140:                tts.speak("Data link lost, check connection.");
141:                heartbeatState = HeartbeatState.LOST_HEARTBEAT;
142:                restartWatchdog(HEARTBEAT_LOST_TIMEOUT);
143:        }
144:
145:        private void restartWatchdog(long timeout)
146:        {
147:                // re-start watchdog
148:                watchdog.removeCallbacks(watchdogCallback);
149:                watchdog.postDelayed(watchdogCallback, timeout);
150:        }
151: }
```

```
 1: package com.droidplanner.drone;
 2:
 3:
 4: import android.content.Context;
 5:
 6: import com.droidplanner.drone.DroneInterfaces.DroneTypeListner;
 7: import com.droidplanner.drone.DroneInterfaces.HomeDistanceChangedListner;
 8: import com.droidplanner.drone.DroneInterfaces.HudUpdatedListner;
 9: import com.droidplanner.drone.DroneInterfaces.InfoListner;
10: import com.droidplanner.drone.DroneInterfaces.MapConfigListener;
11: import com.droidplanner.drone.DroneInterfaces.MapUpdatedListner;
12: import com.droidplanner.drone.DroneInterfaces.ModeChangedListener;
13: import com.droidplanner.drone.DroneInterfaces.OnTuningDataListner;
14: import com.droidplanner.drone.variables.Altitude;
15: import com.droidplanner.drone.variables.Battery;
16: import com.droidplanner.drone.variables.Calibration;
17: import com.droidplanner.drone.variables.GPS;
18: import com.droidplanner.drone.variables.GuidedPoint;
19: import com.droidplanner.drone.variables.GuidedPoint.OnGuidedListener;
20: import com.droidplanner.drone.variables.Home;
21: import com.droidplanner.drone.variables.MissionStats;
22: import com.droidplanner.drone.variables.Navigation;
23: import com.droidplanner.drone.variables.Orientation;
24: import com.droidplanner.drone.variables.Parameters;
25: import com.droidplanner.drone.variables.RC;
26: import com.droidplanner.drone.variables.Speed;
27: import com.droidplanner.drone.variables.State;
28: import com.droidplanner.drone.variables.Type;
29: import com.droidplanner.drone.variables.mission.Mission;
30: import com.droidplanner.drone.variables.mission.WaypointMananger;
31: import com.droidplanner.helpers.TTS;
32: import com.droidplanner.service.MAVLinkClient;
33:
34: public class Drone {
35:         public Type type = new Type(this);
36:         public GPS GPS = new GPS(this);
37:         public RC RC = new RC(this);
38:         public Speed speed = new Speed(this);
39:         public State state = new State(this);
40:         public Battery battery = new Battery(this);
41:         public Home home = new Home(this);
42:         public Mission mission = new Mission(this);
43:         public MissionStats missionStats = new MissionStats(this);
44:         public Altitude altitude = new Altitude(this);
45:         public Orientation orientation = new Orientation(this);
46:         public Navigation navigation = new Navigation(this);
47:         public GuidedPoint guidedPoint = new GuidedPoint(this);
48:         public Parameters parameters = new Parameters(this);
49:         public Calibration calibrationSetup = new Calibration(this);
50:         public WaypointMananger waypointMananger = new WaypointMananger(this);
51:
52:         public TTS tts;
53:         public MAVLinkClient MavClient;
54:         public Context context;
55:
56:         private HudUpdatedListner hudListner;
57:         private MapUpdatedListner mapListner;
58:         private MapConfigListener mapConfigListener;
59:         private DroneTypeListner typeListner;
60:         private InfoListner infoListner;
61:         private HomeDistanceChangedListner homeChangedListner;
62:         private ModeChangedListener modeChangedListener;
63:         private OnGuidedListener guidedListner;
64:         public OnTuningDataListner tuningDataListner;
65:
66:         public Drone(TTS tts, MAVLinkClient mavClient, Context context) {
67:                 this.tts = tts;
68:                 this.MavClient = mavClient;
69:                 this.context = context;
70:         }
71:
72:         public void setHudListner(HudUpdatedListner listner) {
73:                 hudListner = listner;
74:         }
75:
76:         public void setMapListner(MapUpdatedListner listner) {
77:                 mapListner = listner;
78:         }
79:
80:         public void setMapConfigListener(MapConfigListener mapConfigListener) {
81:                 this.mapConfigListener = mapConfigListener;
82:         }
83:
84:         public void setDroneTypeChangedListner(DroneTypeListner listner) {
85:                 typeListner = listner;
86:         }
87:
88:         public void setInfoListner(InfoListner listner) {
89:                 infoListner = listner;
90:         }
91:
92:         public void setHomeChangedListner(HomeDistanceChangedListner listner) {
93:                 homeChangedListner = listner;
94:         }
95:
96:         public void setTuningDataListner(OnTuningDataListner listner) {
97:                 tuningDataListner = listner;
98:         }
99:
100:        public void setModeChangedListener(ModeChangedListener listener){
101:                this.modeChangedListener = listener;
102:        }
103:
104:        public void setGuidedPointListner(OnGuidedListener listner) {
105:                guidedListner = listner;
106:        }
107:
108:        public void setAltitudeGroundAndAirSpeeds(double altitude,
109:                        double groundSpeed, double airSpeed, double climb) {
110:                this.altitude.setAltitude(altitude);
111:                speed.setGroundAndAirSpeeds(groundSpeed, airSpeed, climb);
112:                onSpeedAltitudeAndClimbRateUpdate();
113:        }
114:
115:        public void setDisttowpAndSpeedAltErrors(double disttowp, double alt_error
                ,
116:                        double aspd_error) {
117:                missionStats.setDistanceToWp(disttowp);
118:                altitude.setAltitudeError(alt_error);
119:                speed.setSpeedError(aspd_error);
120:                onOrientationUpdate();
121:        }
122:
123:        public void notifyPositionChange() {
124:                if (mapListner != null) {
125:                        mapListner.onDroneUpdate();
126:                }
127:        }
128:
129:        public void notifyGuidedPointChange() {
130:                if (guidedListner != null) {
131:                        guidedListner.onGuidedPoint();
132:                }
133:        }
```

```
134:
135:         public void notifyInfoChange() {
136:                 if (infoListner != null) {
137:                         infoListner.onInfoUpdate();
138:                 }
139:         }
140:
141:         public void notifyDistanceToHomeChange() {
142:                 if (homeChangedListner!= null) {
143:                         homeChangedListner.onDistanceToHomeHasChanged();
144:                 }
145:         }
146:
147:         public void notifyTypeChanged() {
148:                 if (typeListner != null) {
149:                         typeListner.onDroneTypeChanged();
150:                 }
151:                 if (mapListner != null) {
152:                         mapListner.onDroneTypeChanged();
153:                 }
154:
155:         }
156:
157:         public void onOrientationUpdate() {
158:                 if (hudListner != null)
159:                         hudListner.onOrientationUpdate();
160:         }
161:
162:         public void onSpeedAltitudeAndClimbRateUpdate() {
163:                 if (hudListner != null)
164:                         hudListner.onSpeedAltitudeAndClimbRateUpdate();
165:         }
166:
167:         public void notifyMapTypeChanged() {
168:                 if (mapConfigListener != null)
169:                         mapConfigListener.onMapTypeChanged();
170:         }
171:
172:         public void notifyModeChanged(){
173:                 if (modeChangedListener != null)
174:                         modeChangedListener.onModeChanged();
175:         }
176:
177: }
```

```
    1: package com.droidplanner.drone;
    2:
    3: import com.droidplanner.parameters.Parameter;
    4:
    5: import java.util.List;
    6:
    7: public class DroneInterfaces {
    8:         public interface MapUpdatedListner {
    9:                 public void onDroneUpdate();
   10:                 public void onDroneTypeChanged();
   11:         }
   12:
   13:         public interface MapConfigListener {
   14:                 public void onMapTypeChanged();
   15:         }
   16:
   17:         public interface DroneTypeListner {
   18:                 public void onDroneTypeChanged();
   19:         }
   20:
   21:         public interface InfoListner {
   22:                 public void onInfoUpdate();
   23:         }
   24:
   25:         public interface HomeDistanceChangedListner {
   26:                 public void onDistanceToHomeHasChanged();
   27:         }
   28:
   29:         public interface HudUpdatedListner {
   30:                 public void onOrientationUpdate();
   31:                 public void onSpeedAltitudeAndClimbRateUpdate();
   32:         }
   33:
   34:         public interface ModeChangedListener {
   35:                 public void onModeChanged();
   36:         }
   37:
   38:         public interface OnParameterManagerListner {
   39:                 public void onBeginReceivingParameters();
   40:                 public void onParameterReceived(Parameter parameter, int index, in
t count);
   41:                 public void onEndReceivingParameters(List<Parameter> parameter);
   42:                 public void onParamterMetaDataChanged();
   43:         }
   44:
   45:         public interface OnStateListner {
   46:                 void onFlightStateChanged();
   47:
   48:                 void onArmChanged();
   49:
   50:                 void onFailsafeChanged();
   51:         }
   52:
   53:         public interface OnTuningDataListner{
   54:                 void onNewOrientationData();
   55:
   56:                 void onNewNavigationData();
   57:         }
   58:
   59:         public interface OnRcDataChangedListner{
   60:                 void onNewOutputRcData();
   61:                 void onNewInputRcData();
   62:         }
   63: }
```

```
1: package com.droidplanner.drone;
2:
3: public class DroneVariable {
4:         protected Drone myDrone;
5:
6:         public DroneVariable(Drone myDrone) {
7:                 this.myDrone = myDrone;
8:         }
9: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5:
 6: public class Altitude extends DroneVariable {
 7:         private double altitude = 0;
 8:         private double targetAltitude = 0;
 9:
10:         public Altitude(Drone myDrone) {
11:                 super(myDrone);
12:         }
13:
14:         public double getAltitude() {
15:                 return altitude;
16:         }
17:
18:         public double getTargetAltitude() {
19:                 return targetAltitude;
20:         }
21:
22:         public void setAltitude(double altitude) {
23:                 this.altitude = altitude;
24:         }
25:
26:         public void setAltitudeError(double alt_error) {
27:                 targetAltitude = alt_error + altitude;
28:         }
29:
30: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5:
 6: public class Battery extends DroneVariable {
 7:         private double battVolt = -1;
 8:         private double battRemain = -1;
 9:         private double battCurrent = -1;
10:
11:         public Battery(Drone myDrone) {
12:                 super(myDrone);
13:         }
14:
15:         public double getBattVolt() {
16:                 return battVolt;
17:         }
18:
19:         public double getBattRemain() {
20:                 return battRemain;
21:         }
22:
23:         public double getBattCurrent() {
24:                 return battCurrent;
25:         }
26:
27:         public void setBatteryState(double battVolt, double battRemain,
28:                         double battCurrent) {
29:                 if (this.battVolt != battVolt | this.battRemain != battRemain
30:                                 | this.battCurrent != battCurrent) {
31:                         myDrone.tts.batteryDischargeNotification(battRemain);
32:                         this.battVolt = battVolt;
33:                         this.battRemain = battRemain;
34:                         this.battCurrent = battCurrent;
35:                         myDrone.notifyInfoChange();
36:                 }
37:         }
38: }
```

```java
package com.droidplanner.drone.variables;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.ardupilotmega.msg_statustext;
import com.droidplanner.MAVLink.MavLinkCalibration;
import com.droidplanner.drone.Drone;
import com.droidplanner.drone.DroneVariable;

public class Calibration extends DroneVariable implements OnClickListener {
        private Context context;

        private int count;

        public Calibration(Drone drone) {
                super(drone);
        }

        public void startCalibration(Context context) {
                this.context = context;
                MavLinkCalibration.sendStartCalibrationMessage(myDrone);
                count = 0;
        }

        @Override
        public void onClick(DialogInterface dialog, int id) {
                count++;
                MavLinkCalibration.sendCalibrationAckMessage(count, myDrone);
                if (count >= 6) {
                        createDialog("Calibration Done!");
                        count = 0;
                }
        }

        public void processMessage(MAVLinkMessage msg) {
                if (msg.msgid == msg_statustext.MAVLINK_MSG_ID_STATUSTEXT) {
                        msg_statustext statusMsg = (msg_statustext) msg;
                        if (statusMsg.getText().contains("Place APM")) {
                                createDialog(statusMsg.getText());
                        }
                }
        }

        private void createDialog(String message) {
                AlertDialog.Builder builder = new AlertDialog.Builder(context);
                builder.setMessage(message);
                builder.setPositiveButton("Ok", this);
                builder.setCancelable(false);
                builder.create();
                builder.show();
        }

}
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5: import com.google.android.gms.maps.model.LatLng;
 6:
 7: public class GPS extends DroneVariable {
 8:         private double gps_eph = -1;
 9:         private int satCount = -1;
10:         private int fixType = -1;
11:         private LatLng position;
12:
13:         public GPS(Drone myDrone) {
14:                 super(myDrone);
15:         }
16:
17:         public boolean isPositionValid() {
18:                 return (position != null);
19:         }
20:
21:         public LatLng getPosition() {
22:                 if (isPositionValid()) {
23:                         return position;
24:                 }else{
25:                         return new LatLng(0, 0);
26:                 }
27:         }
28:
29:         public double getGpsEPH() {
30:                 return gps_eph;
31:         }
32:
33:         public int getSatCount() {
34:                 return satCount;
35:         }
36:
37:         public String getFixType() {
38:                 String gpsFix = "";
39:                 switch (fixType) {
40:                 case 2:
41:                         gpsFix = ("2D");
42:                         break;
43:                 case 3:
44:                         gpsFix = ("3D");
45:                         break;
46:                 default:
47:                         gpsFix = ("NoFix");
48:                         break;
49:                 }
50:                 return gpsFix;
51:         }
52:
53:         public void setGpsState(int fix, int satellites_visible, int eph) {
54:                 if (satCount != satellites_visible | fixType != fix) {
55:                         if (fixType != fix) {
56:                                 myDrone.tts.speakGpsMode(fix);
57:                         }
58:                         fixType = fix;
59:                         satCount = satellites_visible;
60:                         gps_eph = (double) eph / 100; // convert from eph(cm) to g
ps_eph(m)
61:
62:                         myDrone.notifyInfoChange();
63:                 }
64:         }
65:
66:         public void setPosition(LatLng position) {
67:                 if (this.position != position) {
68:                         this.position = position;
69:                         myDrone.notifyPositionChange();
70:                         myDrone.notifyDistanceToHomeChange();
71:                 }
72:         }
73: }
```

```
    1: package com.droidplanner.drone.variables;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.content.Context;
    7: import android.widget.Toast;
    8:
    9: import com.droidplanner.MAVLink.MavLinkModes;
   10: import com.droidplanner.drone.Drone;
   11: import com.droidplanner.drone.DroneVariable;
   12: import com.droidplanner.fragments.helpers.MapPath.PathSource;
   13: import com.droidplanner.fragments.markers.GuidedMarker;
   14: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   15: import com.droidplanner.helpers.units.Altitude;
   16: import com.google.android.gms.maps.model.LatLng;
   17: import com.google.android.gms.maps.model.Marker;
   18: import com.google.android.gms.maps.model.MarkerOptions;
   19:
   20: public class GuidedPoint extends DroneVariable implements MarkerSource, PathSource
 {
   21:         private LatLng coord;
   22:         private Altitude altitude;
   23:
   24:         public interface OnGuidedListener {
   25:                 public void onGuidedPoint();
   26:         }
   27:
   28:         public GuidedPoint(Drone myDrone) {
   29:                 super(myDrone);
   30:         }
   31:
   32:         public void newGuidedPointWithCurrentAlt(LatLng coord) {
   33:                 altitude = new Altitude(myDrone.altitude.getAltitude());
   34:                 newGuidedPointwithLastAltitude(coord);
   35:         }
   36:
   37:         public void newGuidedPointwithLastAltitude(LatLng coord) {
   38:                 this.coord = coord;
   39:                 sendGuidedPoint();
   40:         }
   41:
   42:         private void sendGuidedPoint() {
   43:                 myDrone.notifyGuidedPointChange();
   44:                 MavLinkModes.setGuidedMode(myDrone, coord.latitude, coord.longitud
e,
   45:                                 this.altitude.valueInMeters());
   46:                 Toast.makeText(myDrone.context, "Guided Mode (" + altitude + ")",
   47:                                 Toast.LENGTH_SHORT).show();
   48:         }
   49:
   50:         public LatLng getCoord() {
   51:                 return coord;
   52:         }
   53:
   54:         public void invalidateCoord() {
   55:                 if (isValid()) {
   56:                         coord = null;
   57:                         altitude = null;
   58:                         myDrone.notifyGuidedPointChange();
   59:                 }
   60:         }
   61:
   62:         public boolean isValid() {
   63:                 return (coord != null) & (altitude != null);
   64:         }
   65:
   66:         @Override
   67:         public MarkerOptions build(Context context) {
   68:                 return GuidedMarker.build(this, altitude, context);
   69:         }
   70:
   71:         @Override
   72:         public void update(Marker markerFromGcp, Context context) {
   73:                 GuidedMarker.update(markerFromGcp, this, altitude, context);
   74:         }
   75:
   76:         @Override
   77:         public List<LatLng> getPathPoints() {
   78:                 List<LatLng> path = new ArrayList<LatLng>();
   79:                 if (isValid()) {
   80:                         path.add(myDrone.GPS.getPosition());
   81:                         path.add(coord);
   82:                 }
   83:                 return path;
   84:         }
   85: }
```

```
    1: package com.droidplanner.drone.variables;
    2:
    3: import android.content.Context;
    4:
    5: import com.droidplanner.drone.Drone;
    6: import com.droidplanner.drone.DroneVariable;
    7: import com.droidplanner.fragments.markers.HomeMarker;
    8: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
    9: import com.droidplanner.helpers.geoTools.GeoTools;
   10: import com.droidplanner.helpers.units.Altitude;
   11: import com.droidplanner.helpers.units.Length;
   12: import com.google.android.gms.maps.model.LatLng;
   13: import com.google.android.gms.maps.model.Marker;
   14: import com.google.android.gms.maps.model.MarkerOptions;
   15:
   16: public class Home extends DroneVariable implements MarkerSource {
   17:         private LatLng coordinate;
   18:         private Altitude altitude = new Altitude(0);
   19:
   20:         public Home(Drone drone) {
   21:                 super(drone);
   22:         }
   23:
   24:         @Override
   25:         public MarkerOptions build(Context context) {
   26:                 return HomeMarker.build(this);
   27:         }
   28:
   29:         @Override
   30:         public void update(Marker marker, Context context) {
   31:                 HomeMarker.update(marker, this);
   32:         }
   33:
   34:
   35:         public boolean isValid() {
   36:                 return (coordinate!=null);
   37:         }
   38:
   39:         public Home getHome() {
   40:                 return this;
   41:         }
   42:
   43:         public Length getDroneDistanceToHome() {
   44:                 if (isValid()) {
   45:                         return new Length(GeoTools.getDistance(coordinate, myDrone
.GPS.getPosition()));
   46:                 }else{
   47:                         return new Length(0); // TODO fix this
   48:                 }
   49:         }
   50:
   51:         public LatLng getCoord() {
   52:                 return coordinate;
   53:         }
   54:
   55:         public Length getAltitude() {
   56:                 return altitude;
   57:         }
   58:
   59: }
```

```
 1: package com.droidplanner.drone.variables.mission.commands;
 2:
 3: import java.util.List;
 4:
 5: import com.droidplanner.drone.variables.mission.MissionItem;
 6: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 7: import com.google.android.gms.maps.model.LatLng;
 8:
 9: public abstract class MissionCMD extends MissionItem{
10:
11:         public MissionCMD(MissionItem item) {
12:         }
13:
14:         @Override
15:         public List<LatLng> getPath() throws Exception {
16:                 throw new Exception();
17:         }
18:
19:         @Override
20:         public List<MarkerSource> getMarkers() throws Exception {
21:                 throw new Exception();
22:         }
23:
24: }
```

```
 1: package com.droidplanner.drone.variables.mission.commands;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
 4: import com.droidplanner.drone.variables.mission.MissionItem;
 5: import com.droidplanner.fragments.mission.MissionDetailFragment;
 6: import com.droidplanner.fragments.mission.MissionRTLFragment;
 7: import com.droidplanner.helpers.units.Altitude;
 8:
 9: public class ReturnToHome extends MissionCMD{
10:         private Altitude returnAltitude;
11:
12:         public ReturnToHome(MissionItem item) {
13:                 super(item);
14:                 returnAltitude = new Altitude(0);
15:         }
16:
17:         @Override
18:         public MissionDetailFragment getDetailFragment() {
19:                 MissionDetailFragment fragment = new MissionRTLFragment();
20:                 fragment.setItem(this);
21:                 return fragment;
22:         }
23:
24:         public Altitude  getHeight() {
25:                 return returnAltitude;
26:         }
27:
28:         public void setHeight(Altitude altitude) {
29:                 returnAltitude = altitude;
30:         }
31:
32:         @Override
33:         public msg_mission_item packMissionItem() {
34:                 // TODO Auto-generated method stub
35:                 return null;
36:         }
37:
38:         @Override
39:         public void unpackMAVMessage(msg_mission_item mavMessageItem) {
40:                 // TODO Auto-generated method stub
41:
42:         }
43: }
```

```java
package com.droidplanner.drone.variables.mission;

import java.util.ArrayList;
import java.util.List;

import android.widget.Toast;

import com.MAVLink.Messages.ardupilotmega.msg_mission_ack;
import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
import com.droidplanner.DroidPlannerApp.OnWaypointChangedListner;
import com.droidplanner.drone.Drone;
import com.droidplanner.drone.DroneVariable;
import com.droidplanner.drone.variables.mission.survey.Survey;
import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
import com.droidplanner.drone.variables.mission.waypoints.Waypoint;
import com.droidplanner.fragments.helpers.MapPath.PathSource;
import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
import com.droidplanner.helpers.units.Altitude;
import com.google.android.gms.maps.model.LatLng;

public class Mission extends DroneVariable implements PathSource,
                OnWaypointChangedListner {

        private List<MissionItem> itens = new ArrayList<MissionItem>();
        private Altitude defaultAlt = new Altitude(50.0);

        private List<OnWaypointChangedListner> missionListner = new ArrayList<OnWaypointChangedListner>();

        public Mission(Drone myDrone) {
                super(myDrone);
        }

        public Altitude getDefaultAlt() {
                return defaultAlt;
        }

        public void setDefaultAlt(Altitude newAltitude) {
                defaultAlt = newAltitude;
        }

        public void removeWaypoint(SpatialCoordItem waypoint) {
                itens.remove(waypoint);
                onMissionUpdate();
        }

        public void addWaypointsWithDefaultAltitude(List<LatLng> points) {
                for (LatLng point : points) {
                        itens.add(new Waypoint(point,defaultAlt));
                }
                onMissionUpdate();
        }

        public void addWaypoint(LatLng point, Altitude alt) {
                itens.add(new Waypoint(point,alt));
                onMissionUpdate();
        }

        public void replace(MissionItem oldItem, MissionItem newItem) {
                int index = itens.indexOf(oldItem);
                itens.remove(index);
                itens.add(index, newItem);
                onMissionUpdate();
        }

        public void addSurveyPolygon(List<LatLng> points) {
                Survey survey = new Survey(points, myDrone.context);
                itens.add(survey);
                onMissionUpdate();
        }

        public void addOnMissionUpdateListner(OnWaypointChangedListner listner) {
                if (!missionListner.contains(listner)) {
                        missionListner.add(listner);
                }

        }

        public void onMissionReceived(List<msg_mission_item> mission) {
                throw new IllegalArgumentException("NOT implemented"); //TODO implement this
                /*
                if (mission != null) {
                        Toast.makeText(myDrone.context, "Waypoints received from Drone",
                                        Toast.LENGTH_SHORT).show();
                        myDrone.tts.speak("Waypoints received");
                        home.updateData(mission.get(0));
                        mission.remove(0); // Remove Home waypoint
                        clearWaypoints();
                        addWaypoints(mission);
                        onMissionUpdate();
                        myDrone.notifyDistanceToHomeChange();
                }
                */
        }

        public void sendMissionToAPM() {
                throw new IllegalArgumentException("NOT implemented"); //TODO implement this
                /*
                List<Waypoint> data = new ArrayList<Waypoint>();
                data.add(myDrone.home.getHome());
                data.addAll(getWaypoints());
                myDrone.waypointMananger.writeWaypoints(data);
                */
        }

        public void onWriteWaypoints(msg_mission_ack msg) {
                Toast.makeText(myDrone.context, "Waypoints sent", Toast.LENGTH_SHORT)
                                .show();
                myDrone.tts.speak("Waypoints saved to Drone");
        }

        @Override
        public List<LatLng> getPathPoints() {
                List<LatLng> newPath = new ArrayList<LatLng>();
                for (MissionItem item : itens) {
                        try {
                                newPath.addAll(item.getPath());
                        } catch (Exception e) {
                                // Exception when no path for the item
                        }
                }
                return newPath;
        }

        public List<MissionItem> getItems() {
                return itens;
        }

        public List<MarkerSource> getMarkers() {
                List<MarkerSource> markers = new ArrayList<MarkerSource>();
```

```
130:                 for (MissionItem item : itens) {
131:                         try {
132:                                 markers.addAll(item.getMarkers());
133:                         } catch (Exception e) {
134:                                 // Exception when no markers for the item
135:                         }
136:                 }
137:                 return markers;
138:         }
139:
140:         @Override
141:         public void onMissionUpdate() {
142:                 for (OnWaypointChangedListner listner : missionListner) {
143:                         if (listner!=null) {
144:                                 listner.onMissionUpdate();
145:                         }
146:                 }
147:         }
148:
149:         public void removeOnMissionUpdateListner(
150:                         OnWaypointChangedListner listner) {
151:                 missionListner.remove(listner);
152:         }
153:
154: }
```

```
 1: package com.droidplanner.drone.variables.mission;
 2:
 3: import java.util.List;
 4:
 5: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
 6: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 7: import com.droidplanner.fragments.mission.MissionDetailFragment;
 8: import com.google.android.gms.maps.model.LatLng;
 9:
10: public abstract class MissionItem {
11:
12:         /**
13:          * Gets a flight path for this item
14:          * @return the path as a list
15:          * @throws Exception if path not available
16:          */
17:         public abstract List<LatLng> getPath() throws Exception;
18:
19:         /**
20:          * Gets all markers for this item
21:          * @return list of markers
22:          * @throws Exception if this item doesn't have markers
23:          */
24:         public abstract List<MarkerSource> getMarkers() throws Exception;
25:
26:         /**
27:          * Return a new detail Fragment for this MissionItem
28:          * @return
29:          */
30:         public abstract MissionDetailFragment getDetailFragment();
31:
32:         /**
33:          * Return a new MAVLinkMessage msg_mission_item for this MissionItem
34:          * @return
35:          */
36:         public abstract msg_mission_item packMissionItem();
37:
38:         /**
39:          * Gets data from MAVLinkMessage msg_mission_item for this MissionItem
40:          * @return
41:          */
42:         public abstract void unpackMAVMessage(msg_mission_item mavMsg);
43:
44: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey.grid;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.droidplanner.helpers.geoTools.GeoTools;
    7: import com.droidplanner.helpers.geoTools.LineLatLng;
    8: import com.droidplanner.polygon.PolyBounds;
    9: import com.google.android.gms.maps.model.LatLng;
   10:
   11: public class CircumscribedGrid {
   12:         private static final int MAX_NUMBER_OF_LINES = 200;
   13:         List<LineLatLng> grid = new ArrayList<LineLatLng>();
   14:         private LatLng gridLowerLeft;
   15:         private double extrapolatedDiag;
   16:         private Double angle;
   17:
   18:         public CircumscribedGrid(List<LatLng> polygonPoints, Double angle,
   19:                         Double lineDist) throws Exception {
   20:                 this.angle = angle;
   21:
   22:                 findPolygonBounds(polygonPoints);
   23:                 drawGrid(lineDist);
   24:         }
   25:
   26:         private void drawGrid(Double lineDist) throws Exception {
   27:                 int lines = 0;
   28:                 LatLng startPoint = gridLowerLeft;
   29:                 while (lines * lineDist < extrapolatedDiag) {
   30:                         LatLng endPoint = GeoTools.newCoordFromBearingAndDistance(
   31:                                         startPoint, angle, extrapolatedDiag);
   32:
   33:                         LineLatLng line = new LineLatLng(startPoint, endPoint);
   34:                         grid.add(line);
   35:
   36:                         startPoint = GeoTools.newCoordFromBearingAndDistance(start
Point,
   37:                                         angle + 90, lineDist);
   38:                         lines++;
   39:                         if (lines>MAX_NUMBER_OF_LINES) {
   40:                                 throw new Exception("Mission is too lengthy");

   41:                         }
   42:                 }
   43:         }
   44:
   45:         private void findPolygonBounds(List<LatLng> polygonPoints) {
   46:                 PolyBounds bounds = new PolyBounds(polygonPoints);
   47:                 LatLng middlePoint = bounds.getMiddle();
   48:                 gridLowerLeft = GeoTools.newCoordFromBearingAndDistance(middlePoin
t,
   49:                                 angle - 135, bounds.getDiag());
   50:                 extrapolatedDiag = bounds.getDiag() * 1.5;
   51:         }
   52:
   53:         public List<LineLatLng> getGrid() {
   54:                 return grid;
   55:         }
   56:
   57: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey.grid;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.droidplanner.helpers.geoTools.LineLatLng;
    7: import com.droidplanner.helpers.geoTools.LineSampler;
    8: import com.droidplanner.helpers.geoTools.LineTools;
    9: import com.google.android.gms.maps.model.LatLng;
   10:
   11: public class EndpointSorter {
   12:         private static final int MAX_NUMBER_OF_CAMERAS = 2000;
   13:
   14:         private List<LatLng> gridPoints = new ArrayList<LatLng>();
   15:         private List<LineLatLng> grid;
   16:         private Double sampleDistance;
   17:         private List<LatLng> cameraLocations = new ArrayList<LatLng>();
   18:
   19:         public EndpointSorter(List<LineLatLng> grid, Double sampleDistance) {
   20:                 this.grid = grid;
   21:                 this.sampleDistance = sampleDistance;
   22:         }
   23:
   24:         public void sortGrid(LatLng lastpnt, boolean innerWPs) throws Exception {
   25:                 while (grid.size() > 0) {
   26:                         LineLatLng closestLine = LineTools.findClosestLineToPoint(
lastpnt,
   27:                                         grid);
   28:                         LatLng secondWp = processOneGridLine(closestLine, lastpnt,
 innerWPs);
   29:                         lastpnt = secondWp;
   30:                 }
   31:         }
   32:
   33:         private LatLng processOneGridLine(LineLatLng closestLine, LatLng lastpnt,
   34:                         boolean innerWPs) throws Exception {
   35:                 LatLng firstWP = closestLine.getClosestEndpointTo(lastpnt);
   36:                 LatLng secondWp = closestLine.getFarthestEndpointTo(lastpnt);
   37:
   38:                 grid.remove(closestLine);
   39:
   40:                 addWaypointsBetween(firstWP, secondWp, innerWPs);
   41:                 if (cameraLocations.size()>MAX_NUMBER_OF_CAMERAS) {
   42:                         throw new Exception("Too many camera positions");
   43:                 }
   44:                 return secondWp;
   45:         }
   46:
   47:         private void addWaypointsBetween(LatLng firstWP, LatLng secondWp,
   48:                         boolean innerWPs) {
   49:                 List<LatLng> cameraLocationsOnThisStrip = new LineSampler(firstWP,
 secondWp)
   50:                                 .sample(sampleDistance);
   51:                 cameraLocations.addAll(cameraLocationsOnThisStrip);
   52:                 if (innerWPs) {
   53:                         for (LatLng point : cameraLocationsOnThisStrip) {
   54:                                 gridPoints.add(point);
   55:                         }
   56:                 } else {
   57:                         gridPoints.add(firstWP);
   58:                         gridPoints.add(secondWp);
   59:                 }
   60:         }
   61:
   62:         public List<LatLng> getSortedGrid() {
   63:                 return gridPoints;
   64:         }
   65:
   66:         public List<LatLng> getCameraLocations() {
   67:                 return cameraLocations;
   68:         }
   69:
   70: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey.grid;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
    7: import com.droidplanner.drone.variables.mission.waypoints.Waypoint;
    8: import com.droidplanner.helpers.geoTools.PolylineTools;
    9: import com.droidplanner.helpers.units.Altitude;
   10: import com.droidplanner.helpers.units.Length;
   11: import com.google.android.gms.maps.model.LatLng;
   12:
   13: public class Grid {
   14:         public List<LatLng> gridPoints;
   15:         private List<LatLng> cameraLocations;
   16:         private Altitude altitude;
   17:
   18:         public Grid(List<LatLng> list, List<LatLng> cameraLocations) {
   19:                 this.gridPoints = list;
   20:                 this.cameraLocations = cameraLocations;
   21:         }
   22:
   23:         public ArrayList<SpatialCoordItem> getWaypoints() {
   24:                 ArrayList<SpatialCoordItem> list = new ArrayList<SpatialCoordItem>
();
   25:                 for (LatLng point : gridPoints) {
   26:                         list.add(new Waypoint(point, altitude));
   27:                 }
   28:                 return list;
   29:         }
   30:
   31:         public Length getLength(){
   32:                 return PolylineTools.getPolylineLength(gridPoints);
   33:         }
   34:
   35:         public int getNumberOfLines(){
   36:                 return gridPoints.size()/2;
   37:         }
   38:
   39:         public List<LatLng> getCameraLocations() {
   40:                 return cameraLocations;
   41:         }
   42:
   43:         public void setAltitude(Altitude altitude) {
   44:                 this.altitude = altitude;
   45:         }
   46:
   47:         public int getCameraCount() {
   48:                 return getCameraLocations().size();
   49:         }
   50:
   51: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey.grid;
    2:
    3: import java.util.List;
    4:
    5: import com.droidplanner.drone.variables.mission.survey.SurveyData;
    6: import com.droidplanner.helpers.geoTools.LineLatLng;
    7: import com.droidplanner.polygon.Polygon;
    8: import com.google.android.gms.maps.model.LatLng;
    9:
   10: public class GridBuilder {
   11:
   12:         private Polygon poly;
   13:         private Double angle;
   14:         private Double lineDist;
   15:         private LatLng origin;
   16:         private boolean innerWPs;
   17:         private Double wpDistance;
   18:
   19:         private Grid grid;
   20:
   21:         public GridBuilder(Polygon polygon, SurveyData surveyData,
   22:                         LatLng originPoint) {
   23:                 this.poly = polygon;
   24:                 this.origin = originPoint;
   25:                 this.angle = surveyData.getAngle();
   26:                 this.lineDist = surveyData.getLateralPictureDistance().valueInMete
rs();
   27:                 this.innerWPs = surveyData.shouldGenerateInnerWPs();
   28:                 this.wpDistance = surveyData.getLongitudinalPictureDistance()
   29:                                 .valueInMeters();
   30:         }
   31:
   32:         public GridBuilder(Polygon polygon, double angle, double distance,
   33:                         LatLng originPoint) {
   34:                 this.poly = polygon;
   35:                 this.origin = originPoint;
   36:                 this.angle = angle;
   37:                 this.lineDist = distance;
   38:                 this.innerWPs = false;
   39:         }
   40:
   41:         public Grid generate() throws Exception {
   42:                 List<LatLng> polygonPoints = poly.getLatLngList();
   43:
   44:                 List<LineLatLng> circumscribedGrid = new CircumscribedGrid(
   45:                                 polygonPoints, angle, lineDist).getGrid();
   46:                 List<LineLatLng> trimedGrid = new Trimmer(circumscribedGrid,
   47:                                 poly.getLines()).getTrimmedGrid();
   48:                 EndpointSorter gridSorter = new EndpointSorter(trimedGrid, wpDista
nce);
   49:                 gridSorter.sortGrid(origin, innerWPs);
   50:                 grid = new Grid(gridSorter.getSortedGrid(),gridSorter.getCameraLoc
ations());
   51:                 return grid;
   52:         }
   53:
   54: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey.grid;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.droidplanner.helpers.geoTools.LineLatLng;
    7: import com.droidplanner.helpers.geoTools.LineTools;
    8: import com.google.android.gms.maps.model.LatLng;
    9:
   10: public class Trimmer {
   11:         List<LineLatLng> trimedGrid = new ArrayList<LineLatLng>();
   12:
   13:         public Trimmer(List<LineLatLng> grid, List<LineLatLng> polygon) {
   14:                 for (LineLatLng gridLine : grid) {
   15:                         ArrayList<LatLng> crosses = findCrossings(polygon, gridLin
e);
   16:                         processCrossings(crosses, gridLine);
   17:                 }
   18:         }
   19:
   20:         private ArrayList<LatLng> findCrossings(List<LineLatLng> polygon,
   21:                         LineLatLng gridLine) {
   22:
   23:                 ArrayList<LatLng> crossings = new ArrayList<LatLng>();
   24:                 for (LineLatLng polyLine : polygon) {
   25:                         try {
   26:                                 crossings.add(LineTools
   27:                                                 .FindLineIntersection(polyLine, gr
idLine));
   28:                         } catch (Exception e) {
   29:                         }
   30:                 }
   31:
   32:                 return crossings;
   33:         }
   34:
   35:         private void processCrossings(ArrayList<LatLng> crosses, LineLatLng gridLi
ne) {
   36:                 switch (crosses.size()) {
   37:                 case 0:
   38:                 case 1:
   39:                         break;
   40:                 case 2:
   41:                         trimedGrid.add(new LineLatLng(crosses.get(0), crosses.get(
1)));
   42:                         break;
   43:                 default: // TODO handle multiple crossings in a better way
   44:                         trimedGrid.add(LineTools.findExternalPoints(crosses));
   45:                 }
   46:         }
   47:
   48:         public List<LineLatLng> getTrimmedGrid() {
   49:                 return trimedGrid;
   50:         }
   51:
   52: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.content.Context;
    7: import android.widget.Toast;
    8:
    9: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
   10: import com.droidplanner.drone.variables.mission.MissionItem;
   11: import com.droidplanner.drone.variables.mission.survey.grid.Grid;
   12: import com.droidplanner.drone.variables.mission.survey.grid.GridBuilder;
   13: import com.droidplanner.file.IO.CameraInfoReader;
   14: import com.droidplanner.file.help.CameraInfoLoader;
   15: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   16: import com.droidplanner.fragments.mission.MissionDetailFragment;
   17: import com.droidplanner.fragments.mission.MissionSurveyFragment;
   18: import com.droidplanner.helpers.units.Altitude;
   19: import com.droidplanner.polygon.Polygon;
   20: import com.google.android.gms.maps.model.LatLng;
   21:
   22: public class Survey extends MissionItem {
   23:
   24:         private Polygon polygon = new Polygon();
   25:         private SurveyData surveyData = new SurveyData();
   26:         private CameraInfoLoader avaliableCameras;
   27:         private Context context;
   28:
   29:         public Survey(List<LatLng> points, Context context) {
   30:                 this.context = context;
   31:                 avaliableCameras = new CameraInfoLoader(context);
   32:                 polygon.addPoints(points);
   33:
   34:                 surveyData.setCameraInfo(CameraInfoReader.getNewMockCameraInfo());
   35:         }
   36:
   37:         @Override
   38:         public List<LatLng> getPath() throws Exception {
   39:                 surveyData.update(0, new Altitude(50), 0, 0);
   40:
   41:                 try {
   42:                         GridBuilder gridBuilder = new GridBuilder(polygon, surveyD
ata, new LatLng(0, 0));
   43:                         polygon.checkIfValid();
   44:                         Grid grid = gridBuilder.generate();
   45:                         grid.setAltitude(surveyData.getAltitude());
   46:                         return grid.getCameraLocations();
   47:                 } catch (Exception e) {
   48:                         Toast.makeText(context, e.getMessage(), Toast.LENGTH_SHORT
).show();
   49:                 }
   50:                 throw new Exception();
   51:         }
   52:
   53:         @Override
   54:         public List<MarkerSource> getMarkers() throws Exception {
   55:                 ArrayList<MarkerSource> markers = new ArrayList<MarkerSource>();
   56:                 markers.addAll(polygon.getPolygonPoints());
   57:                 return markers;
   58:         }
   59:
   60:         @Override
   61:         public MissionDetailFragment getDetailFragment() {
   62:                 MissionDetailFragment fragment = new MissionSurveyFragment();
   63:                 fragment.setItem(this);
   64:                 return fragment;
   65:         }
   66:
   67:         @Override
   68:         public msg_mission_item packMissionItem() {
   69:                 // TODO Auto-generated method stub
   70:                 return null;
   71:         }
   72:
   73:         @Override
   74:         public void unpackMAVMessage(msg_mission_item mavMsg) {
   75:                 // TODO Auto-generated method stub
   76:
   77:         }
   78:
   79: }
```

```
    1: package com.droidplanner.drone.variables.mission.survey;
    2:
    3: import java.util.Locale;
    4:
    5: import com.droidplanner.file.IO.CameraInfo;
    6: import com.droidplanner.helpers.units.Altitude;
    7: import com.droidplanner.helpers.units.Area;
    8: import com.droidplanner.helpers.units.Length;
    9:
   10: public class SurveyData {
   11:         private Altitude altitude = new Altitude(0.0);
   12:         private Double angle = 0.0;
   13:         private Double overlap = 50.0;
   14:         private Double sidelap = 60.0;
   15:         private boolean generateInnerWps = false;
   16:         private CameraInfo camera = new CameraInfo();
   17:
   18:         public void update(double angle, Altitude altitude, double overlap,
   19:                         double sidelap) {
   20:                 this.angle = angle;
   21:                 this.altitude = altitude;
   22:                 this.overlap = overlap;
   23:                 this.sidelap = sidelap;
   24:         }
   25:
   26:         public Length getLateralFootPrint() {
   27:                 return new Length(altitude.valueInMeters() * camera.getSensorLater
alSize() / camera.focalLength);
   28:
   29:         }
   30:
   31:         public Length getLongitudinalFootPrint() {
   32:                 return new Length(altitude.valueInMeters() * camera.getSensorLongi
tudinalSize()
   33:                                 / camera.focalLength);
   34:         }
   35:
   36:         public Area getGroundResolution() {
   37:                 return new Area(((altitude.valueInMeters()
   38:                                 * camera.getSensorLateralSize()
   39:                                 / camera.focalLength
   40:                                 * (altitude.valueInMeters() * camera.getSensorLong
itudinalSize() / camera.focalLength)
   41:                                 / (camera.sensorResolution * 1000)))/10000);
   42:         }
   43:
   44:         public Length getLongitudinalPictureDistance() {
   45:                 return new Length(getLongitudinalFootPrint().valueInMeters() * (1
- overlap * .01));
   46:         }
   47:
   48:         public Length getLateralPictureDistance() {
   49:                 return new Length(getLateralFootPrint().valueInMeters() * (1 - sid
elap * .01));
   50:         }
   51:
   52:         public void setCameraInfo(CameraInfo info) {
   53:                 this.camera = info;
   54:                 tryToLoadOverlapFromCamera();
   55:         }
   56:
   57:         private void tryToLoadOverlapFromCamera() {
   58:                 if (camera.overlap != null) {
   59:                         this.overlap = camera.overlap;
   60:                 }
   61:                 if (camera.sidelap != null) {
   62:                         this.sidelap = camera.sidelap;
   63:                 }
   64:         }
   65:
   66:         public void setInnerWpsState(boolean state) {
   67:                 generateInnerWps = state;
   68:         }
   69:
   70:         public Altitude getAltitude() {
   71:                 return altitude;
   72:         }
   73:
   74:         public Double getAngle() {
   75:                 return angle;
   76:         }
   77:
   78:         public double getSidelap() {
   79:                 return sidelap;
   80:         }
   81:
   82:         public double getOverlap() {
   83:                 return overlap;
   84:         }
   85:
   86:         public boolean shouldGenerateInnerWPs() {
   87:                 return generateInnerWps;
   88:         }
   89:
   90:         @Override
   91:         public String toString() {
   92:                 return String.format(Locale.US,
   93:                                 "Altitude: %f Angle %f Overlap: %f Sidelap: %f", a
ltitude,
   94:                                 angle, overlap, sidelap);
   95:         }
   96:
   97:         public void setAltitude(Altitude altitude) {
   98:                 this.altitude = altitude;
   99:         }
  100:
  101: }
```

```
package com.droidplanner.drone.variables.mission;

import java.util.ArrayList;
import java.util.List;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.ardupilotmega.msg_mission_ack;
import com.MAVLink.Messages.ardupilotmega.msg_mission_count;
import com.MAVLink.Messages.ardupilotmega.msg_mission_current;
import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
import com.MAVLink.Messages.ardupilotmega.msg_mission_item_reached;
import com.MAVLink.Messages.ardupilotmega.msg_mission_request;
import com.droidplanner.MAVLink.MavLinkWaypoint;
import com.droidplanner.drone.Drone;
import com.droidplanner.drone.DroneVariable;

/**
 * Class to manage the communication of waypoints to the MAV.
 *
 * Should be initialized with a MAVLink Object, so the manager can send messages
 * via the MAV link. The function processMessage must be called with every new
 * MAV Message.
 *
 */
public class WaypointMananger extends DroneVariable {
        /**
         * Try to receive all waypoints from the MAV.
         *
         * If all runs well the callback will return the list of waypoints.
         */
        public void getWaypoints() {
                state = waypointStates.READ_REQUEST;
                MavLinkWaypoint.requestWaypointsList(myDrone);
        }

        /**
         * Write a list of waypoints to the MAV.
         *
         * The callback will return the status of this operation
         *
         * @param data
         *             waypoints to be written
         */
        public void writeWaypoints(List<msg_mission_item> data) {
                if ((mission != null)) {
                        mission.clear();
                        mission.addAll(data);
                        writeIndex = 0;
                        state = waypointStates.WRITTING_WP;
                        MavLinkWaypoint.sendWaypointCount(myDrone, mission.size())
;
                }
        }

        /**
         * Sets the current waypoint in the MAV
         *
         * The callback will return the status of this operation
         */
        public void setCurrentWaypoint(int i) {
                if ((mission != null)) {
                        MavLinkWaypoint.sendSetCurrentWaypoint(myDrone, (short) i)
;
                }
        }

        /**
         * Callback for when a waypoint has been reached
         *
         * @param wpNumber
         *             number of the completed waypoint
         */
        public void onWaypointReached(int wpNumber) {
        }

        /**
         * Callback for a change in the current waypoint the MAV is heading for
         *
         * @param seq
         *             number of the updated waypoint
         */
        private void onCurrentWaypointUpdate(short seq) {
        }

        /**
         * number of waypoints to be received, used when reading waypoints
         */
        private short waypointCount;
        /**
         * list of waypoints used when writing or receiving
         */
        private List<msg_mission_item> mission = new ArrayList<msg_mission_item>()
;
        /**
         * waypoint witch is currently being written
         */
        private int writeIndex;

        enum waypointStates {
                IDLE, READ_REQUEST, READING_WP, WRITTING_WP, WAITING_WRITE_ACK
        }

        waypointStates state = waypointStates.IDLE;

        public WaypointMananger(Drone drone) {
                super(drone);
        }

        /**
         * Try to process a Mavlink message if it is a mission related message
         *
         * @param msg
         *             Mavlink message to process
         * @return Returns true if the message has been processed
         */
        public boolean processMessage(MAVLinkMessage msg) {
                switch (state) {
                default:
                case IDLE:
                        break;
                case READ_REQUEST:
                        if (msg.msgid == msg_mission_count.MAVLINK_MSG_ID_MISSION_
COUNT) {
                                waypointCount = ((msg_mission_count) msg).count;
                                mission.clear();
                                MavLinkWaypoint.requestWayPoint(myDrone, mission.s
ize());
                                state = waypointStates.READING_WP;
                                return true;
                        }
                        break;
                case READING_WP:
                        if (msg.msgid == msg_mission_item.MAVLINK_MSG_ID_MISSION_I
TEM) {
```

```
  129:                                    mission.add((msg_mission_item) msg);
  130:                                    if (mission.size() < waypointCount) {
  131:                                            MavLinkWaypoint.requestWayPoint(myDrone, m
ission.size());
  132:                                    } else {
  133:                                            state = waypointStates.IDLE;
  134:                                            MavLinkWaypoint.sendAck(myDrone);
  135:                                            myDrone.mission.onMissionReceived(mission)
;
  136:                                    }
  137:                                    return true;
  138:                            }
  139:                            break;
  140:                    case WRITTING_WP:
  141:                            if (msg.msgid == msg_mission_request.MAVLINK_MSG_ID_MISSIO
N_REQUEST) {
  142:                                    msg_mission_item item = mission.get(writeIndex);
  143:                                    myDrone.MavClient.sendMavPacket(item.pack());
  144:                                    writeIndex++;
  145:                                    if (writeIndex >= mission.size()) {
  146:                                            state = waypointStates.WAITING_WRITE_ACK;
  147:                                    }
  148:                                    return true;
  149:                            }
  150:                            break;
  151:                    case WAITING_WRITE_ACK:
  152:                            if (msg.msgid == msg_mission_ack.MAVLINK_MSG_ID_MISSION_AC
K) {
  153:                                    myDrone.mission.onWriteWaypoints((msg_mission_ack)
 msg);
  154:                                    state = waypointStates.IDLE;
  155:                                    return true;
  156:                            }
  157:                            break;
  158:                    }
  159:
  160:                    if (msg.msgid == msg_mission_item_reached.MAVLINK_MSG_ID_MISSION_I
TEM_REACHED) {
  161:                            onWaypointReached(((msg_mission_item_reached) msg).seq);
  162:                            return true;
  163:                    }
  164:                    if (msg.msgid == msg_mission_current.MAVLINK_MSG_ID_MISSION_CURREN
T) {
  165:                            onCurrentWaypointUpdate(((msg_mission_current) msg).seq);
  166:                            return true;
  167:                    }
  168:                    return false;
  169:            }
  170: }
```

```
    1: package com.droidplanner.drone.variables.mission.waypoints;
    2:
    3: import android.content.Context;
    4:
    5: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
    6: import com.MAVLink.Messages.enums.MAV_CMD;
    7: import com.droidplanner.R;
    8: import com.droidplanner.drone.variables.mission.MissionItem;
    9: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   10: import com.droidplanner.fragments.markers.helpers.MarkerWithText;
   11: import com.droidplanner.fragments.mission.MissionDetailFragment;
   12: import com.droidplanner.fragments.mission.MissionLandFragment;
   13: import com.google.android.gms.maps.model.BitmapDescriptor;
   14: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   15:
   16: public class Land extends SpatialCoordItem implements MarkerSource {
   17:
   18:         private double yawAngle;
   19:
   20:         public Land(MissionItem item) {
   21:                 super(item);
   22:         }
   23:
   24:         @Override
   25:         protected BitmapDescriptor getIcon(Context context) {
   26:                 return BitmapDescriptorFactory.fromBitmap(MarkerWithText
   27:                                 .getMarkerWithTextAndDetail(R.drawable.ic_wp_map,
"text",
   28:                                                 "detail", context));
   29:         }
   30:
   31:         @Override
   32:         public MissionDetailFragment getDetailFragment() {
   33:                 MissionDetailFragment fragment = new MissionLandFragment();
   34:                 fragment.setItem(this);
   35:                 return fragment;
   36:         }
   37:
   38:         @Override
   39:         public msg_mission_item packMissionItem() {
   40:                 msg_mission_item mavMsg = super.packMissionItem();
   41:                 mavMsg.command = MAV_CMD.MAV_CMD_NAV_LAND;
   42:                 mavMsg.param4 = (float) getYawAngle();
   43:                 return mavMsg;
   44:         }
   45:
   46:         @Override
   47:         public void unpackMAVMessage(msg_mission_item mavMsg) {
   48:                 super.unpackMAVMessage(mavMsg);
   49:                 setYawAngle(mavMsg.param4);
   50:         }
   51:
   52:         public double getYawAngle() {
   53:                 return yawAngle;
   54:         }
   55:
   56:         public void setYawAngle(double yawAngle) {
   57:                 this.yawAngle = yawAngle;
   58:         }
   59:
   60: }
```

```
    1: package com.droidplanner.drone.variables.mission.waypoints;
    2:
    3: import android.content.Context;
    4:
    5: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
    6: import com.droidplanner.R;
    7: import com.droidplanner.drone.variables.mission.MissionItem;
    8: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
    9: import com.droidplanner.fragments.markers.helpers.MarkerWithText;
   10: import com.droidplanner.helpers.units.Altitude;
   11: import com.google.android.gms.maps.model.BitmapDescriptor;
   12: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   13: import com.google.android.gms.maps.model.LatLng;
   14:
   15: public abstract class Loiter extends SpatialCoordItem implements MarkerSource {
   16:         private double orbitalRadius;
   17:         private double yawAngle;
   18:         private boolean orbitCCW;
   19:
   20:         public Loiter(MissionItem item) {
   21:                 super(item);
   22:         }
   23:
   24:         public Loiter(LatLng coord, Altitude altitude) {
   25:                 super(coord, altitude);
   26:         }
   27:
   28:         public void setOrbitalRadius(double radius) {
   29:                 this.orbitalRadius = radius;
   30:         }
   31:
   32:         public double getOrbitalRadius(){
   33:                 return this.orbitalRadius;
   34:         }
   35:
   36:         public boolean isOrbitCCW() {
   37:                 return orbitCCW;
   38:         }
   39:
   40:         public void setOrbitCCW(boolean orbitCCW) {
   41:                 this.orbitCCW = orbitCCW;
   42:         }
   43:
   44:
   45:         public double getYawAngle() {
   46:                 return yawAngle;
   47:         }
   48:
   49:         public void setYawAngle(double yawAngle) {
   50:                 this.yawAngle = yawAngle;
   51:         }
   52:
   53:         @Override
   54:         public msg_mission_item packMissionItem() {
   55:                 msg_mission_item mavMsg = super.packMissionItem();
   56:                 mavMsg.param3 = (float) (isOrbitCCW()?getOrbitalRadius()*-1.0:getO
rbitalRadius());
   57:                 mavMsg.param4 = (float) getYawAngle();
   58:                 return mavMsg;
   59:         }
   60:
   61:         @Override
   62:         public void unpackMAVMessage(msg_mission_item mavMsg) {
   63:                 super.unpackMAVMessage(mavMsg);
   64:                 setOrbitCCW(mavMsg.param3<0);
   65:                 setOrbitalRadius(Math.abs(mavMsg.param3));
   66:         }
   67:
   68:         @Override
   69:         protected BitmapDescriptor getIcon(Context context) {
   70:                 return BitmapDescriptorFactory.fromBitmap(MarkerWithText
   71:                                 .getMarkerWithTextAndDetail(R.drawable.ic_wp_loite
r, "text",
   72:                                                 "detail", context));
   73:         }
   74: }
```

```
 1: package com.droidplanner.drone.variables.mission.waypoints;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
 4: import com.droidplanner.drone.variables.mission.MissionItem;
 5: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 6: import com.droidplanner.fragments.mission.MissionDetailFragment;
 7: import com.droidplanner.fragments.mission.MissionLoiterFragment;
 8:
 9: public class LoiterInfinite extends Loiter implements MarkerSource {
10:
11:         public LoiterInfinite(MissionItem item) {
12:                 super(item);
13:         }
14:
15:         @Override
16:         public MissionDetailFragment getDetailFragment() {
17:                 MissionDetailFragment fragment = new MissionLoiterFragment();
18:                 fragment.setItem(this);
19:                 return fragment;
20:         }
21:
22:         @Override
23:         public msg_mission_item packMissionItem() {
24:                 return super.packMissionItem();
25:         }
26:
27:         @Override
28:         public void unpackMAVMessage(msg_mission_item mavMsg) {
29:                 super.unpackMAVMessage(mavMsg);
30:         }
31:
32: }
```

```
 1: package com.droidplanner.drone.variables.mission.waypoints;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
 4: import com.MAVLink.Messages.enums.MAV_CMD;
 5: import com.droidplanner.drone.variables.mission.MissionItem;
 6: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 7: import com.droidplanner.fragments.mission.MissionDetailFragment;
 8: import com.droidplanner.fragments.mission.MissionLoiterTFragment;
 9:
10: public class LoiterTime extends Loiter implements MarkerSource {
11:         double time;
12:
13:         public LoiterTime(MissionItem item) {
14:                 super(item);
15:         }
16:
17:         public double getTime() {
18:                 return time;
19:         }
20:
21:         public void setTime(double time) {
22:                 this.time = time;
23:         }
24:
25:         @Override
26:         public MissionDetailFragment getDetailFragment() {
27:                 MissionDetailFragment fragment = new MissionLoiterTFragment();
28:                 fragment.setItem(this);
29:                 return fragment;
30:         }
31:
32:         @Override
33:         public msg_mission_item packMissionItem() {
34:                 msg_mission_item mavMsg = super.packMissionItem();
35:                 mavMsg.command = MAV_CMD.MAV_CMD_NAV_LOITER_TIME;
36:                 mavMsg.param1 = (float) getTime();
37:                 return mavMsg;
38:         }
39:
40:         @Override
41:         public void unpackMAVMessage(msg_mission_item mavMsg) {
42:                 super.unpackMAVMessage(mavMsg);
43:                 setTime(mavMsg.param1);
44:         }
45:
46:
47: }
```

```
 1: package com.droidplanner.drone.variables.mission.waypoints;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
 4: import com.MAVLink.Messages.enums.MAV_CMD;
 5: import com.droidplanner.drone.variables.mission.MissionItem;
 6: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 7: import com.droidplanner.fragments.mission.MissionDetailFragment;
 8: import com.droidplanner.fragments.mission.MissionLoiterNFragment;
 9:
10: public class LoiterTurns extends Loiter implements MarkerSource {
11:         private int turns;
12:
13:         public LoiterTurns(MissionItem item) {
14:                 super(item);
15:         }
16:
17:         public int getTurns() {
18:                 return turns;
19:         }
20:
21:         public void setTurns(int turns) {
22:                 this.turns = turns;
23:         }
24:
25:         @Override
26:         public MissionDetailFragment getDetailFragment() {
27:                 MissionDetailFragment fragment = new MissionLoiterNFragment();
28:                 fragment.setItem(this);
29:                 return fragment;
30:         }
31:
32:         @Override
33:         public msg_mission_item packMissionItem() {
34:                 msg_mission_item mavMsg = super.packMissionItem();
35:                 mavMsg.command = MAV_CMD.MAV_CMD_NAV_LOITER_TURNS;
36:                 mavMsg.param1 = (float) getTurns();
37:                 return mavMsg;
38:         }
39:
40:         @Override
41:         public void unpackMAVMessage(msg_mission_item mavMsg) {
42:                 super.unpackMAVMessage(mavMsg);
43:                 setTurns((int) mavMsg.param1);
44:         }
45: }
```

```
    1: package com.droidplanner.drone.variables.mission.waypoints;
    2:
    3: import java.util.List;
    4:
    5: import android.content.Context;
    6:
    7: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
    8: import com.droidplanner.R;
    9: import com.droidplanner.drone.variables.mission.MissionItem;
   10: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   11: import com.droidplanner.fragments.markers.helpers.MarkerWithText;
   12: import com.droidplanner.fragments.mission.MissionDetailFragment;
   13: import com.droidplanner.fragments.mission.MissionRegionOfInterestFragment;
   14: import com.google.android.gms.maps.model.BitmapDescriptor;
   15: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   16: import com.google.android.gms.maps.model.LatLng;
   17:
   18: public class RegionOfInterest extends SpatialCoordItem implements MarkerSource{
   19:
   20:         public RegionOfInterest(MissionItem item) {
   21:                 super(item);
   22:         }
   23:
   24:         @Override
   25:         protected BitmapDescriptor getIcon(Context context) {
   26:                 return BitmapDescriptorFactory.fromBitmap(MarkerWithText
   27:                                 .getMarkerWithTextAndDetail(R.drawable.ic_wp_map,
"text",
   28:                                                 "detail", context));
   29:         }
   30:
   31:         @Override
   32:         public List<LatLng> getPath() throws Exception {
   33:                 throw new Exception();
   34:         }
   35:
   36:         @Override
   37:         public MissionDetailFragment getDetailFragment() {
   38:                 MissionDetailFragment fragment = new MissionRegionOfInterestFragme
nt();
   39:                 fragment.setItem(this);
   40:                 return fragment;
   41:         }
   42:
   43:         @Override
   44:         public msg_mission_item packMissionItem() {
   45:                 // TODO Auto-generated method stub
   46:                 return super.packMissionItem();
   47:         }
   48:
   49:         @Override
   50:         public void unpackMAVMessage(msg_mission_item mavMsg) {
   51:                 // TODO Auto-generated method stub
   52:                 super.unpackMAVMessage(mavMsg);
   53:         }
   54:
   55:         /*
   56:         private static String getRoiDetail(GenericWaypoint wp, Context context) {
   57:                 if (wp.getParam1() == MAV_ROI.MAV_ROI_WPNEXT)
   58:                         return context.getString(R.string.next);
   59:                 else if (wp.getParam1() == MAV_ROI.MAV_ROI_TARGET)
   60:                         return String.format(Locale.US,"wp#%.0f", wp.getParam2());
   61:                 else if (wp.getParam1() == MAV_ROI.MAV_ROI_TARGET)
   62:                         return String.format(Locale.US,"tg#%.0f", wp.getParam2());
   63:                 else
   64:                         return "";
   65:         }
   66:         */
   67: }
```

```java
package com.droidplanner.drone.variables.mission.waypoints;

import java.util.ArrayList;
import java.util.List;

import android.content.Context;

import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
import com.droidplanner.drone.variables.mission.MissionItem;
import com.droidplanner.fragments.markers.GenericMarker;
import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
import com.droidplanner.helpers.units.Altitude;
import com.google.android.gms.maps.model.BitmapDescriptor;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

/**
 * Generic Mission item with Spatial Coordinates
 *
 */
public abstract class SpatialCoordItem extends MissionItem implements
                MarkerSource {
        protected abstract BitmapDescriptor getIcon(Context context);

        LatLng coordinate;
        Altitude altitude;

        public SpatialCoordItem(LatLng coord, Altitude altitude) {
                this.coordinate = coord;
                this.altitude = altitude;
        }

        public SpatialCoordItem(MissionItem item) {
                if (item instanceof SpatialCoordItem) {
                        coordinate = ((SpatialCoordItem) item).getCoordinate();
                        altitude = ((SpatialCoordItem) item).getAltitude();
                } else {
                        coordinate = new LatLng(0, 0);
                        altitude = new Altitude(0);
                }
        }

        @Override
        public MarkerOptions build(Context context) {
                return GenericMarker.build(coordinate).icon(getIcon(context));
        }

        @Override
        public void update(Marker marker, Context context) {
                marker.setPosition(coordinate);
                marker.setIcon(getIcon(context));
        }

        @Override
        public List<MarkerSource> getMarkers() throws Exception {
                ArrayList<MarkerSource> marker = new ArrayList<MarkerSource>();
                marker.add(this);
                return marker;
        }

        @Override
        public List<LatLng> getPath() throws Exception {
                ArrayList<LatLng> points = new ArrayList<LatLng>();
                points.add(coordinate);
                return points;
        }

        public void setCoordinate(LatLng position) {
                coordinate = position;
        }

        public LatLng getCoordinate() {
                return coordinate;
        }

        public Altitude getAltitude() {
                return altitude;
        }

        public void setAltitude(Altitude altitude) {
                this.altitude = altitude;
        }

        @Override
        public msg_mission_item packMissionItem() {
                msg_mission_item mavMsg = new msg_mission_item();
                mavMsg.autocontinue = 1;
                mavMsg.target_component = 1;
                mavMsg.target_system = 1;
                mavMsg.x = (float) getCoordinate().latitude;
                mavMsg.y = (float) getCoordinate().longitude;
                mavMsg.z = (float) getAltitude().valueInMeters();
//              mavMsg.compid =
                return mavMsg;
        }

        @Override
        public void unpackMAVMessage(msg_mission_item mavMsg) {
                LatLng coord = new LatLng(mavMsg.x,mavMsg.y);
                Altitude alt = new Altitude(mavMsg.z);
                setCoordinate(coord);
                setAltitude(alt);
        }
}
```

```
    1: package com.droidplanner.drone.variables.mission.waypoints;
    2:
    3: import android.content.Context;
    4:
    5: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
    6: import com.MAVLink.Messages.enums.MAV_CMD;
    7: import com.droidplanner.R;
    8: import com.droidplanner.drone.variables.mission.MissionItem;
    9: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   10: import com.droidplanner.fragments.markers.helpers.MarkerWithText;
   11: import com.droidplanner.fragments.mission.MissionDetailFragment;
   12: import com.droidplanner.fragments.mission.MissionTakeoffFragment;
   13: import com.google.android.gms.maps.model.BitmapDescriptor;
   14: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   15:
   16: public class Takeoff extends SpatialCoordItem implements MarkerSource {
   17:
   18:         private double yawAngle;
   19:         private double minPitch;
   20:
   21:         public Takeoff(MissionItem item) {
   22:                 super(item);
   23:         }
   24:
   25:         @Override
   26:         protected BitmapDescriptor getIcon(Context context) {
   27:                 return BitmapDescriptorFactory.fromBitmap(MarkerWithText
   28:                                 .getMarkerWithTextAndDetail(R.drawable.ic_wp_map,
"text",
   29:                                                 "detail", context));
   30:         }
   31:
   32:         @Override
   33:         public MissionDetailFragment getDetailFragment() {
   34:                 MissionDetailFragment fragment = new MissionTakeoffFragment();
   35:                 fragment.setItem(this);
   36:                 return fragment;
   37:         }
   38:
   39:         @Override
   40:         public msg_mission_item packMissionItem() {
   41:                 msg_mission_item mavMsg = super.packMissionItem();
   42:                 mavMsg.command = MAV_CMD.MAV_CMD_NAV_TAKEOFF;
   43:                 mavMsg.param1 = (float) getMinPitch();
   44:                 mavMsg.param4 = (float) getYawAngle();
   45:                 return mavMsg;
   46:         }
   47:
   48:         @Override
   49:         public void unpackMAVMessage(msg_mission_item mavMsg) {
   50:                 super.unpackMAVMessage(mavMsg);
   51:                 setMinPitch(mavMsg.param1);
   52:                 setYawAngle(mavMsg.param4);
   53:         }
   54:
   55:         public double getYawAngle() {
   56:                 return yawAngle;
   57:         }
   58:
   59:         public void setYawAngle(double yawAngle) {
   60:                 this.yawAngle = yawAngle;
   61:         }
   62:
   63:         public double getMinPitch() {
   64:                 return minPitch;
   65:         }
   66:
   67:         public void setMinPitch(double minPitch) {
   68:                 this.minPitch = minPitch;
   69:         }
   70: }
```

```
    1: package com.droidplanner.drone.variables.mission.waypoints;
    2:
    3: import android.content.Context;
    4:
    5: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
    6: import com.MAVLink.Messages.enums.MAV_CMD;
    7: import com.droidplanner.R;
    8: import com.droidplanner.drone.variables.mission.MissionItem;
    9: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   10: import com.droidplanner.fragments.markers.helpers.MarkerWithText;
   11: import com.droidplanner.fragments.mission.MissionDetailFragment;
   12: import com.droidplanner.fragments.mission.MissionWaypointFragment;
   13: import com.droidplanner.helpers.units.Altitude;
   14: import com.google.android.gms.maps.model.BitmapDescriptor;
   15: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   16: import com.google.android.gms.maps.model.LatLng;
   17:
   18: public class Waypoint extends SpatialCoordItem implements MarkerSource {
   19:         private double delay;
   20:         private double acceptanceRadius;
   21:         private double yawAngle;
   22:         private double orbitalRadius;
   23:         private boolean orbitCCW;
   24:
   25:         public Waypoint(MissionItem item) {
   26:                 super(item);
   27:         }
   28:
   29:         public Waypoint(LatLng point, Altitude defaultAlt) {
   30:                 super(point, defaultAlt);
   31:         }
   32:
   33:         @Override
   34:         protected BitmapDescriptor getIcon(Context context) {
   35:                 return BitmapDescriptorFactory.fromBitmap(MarkerWithText
   36:                                 .getMarkerWithTextAndDetail(R.drawable.ic_wp_map,
"text",
   37:                                                 "detail", context));
   38:         }
   39:
   40:         @Override
   41:         public MissionDetailFragment getDetailFragment() {
   42:                 MissionDetailFragment fragment = new MissionWaypointFragment();
   43:                 fragment.setItem(this);
   44:                 return fragment;
   45:         }
   46:
   47:         @Override
   48:         public msg_mission_item packMissionItem() {
   49:                 msg_mission_item mavMsg = super.packMissionItem();
   50:                 mavMsg.command = MAV_CMD.MAV_CMD_NAV_WAYPOINT;
   51:                 mavMsg.param1 = (float) getDelay();
   52:                 mavMsg.param2 = (float) getAcceptanceRadius();
   53:                 mavMsg.param3 = (float) (isOrbitCCW()?getOrbitalRadius()*-1.0:getO
rbitalRadius());
   54:                 mavMsg.param4 = (float) getYawAngle();
   55:                 return mavMsg;
   56:         }
   57:
   58:         @Override
   59:         public void unpackMAVMessage(msg_mission_item mavMsg) {
   60:                 super.unpackMAVMessage(mavMsg);
   61:                 setDelay(mavMsg.param1);
   62:                 setAcceptanceRadius(mavMsg.param2);
   63:                 setOrbitCCW(mavMsg.param3<0);
   64:                 setOrbitalRadius(Math.abs(mavMsg.param3));
   65:                 setYawAngle(mavMsg.param4);
   66:         }
   67:
   68:         public double getDelay() {
   69:                 return delay;
   70:         }
   71:
   72:         public void setDelay(double delay) {
   73:                 this.delay = delay;
   74:         }
   75:
   76:         public double getAcceptanceRadius() {
   77:                 return acceptanceRadius;
   78:         }
   79:
   80:         public void setAcceptanceRadius(double acceptanceRadius) {
   81:                 this.acceptanceRadius = acceptanceRadius;
   82:         }
   83:
   84:         public double getYawAngle() {
   85:                 return yawAngle;
   86:         }
   87:
   88:         public void setYawAngle(double yawAngle) {
   89:                 this.yawAngle = yawAngle;
   90:         }
   91:
   92:         public double getOrbitalRadius() {
   93:                 return orbitalRadius;
   94:         }
   95:
   96:         public void setOrbitalRadius(double orbitalRadius) {
   97:                 this.orbitalRadius = orbitalRadius;
   98:         }
   99:
  100:         public boolean isOrbitCCW() {
  101:                 return orbitCCW;
  102:         }
  103:
  104:         public void setOrbitCCW(boolean orbitCCW) {
  105:                 this.orbitCCW = orbitCCW;
  106:         }
  107:
  108: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5:
 6: public class MissionStats extends DroneVariable{
 7:         private double distanceToWp = 0;
 8:         private short goingForWaypoint = -1;
 9:
10:         public MissionStats(Drone myDrone) {
11:                 super(myDrone);
12:         }
13:
14:         public void setDistanceToWp(double disttowp) {
15:                 this.distanceToWp  = disttowp;
16:         }
17:
18:         public void setWpno(short seq) {
19:                 goingForWaypoint  = seq;
20:
21:         }
22:
23: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5:
 6: public class Navigation extends DroneVariable {
 7:
 8:         private double nav_pitch;
 9:         private double nav_roll;
10:         private double nav_bearing;
11:
12:         public Navigation(Drone myDrone) {
13:                 super(myDrone);
14:         }
15:
16:         public void setNavPitchRollYaw(float nav_pitch, float nav_roll,
17:                         short nav_bearing) {
18:                 this.nav_pitch = (double) nav_pitch;
19:                 this.nav_roll = (double) nav_roll;
20:                 this.nav_bearing = (double) nav_bearing;
21:                 notifyNewNavigationData();
22:         }
23:
24:         private void notifyNewNavigationData() {
25:                 if (myDrone.tuningDataListner != null) {
26:                         myDrone.tuningDataListner.onNewNavigationData();
27:                 }
28:         }
29:
30:         public double getNavPitch() {
31:                 return nav_pitch;
32:         }
33:
34:         public double getNavRoll() {
35:                 return nav_roll;
36:         }
37:
38:         public double getNavBearing() {
39:                 return nav_bearing;
40:         }
41:
42: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5:
 6: public class Orientation extends DroneVariable {
 7:         private double roll = 0;
 8:         private double pitch = 0;
 9:         private double yaw = 0;
10:
11:         public Orientation(Drone myDrone) {
12:                 super(myDrone);
13:         }
14:
15:         public double getRoll() {
16:                 return roll;
17:         }
18:
19:         public double getPitch() {
20:                 return pitch;
21:         }
22:
23:         public double getYaw() {
24:                 return yaw;
25:         }
26:
27:         public void setRollPitchYaw(double roll, double pitch, double yaw) {
28:                 this.roll = roll;
29:                 this.pitch = pitch;
30:                 this.yaw = yaw;
31:                 myDrone.onOrientationUpdate();
32:                 notifyNewOrientationData();
33:         }
34:
35:         private void notifyNewOrientationData() {
36:                 if (myDrone.tuningDataListner != null) {
37:                         myDrone.tuningDataListner.onNewOrientationData();
38:                 }
39:         }
40:
41: }
```

```java
package com.droidplanner.drone.variables;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.*;

import android.content.Context;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;
import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.ardupilotmega.msg_param_value;
import com.droidplanner.MAVLink.MavLinkParameters;
import com.droidplanner.R;
import com.droidplanner.drone.Drone;
import com.droidplanner.drone.DroneInterfaces;
import com.droidplanner.drone.DroneVariable;
import com.droidplanner.file.DirectoryPath;
import com.droidplanner.file.IO.ParameterMetadataMap;
import com.droidplanner.file.IO.ParameterMetadataMapReader;
import com.droidplanner.parameters.Parameter;
import com.droidplanner.parameters.ParameterMetadata;

/**
 * Class to manage the communication of parameters to the MAV.
 *
 * Should be initialized with a MAVLink Object, so the manager can send messages
 * via the MAV link. The function processMessage must be called with every new
 * MAV Message.
 *
 */
public class Parameters extends DroneVariable {

    private static final String PARAMETERMETADATA_PATH = "Parameters/ParameterMetaData.xml";

    private List<Parameter> parameters = new ArrayList<Parameter>();
    private ParameterMetadataMap metadataMap;

        public DroneInterfaces.OnParameterManagerListner parameterListner;

        public Parameters(Drone myDrone) {
                super(myDrone);
        }

        public void getAllParameters() {
                parameters.clear();
                if(parameterListner!=null)
                        parameterListner.onBeginReceivingParameters();

                MavLinkParameters.requestParametersList(myDrone);
        }

        /**
         * Try to process a Mavlink message if it is a parameter related message
         *
         * @param msg
         *            Mavlink message to process
         * @return Returns true if the message has been processed
         */
        public boolean processMessage(MAVLinkMessage msg) {
                if (msg.msgid == msg_param_value.MAVLINK_MSG_ID_PARAM_VALUE) {
                        processReceivedParam((msg_param_value) msg);
                        return true;
                }
                return false;
        }

        private void processReceivedParam(msg_param_value m_value) {

                // collect params in parameter list
                Parameter param = new Parameter(m_value);
                parameters.add(param);

                // update listener
                if(parameterListner!=null)
                        parameterListner.onParameterReceived(param, m_value.param_index, m_value.param_count);

                // last param? notify listener w/ params
                if (m_value.param_index == m_value.param_count - 1) {
                        if(parameterListner!=null)
                                parameterListner.onEndReceivingParameters(parameters);
                }
        }

        public void sendParameter(Parameter parameter) {
                MavLinkParameters.sendParameter(myDrone, parameter);
        }

    public void notifyParameterMetadataChanged() {
        if(parameterListner != null)
            parameterListner.onParamterMetaDataChanged();
    }

    public ParameterMetadata getMetadata(String name) {
        return (metadataMap == null) ? null : metadataMap.get(name);
    }

    public void loadMetadata(Context context, String metadataType) {
        metadataMap = null;

        // use metadata type from prefs if not specified, bail if none
        if(metadataType == null) {
            final SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
            metadataType = prefs.getString("pref_param_metadata", null);
        }
        if(metadataType == null || metadataType.equals(context.getString(R.string.none)))
            return;

        try {
            // use user supplied file in ~/Parameters if available, else fallback to asset from resources
            final InputStream inputStream;
            final File file = new File(DirectoryPath.getDroidPlannerPath() + PARAMETERMETADATA_PATH);
            if(file.exists()) {
                // load from file
                inputStream = new FileInputStream(file);
            } else {
                // load from resource
                inputStream = context.getAssets().open(PARAMETERMETADATA_PATH);
            }

            // parse
            metadataMap = ParameterMetadataMapReader.open(inputStream, metadataType);

        } catch (Exception ex) {
            // nop

```

```
127:         }
128:     }
129:
130:         public void ReadParameter(String name) {
131:                 MavLinkParameters.readParameter(myDrone, name);
132:         }
133: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_rc_channels_raw;
 4: import com.MAVLink.Messages.ardupilotmega.msg_servo_output_raw;
 5: import com.droidplanner.drone.Drone;
 6: import com.droidplanner.drone.DroneInterfaces.OnRcDataChangedListner;
 7: import com.droidplanner.drone.DroneVariable;
 8:
 9: public class RC extends DroneVariable {
10:
11:         public OnRcDataChangedListner listner;
12:
13:
14:         public int in[] = new int[8];
15:         public int out[] = new int[8];
16:
17:         public RC(Drone myDrone) {
18:                 super(myDrone);
19:         }
20:
21:         public void setListner(OnRcDataChangedListner listner) {
22:                 this.listner = listner;
23:         }
24:
25:         public void setRcInputValues(msg_rc_channels_raw msg) {
26:                 in[0] = msg.chan1_raw;
27:                 in[1] = msg.chan2_raw;
28:                 in[2] = msg.chan3_raw;
29:                 in[3] = msg.chan4_raw;
30:                 in[4] = msg.chan5_raw;
31:                 in[5] = msg.chan6_raw;
32:                 in[6] = msg.chan7_raw;
33:                 in[7] = msg.chan8_raw;
34:                 if (listner!=null) {
35:                         listner.onNewInputRcData();
36:                 }
37:         }
38:
39:         public void setRcOutputValues(msg_servo_output_raw msg) {
40:                 out[0] = msg.servo1_raw;
41:                 out[1] = msg.servo2_raw;
42:                 out[2] = msg.servo3_raw;
43:                 out[3] = msg.servo4_raw;
44:                 out[4] = msg.servo5_raw;
45:                 out[5] = msg.servo6_raw;
46:                 out[6] = msg.servo7_raw;
47:                 out[7] = msg.servo8_raw;
48:                 if (listner!=null) {
49:                         listner.onNewOutputRcData();
50:                 }
51:         }
52:
53:
54:
55: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.droidplanner.drone.Drone;
 4: import com.droidplanner.drone.DroneVariable;
 5:
 6: public class Speed extends DroneVariable {
 7:         private double verticalSpeed = 0;
 8:         private double groundSpeed = 0;
 9:         private double airSpeed = 0;
10:         private double targetSpeed = 0;
11:
12:         public Speed(Drone myDrone) {
13:                 super(myDrone);
14:         }
15:
16:         public double getVerticalSpeed() {
17:                 return verticalSpeed;
18:         }
19:
20:         public double getGroundSpeed() {
21:                 return groundSpeed;
22:         }
23:
24:         public double getAirSpeed() {
25:                 return airSpeed;
26:         }
27:
28:         public double getTargetSpeed() {
29:                 return targetSpeed;
30:         }
31:
32:         public void setSpeedError(double aspd_error) {
33:                 targetSpeed = aspd_error + airSpeed;
34:         }
35:
36:         public void setGroundAndAirSpeeds(double groundSpeed, double airSpeed,
37:                         double climb) {
38:                 this.groundSpeed = groundSpeed;
39:                 this.airSpeed = airSpeed;
40:                 this.verticalSpeed = climb;
41:         }
42: }
```

```
    1: package com.droidplanner.drone.variables;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.util.Log;
    7:
    8: import com.MAVLink.Messages.ApmModes;
    9: import com.droidplanner.MAVLink.MavLinkModes;
   10: import com.droidplanner.drone.Drone;
   11: import com.droidplanner.drone.DroneInterfaces.OnStateListner;
   12: import com.droidplanner.drone.DroneVariable;
   13:
   14: public class State extends DroneVariable {
   15:         private boolean failsafe = false;
   16:         private boolean armed = false;
   17:         private boolean isFlying = false;
   18:         private ApmModes mode = ApmModes.UNKNOWN;
   19:         public List<OnStateListner> stateListner = new ArrayList<OnStateListner>()
;
   20:
   21:         public State(Drone myDrone) {
   22:                 super(myDrone);
   23:         }
   24:
   25:         public boolean isFailsafe() {
   26:                 return failsafe;
   27:         }
   28:
   29:         public boolean isArmed() {
   30:                 return armed;
   31:         }
   32:
   33:         public boolean isFlying() {
   34:                 return isFlying;
   35:         }
   36:
   37:         public ApmModes getMode() {
   38:                 return mode;
   39:         }
   40:
   41:         public void setIsFlying(boolean newState) {
   42:                 if (newState != isFlying) {
   43:                         isFlying = newState;
   44:                         notifyFlightStateChanged();
   45:                 }
   46:         }
   47:
   48:         public void setFailsafe(boolean newFailsafe) {
   49:                 if(this.failsafe!=newFailsafe){
   50:                         this.failsafe=newFailsafe;
   51:                         notifyFailsafeChanged();
   52:                 }
   53:         }
   54:
   55:         public void setArmed(boolean newState) {
   56:                 if (this.armed != newState) {
   57:                         this.armed = newState;
   58:                         notifyArmChanged();
   59:                 }
   60:         }
   61:
   62:         public void setMode(ApmModes mode) {
   63:                 if (this.mode != mode) {
   64:                         this.mode = mode;
   65:                         myDrone.tts.speakMode(mode);
   66:                         myDrone.notifyModeChanged();
   67:
   68:                         if (getMode() != ApmModes.ROTOR_GUIDED) {
   69:                                 myDrone.guidedPoint.invalidateCoord();
   70:                         }
   71:                 }
   72:         }
   73:
   74:         public void changeFlightMode(ApmModes mode) {
   75:                 Log.d("MODE", "mode " + mode.getName());
   76:                 if (ApmModes.isValid(mode)) {
   77:                         Log.d("MODE", "mode " + mode.getName() + " is valid");
   78:                         MavLinkModes.changeFlightMode(myDrone, mode);
   79:                 }
   80:         }
   81:
   82:         public void addFlightStateListner(OnStateListner listner) {
   83:                 stateListner.add(listner);
   84:         }
   85:
   86:         public void removeFlightStateListner(OnStateListner listner) {
   87:                 if (stateListner.contains(listner)) {
   88:                         stateListner.remove(listner);
   89:                 }
   90:         }
   91:
   92:         private void notifyFlightStateChanged() {
   93:                 for (OnStateListner listner : stateListner) {
   94:                         listner.onFlightStateChanged();
   95:                 }
   96:         }
   97:
   98:         private void notifyFailsafeChanged() {
   99:                 for (OnStateListner listner : stateListner) {
  100:                         listner.onFailsafeChanged();
  101:                 }
  102:         }
  103:
  104:         private void notifyArmChanged() {
  105:                 myDrone.tts.speakArmedState(armed);
  106:                 for (OnStateListner listner : stateListner) {
  107:                         listner.onArmChanged();
  108:                 }
  109:         }
  110:
  111: }
```

```
 1: package com.droidplanner.drone.variables;
 2:
 3: import com.MAVLink.Messages.enums.MAV_TYPE;
 4: import com.droidplanner.drone.Drone;
 5: import com.droidplanner.drone.DroneVariable;
 6:
 7: public class Type extends DroneVariable {
 8:         private int type = MAV_TYPE.MAV_TYPE_FIXED_WING;
 9:
10:         public Type(Drone myDrone) {
11:                 super(myDrone);
12:         }
13:
14:         public int getType() {
15:                 return type;
16:         }
17:
18:         public void setType(int type) {
19:                 if (this.type != type) {
20:                         this.type = type;
21:                         myDrone.notifyTypeChanged();
22:                 }
23:         }
24: }
```

```
 1: package com.droidplanner.file;
 2:
 3: import android.os.Environment;
 4:
 5: public class DirectoryPath {
 6:
 7:     static public String getDroidPlannerPath() {
 8:                 String root = Environment.getExternalStorageDirectory().getPath();
 9:                 return (root + "/DroidPlanner/");
10:         }
11:
12:         static public String getParametersPath() {
13:                 return getDroidPlannerPath() + "/Parameters/";
14:         }
15:
16:         static public String getWaypointsPath() {
17:                 return getDroidPlannerPath() + "/Waypoints/";
18:         }
19:
20:         static public String getGCPPath() {
21:                 return getDroidPlannerPath() + "/GCP/";
22:         }
23:
24:         static public String getTLogPath() {
25:                 return getDroidPlannerPath() + "/Logs/";
26:         }
27:
28:         static public String getMapsPath() {
29:                 return getDroidPlannerPath() + "/Maps/";
30:         }
31:
32:         public static String getCameraInfoPath() {
33:                 return getDroidPlannerPath() + "/CameraInfo/";
34:         }
35:
36: }
```

```
    1: package com.droidplanner.file;
    2:
    3: import java.io.File;
    4: import java.io.FilenameFilter;
    5:
    6: public class FileList {
    7:
    8:         static public String[] getWaypointFileList() {
    9:                 FilenameFilter filter = new FilenameFilter() {
   10:                         public boolean accept(File dir, String filename) {
   11:                                 return filename.contains(".txt");
   12:                         }
   13:                 };
   14:                 return getFileList(DirectoryPath.getWaypointsPath(), filter);
   15:         }
   16:
   17:         public static String[] getParametersFileList() {
   18:                 FilenameFilter filter = new FilenameFilter() {
   19:                         public boolean accept(File dir, String filename) {
   20:                                 return filename.contains(".param");
   21:                         }
   22:                 };
   23:                 return getFileList(DirectoryPath.getParametersPath(), filter);
   24:         }
   25:
   26:         static public String[] getKMZFileList() {
   27:                 FilenameFilter filter = new FilenameFilter() {
   28:                         public boolean accept(File dir, String filename) {
   29:                                 return filename.contains(".kml") || filename.conta
ins(".kmz");
   30:                         }
   31:                 };
   32:                 return getFileList(DirectoryPath.getGCPPath(), filter);
   33:         }
   34:
   35:         public static String[] getCameraInfoFileList() {
   36:                 FilenameFilter filter = new FilenameFilter() {
   37:                         public boolean accept(File dir, String filename) {
   38:                                 return filename.contains(".xml");
   39:                         }
   40:                 };
   41:                 return getFileList(DirectoryPath.getCameraInfoPath(), filter);
   42:         }
   43:
   44:         static public String[] getFileList(String path, FilenameFilter filter) {
   45:                 File mPath = new File(path);
   46:                 try {
   47:                         mPath.mkdirs();
   48:                         if (mPath.exists()) {
   49:                                 return mPath.list(filter);
   50:                         }
   51:                 } catch (SecurityException e) {
   52:                 }
   53:                 return new String[0];
   54:         }
   55:
   56: }
```

```
 1: package com.droidplanner.file;
 2:
 3: import java.text.SimpleDateFormat;
 4: import java.util.Date;
 5: import java.util.Locale;
 6:
 7: import android.os.Environment;
 8:
 9: public class FileManager {
10:
11:         /**
12:          * Timestamp for logs in the Mission Planner Format
13:          */
14:         static public String getTimeStamp() {
15:                 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH-mm-ss",
16:                                 Locale.US);
17:                 String timeStamp = sdf.format(new Date());
18:                 return timeStamp;
19:         }
20:
21:         public static boolean isExternalStorageAvaliable() {
22:                 String state = Environment.getExternalStorageState();
23:                 if (Environment.MEDIA_MOUNTED.equals(state)) {
24:                         return true;
25:                 }
26:                 return false;
27:         }
28:
29: }
```

```
    1: package com.droidplanner.file;
    2:
    3: import java.io.BufferedOutputStream;
    4: import java.io.File;
    5: import java.io.FileNotFoundException;
    6: import java.io.FileOutputStream;
    7: import java.io.IOException;
    8:
    9: public class FileStream {
   10:         public static FileOutputStream getParameterFileStream()
   11:                         throws FileNotFoundException {
   12:                 File myDir = new File(DirectoryPath.getParametersPath());
   13:                 myDir.mkdirs();
   14:                 File file = new File(myDir, "Parameters-" + FileManager.getTimeSta
mp()
   15:                                 + ".param");
   16:                 if (file.exists())
   17:                         file.delete();
   18:                 FileOutputStream out = new FileOutputStream(file);
   19:                 return out;
   20:         }
   21:
   22:         static public FileOutputStream getWaypointFileStream(String name)
   23:                         throws FileNotFoundException {
   24:                 File myDir = new File(DirectoryPath.getWaypointsPath());
   25:                 myDir.mkdirs();
   26:                 File file = new File(myDir, name + "-" + FileManager.getTimeStamp(
)
   27:                                 + ".txt");
   28:                 if (file.exists())
   29:                         file.delete();
   30:                 FileOutputStream out = new FileOutputStream(file);
   31:                 return out;
   32:         }
   33:
   34:         /**
   35:          * Get a file Stream for logging purposes
   36:          *
   37:          * @return output file stream for the log file
   38:          */
   39:         static public BufferedOutputStream getTLogFileStream()
   40:                         throws FileNotFoundException {
   41:                 File myDir = new File(DirectoryPath.getTLogPath());
   42:                 myDir.mkdirs();
   43:                 File file = new File(myDir, FileManager.getTimeStamp() + ".tlog");
   44:                 if (file.exists())
   45:                         file.delete();
   46:                 BufferedOutputStream out = new BufferedOutputStream(
   47:                                 new FileOutputStream(file));
   48:                 return out;
   49:         }
   50:
   51:         /**
   52:          * Creates a new .nomedia file on the maps folder
   53:          *
   54:          * It's used to hide the maps tiles from android gallery
   55:          * @throws IOException
   56:          *
   57:          */
   58:         static public void createNoMediaFile()
   59:                         throws IOException {
   60:                 File myDir = new File(DirectoryPath.getMapsPath());
   61:                 myDir.mkdirs();
   62:                 new File(myDir,".nomedia").createNewFile();
   63:         }
   64:
   65: }
```

```
    1: package com.droidplanner.file.help;
    2:
    3: import java.io.FileInputStream;
    4: import java.io.FileNotFoundException;
    5: import java.io.IOException;
    6: import java.io.InputStream;
    7: import java.util.HashMap;
    8:
    9: import android.content.Context;
   10: import android.widget.ArrayAdapter;
   11: import android.widget.SpinnerAdapter;
   12:
   13: import com.droidplanner.file.DirectoryPath;
   14: import com.droidplanner.file.FileList;
   15: import com.droidplanner.file.IO.CameraInfo;
   16: import com.droidplanner.file.IO.CameraInfoReader;
   17:
   18: public class CameraInfoLoader {
   19:
   20:         private static final String CAMERA_INFO_ASSESTS_FOLDER = "CameraInfo";
   21:         private Context context;
   22:         private HashMap<String, String> filesInSdCard = new HashMap<String, String
>();
   23:         private HashMap<String, String> filesInAssets = new HashMap<String, String
>();
   24:
   25:         public CameraInfoLoader(Context context) {
   26:                 this.context = context;
   27:         }
   28:
   29:         public CameraInfo openFile(String file) throws Exception {
   30:                 if (filesInSdCard.containsKey(file)) {
   31:                         return readSdCardFile(file);
   32:                 } else if (filesInAssets.containsKey(file)) {
   33:                         return readAssetsFile(file);
   34:                 } else {
   35:                         throw new FileNotFoundException();
   36:                 }
   37:         }
   38:
   39:         private CameraInfo readSdCardFile(String file) throws Exception {
   40:                 CameraInfoReader reader = new CameraInfoReader();
   41:                 InputStream inputStream = new FileInputStream(filesInSdCard.get(fi
le));
   42:                 reader.openFile(inputStream);
   43:                 inputStream.close();
   44:                 return reader.getCameraInfo();
   45:         }
   46:
   47:         private CameraInfo readAssetsFile(String file) throws Exception {
   48:                 CameraInfoReader reader = new CameraInfoReader();
   49:                 InputStream inputStream = context.getAssets().open(
   50:                                 filesInAssets.get(file));
   51:                 reader.openFile(inputStream);
   52:                 inputStream.close();
   53:                 return reader.getCameraInfo();
   54:         }
   55:
   56:         public SpinnerAdapter getCameraInfoList() {
   57:                 ArrayAdapter<CharSequence> avaliableCameras = new ArrayAdapter<Cha
rSequence>(
   58:                                 context, android.R.layout.simple_spinner_dropdown_
item);
   59:                 avaliableCameras.addAll(getCameraInfoListFromStorage());
   60:                 avaliableCameras.addAll(getCameraInfoListFromAssets());
   61:                 return avaliableCameras;
   62:         }
   63:
   64:         private String[] getCameraInfoListFromAssets() {
   65:                 try {
   66:                         String[] list = context.getAssets()
   67:                                         .list(CAMERA_INFO_ASSESTS_FOLDER);
   68:                         filesInAssets.clear();
   69:                         for (String string : list) {
   70:                                 filesInAssets.put(string, CAMERA_INFO_ASSESTS_FOLD
ER + "/"
   71:                                                 + string);
   72:                         }
   73:                         return list;
   74:
   75:                 } catch (IOException e) {
   76:                         return new String[0];
   77:                 }
   78:         }
   79:
   80:         private String[] getCameraInfoListFromStorage() {
   81:                 String[] list = FileList.getCameraInfoFileList();
   82:                 filesInSdCard.clear();
   83:                 for (String string : list) {
   84:                         filesInSdCard.put(string, DirectoryPath.getCameraInfoPath(
)
   85:                                         + string);
   86:                 }
   87:                 return list;
   88:         }
   89: }
```

```
    1: package com.droidplanner.file.IO;
    2:
    3: public class CameraInfo {
    4:
    5:         public Double sensorWidth;
    6:         public Double sensorHeight;
    7:         public Double focalLength;
    8:         public Double overlap;
    9:         public Double sidelap;
   10:         public boolean isInLandscapeOrientation = true;
   11:         public Double sensorResolution;
   12:
   13:         public Double getSensorLateralSize() {
   14:                 if (isInLandscapeOrientation){
   15:                         return  sensorWidth;
   16:                 }else{
   17:                         return sensorHeight;
   18:                 }
   19:         }
   20:
   21:         public Double getSensorLongitudinalSize() {
   22:                 if (isInLandscapeOrientation){
   23:                         return  sensorHeight;
   24:                 }else{
   25:                         return sensorWidth;
   26:                 }
   27:         }
   28:
   29:         @Override
   30:         public String toString() {
   31:                 return "ImageWidth:" + sensorWidth + " ImageHeight:" + sensorHeigh
t
   32:                                 + " FocalLength:" + focalLength + " Overlap:" + ov
erlap
   33:                                 + " Sidelap:" + sidelap + " isInLandscapeOrientati
on:"
   34:                                 + isInLandscapeOrientation;
   35:
   36:         }
   37:
   38: }
```

```
 1: package com.droidplanner.file.IO;
 2:
 3: import java.io.IOException;
 4: import java.io.InputStream;
 5:
 6: import org.xmlpull.v1.XmlPullParser;
 7: import org.xmlpull.v1.XmlPullParserException;
 8:
 9: import android.util.Xml;
10:
11: /**
12:  * Class to parse a Kml file, based on the code from
13:  * http://developer.android.com/training/basics/network-ops/xml.html
14:  *
15:  */
16: public class CameraInfoReader {
17:
18:         private XmlPullParser parser;
19:
20:         private CameraInfo cameraInfo = new CameraInfo();
21:
22:         public void openFile(InputStream inputStream) throws Exception {
23:                 parse(inputStream);
24:                 inputStream.close();
25:         }
26:
27:         public CameraInfo getCameraInfo() {
28:                 return cameraInfo;
29:         }
30:
31:         public static CameraInfo getNewMockCameraInfo() {
32:                 CameraInfo cameraInfo = new CameraInfo();
33:                 cameraInfo.sensorHeight = 4.22;
34:                 cameraInfo.sensorWidth = 6.12;
35:                 cameraInfo.focalLength = 7.0;
36:                 cameraInfo.sensorResolution = 10.1;
37:                 cameraInfo.overlap = 50.0;
38:                 cameraInfo.sidelap = 60.0;
39:                 cameraInfo.isInLandscapeOrientation = false;
40:                 return cameraInfo;
41:         }
42:
43:         public void parse(InputStream in) throws XmlPullParserException,
44:                         IOException {
45:                 parser = Xml.newPullParser();
46:                 parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false)
;
47:                 parser.setInput(in, null);
48:                 parser.nextTag();
49:                 readCameraInfo();
50:         }
51:
52:         private void readCameraInfo() throws XmlPullParserException, IOException {
53:
54:                 parser.require(XmlPullParser.START_TAG, null, "cameraInfo");
55:                 while (parser.next() != XmlPullParser.END_TAG) {
56:                         if (parser.getEventType() != XmlPullParser.START_TAG) {
57:                                 continue;
58:                         }
59:                         String name = parser.getName();
60:                         // Starts by looking for the entry tag
61:                         if (name.equals("SensorWidth")) {
62:                                 cameraInfo.sensorWidth = readDouble("SensorWidth")
;
63:                         } else if (name.equals("SensorHeight")) {
64:                                 cameraInfo.sensorHeight = readDouble("SensorHeight
");
65:                         } else if (name.equals("SensorResolution")) {
66:                                 cameraInfo.sensorResolution = readDouble("SensorRe
solution");
67:                         } else if (name.equals("FocalLength")) {
68:                                 cameraInfo.focalLength = readDouble("FocalLength")
;
69:                         } else if (name.equals("Overlap")) {
70:                                 cameraInfo.overlap = readDouble("Overlap");
71:                         } else if (name.equals("Sidelap")) {
72:                                 cameraInfo.sidelap = readDouble("Sidelap");
73:                         } else if (name.equals("Orientation")) {
74:                                 cameraInfo.isInLandscapeOrientation = readText().e
quals("Portrait")?false:true;
75:                         } else {
76:                                 skip();
77:                         }
78:                 }
79:         }
80:
81:         private Double readDouble(String entry) throws IOException,
82:                         XmlPullParserException {
83:                 parser.require(XmlPullParser.START_TAG, null, entry);
84:                 Double value = Double.valueOf(readText());
85:                 parser.require(XmlPullParser.END_TAG, null, entry);
86:                 return value;
87:         }
88:
89:         // For the tags title and summary, extracts their text values.
90:         private String readText() throws IOException, XmlPullParserException {
91:                 String result = "";
92:                 if (parser.next() == XmlPullParser.TEXT) {
93:                         result = parser.getText();
94:                         parser.nextTag();
95:                 }
96:                 return result;
97:         }
98:
99:         // Skips tags the parser isn't interested in. Uses depth to handle
100:         // nested tags. i.e.,
101:         // if the next tag after a START_TAG isn't a matching END_TAG, it keeps
102:         // going until it
103:         // finds the matching END_TAG (as indicated by the value of "depth"
104:         // being 0).
105:         private void skip() throws XmlPullParserException, IOException {
106:                 if (parser.getEventType() != XmlPullParser.START_TAG) {
107:                         throw new IllegalStateException();
108:                 }
109:                 int depth = 1;
110:                 while (depth != 0) {
111:                         switch (parser.next()) {
112:                         case XmlPullParser.END_TAG:
113:                                 depth--;
114:                                 break;
115:                         case XmlPullParser.START_TAG:
116:                                 depth++;
117:                                 break;
118:                         }
119:                 }
120:         }
121: }
```

```
 1: package com.droidplanner.file.IO;
 2:
 3: import java.io.FileInputStream;
 4: import java.io.FileNotFoundException;
 5: import java.io.IOException;
 6: import java.io.InputStream;
 7: import java.util.ArrayList;
 8: import java.util.List;
 9: import java.util.zip.ZipEntry;
10: import java.util.zip.ZipInputStream;
11:
12: import org.xmlpull.v1.XmlPullParser;
13: import org.xmlpull.v1.XmlPullParserException;
14:
15: import android.util.Xml;
16:
17: import com.droidplanner.dialogs.openfile.OpenFileDialog.FileReader;
18: import com.droidplanner.file.DirectoryPath;
19: import com.droidplanner.file.FileList;
20: import com.droidplanner.gcp.Gcp;
21:
22: /**
23:  * Class to parse a Kml file, based on the code from
24:  * http://developer.android.com/training/basics/network-ops/xml.html ;
25:  *
26:  */
27: public class GcpReader implements FileReader {
28:         private final String ns = null;
29:
30:         public List<Gcp> gcpList;
31:
32:         public boolean openGCPFile(String fileWithPath) {
33:                 boolean returnValue = false;
34:                 if (fileWithPath.endsWith(".kmz")) {
35:                         returnValue = openKMZ(fileWithPath);
36:                 } else if (fileWithPath.endsWith(".kml")) {
37:                         returnValue = openKML(fileWithPath);
38:                 }
39:                 return returnValue;
40:         }
41:
42:         private boolean openKML(String fileWithPath) {
43:                 try {
44:                         FileInputStream in = new FileInputStream(fileWithPath);
45:
46:                         gcpList = parse(in);
47:                         in.close();
48:                 } catch (FileNotFoundException e) {
49:                         e.printStackTrace();
50:                         return false;
51:                 } catch (XmlPullParserException e) {
52:                         e.printStackTrace();
53:                         return false;
54:                 } catch (IOException e) {
55:                         e.printStackTrace();
56:                         return false;
57:                 }
58:                 return true;
59:         }
60:
61:         private boolean openKMZ(String fileWithPath) {
62:                 try {
63:                         ZipInputStream zin = new ZipInputStream(new FileInputStrea
m(
64:                                         fileWithPath));
65:                         ZipEntry ze;
66:                         while ((ze = zin.getNextEntry()) != null) {
67:                                 if (ze.getName().contains(".kml")) {
68:                                         gcpList = parse(zin);
69:                                 }
70:                         }
71:                         zin.close();
72:                 } catch (FileNotFoundException e) {
73:                         e.printStackTrace();
74:                         return false;
75:                 } catch (XmlPullParserException e) {
76:                         e.printStackTrace();
77:                         return false;
78:                 } catch (IOException e) {
79:                         e.printStackTrace();
80:                         return false;
81:                 }
82:                 return true;
83:         }
84:
85:         public List<Gcp> parse(InputStream in) throws XmlPullParserException,
86:                         IOException {
87:                 gcpList = new ArrayList<Gcp>();
88:                 XmlPullParser parser = Xml.newPullParser();
89:                 parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false)
90:                 parser.setInput(in, null);
91:                 parser.nextTag();
92:                 readFeed(parser);
93:                 return gcpList;
94:         }
95:
96:         private void readFeed(XmlPullParser parser) throws XmlPullParserException,
97:                         IOException {
98:
99:                 parser.require(XmlPullParser.START_TAG, ns, "kml");
100:                while (parser.next() != XmlPullParser.END_DOCUMENT) {
101:                        if (parser.getEventType() != XmlPullParser.START_TAG) {
102:                                continue;
103:                        }
104:                        String name = parser.getName();
105:                        // Starts by looking for the entry tag
106:                        if (name.equals("Placemark")) {
107:                                readPlacemark(parser);
108:                        }
109:                }
110:        }
111:
112:        private void readPlacemark(XmlPullParser parser)
113:                        throws XmlPullParserException, IOException {
114:                parser.require(XmlPullParser.START_TAG, ns, "Placemark");
115:                Gcp point = null;
116:                while (parser.next() != XmlPullParser.END_TAG) {
117:                        if (parser.getEventType() != XmlPullParser.START_TAG) {
118:                                continue;
119:                        }
120:                        String name = parser.getName();
121:                        if (name.equals("Point")) {
122:                                point = readPoint(parser);
123:                                if (point != null) {
124:                                        gcpList.add(point);
125:                                }
126:                        } else {
127:                                skip(parser);
128:                        }
129:                }
130:        }
131:
132:        // Processes Point tags in the feed.
```

```
133:         private Gcp readPoint(XmlPullParser parser) throws IOException,
134:                         XmlPullParserException {
135:                 Gcp point = null;
136:                 while (parser.next() != XmlPullParser.END_TAG) {
137:                         if (parser.getEventType() != XmlPullParser.START_TAG) {
138:                                 continue;
139:                         }
140:                         String name = parser.getName();
141:                         if (name.equals("coordinates")) {
142:                                 point = readCoordinate(parser);
143:                         } else {
144:                                 skip(parser);
145:                         }
146:                 }
147:                 return point;
148:         }
149:
150:         private Gcp readCoordinate(XmlPullParser parser) throws IOException,
151:                         XmlPullParserException {
152:                 Double Lat, Lng;
153:
154:                 parser.require(XmlPullParser.START_TAG, ns, "coordinates");
155:                 String coordString = readText(parser);
156:                 parser.require(XmlPullParser.END_TAG, ns, "coordinates");
157:
158:                 String title[] = coordString.split(",");
159:                 Lng = Double.valueOf(title[0]);
160:                 Lat = Double.valueOf(title[1]);
161:
162:                 return (new Gcp(Lat, Lng));
163:         }
164:
165:         // For the tags title and summary, extracts their text values.
166:         private String readText(XmlPullParser parser) throws IOException,
167:                         XmlPullParserException {
168:                 String result = "";
169:                 if (parser.next() == XmlPullParser.TEXT) {
170:                         result = parser.getText();
171:                         parser.nextTag();
172:                 }
173:                 return result;
174:         }
175:
176:         // Skips tags the parser isn't interested in. Uses depth to handle
177:         // nested tags. i.e.,
178:         // if the next tag after a START_TAG isn't a matching END_TAG, it keeps
179:         // going until it
180:         // finds the matching END_TAG (as indicated by the value of "depth"
181:         // being 0).
182:         private void skip(XmlPullParser parser) throws XmlPullParserException,
183:                         IOException {
184:                 if (parser.getEventType() != XmlPullParser.START_TAG) {
185:                         throw new IllegalStateException();
186:                 }
187:                 int depth = 1;
188:                 while (depth != 0) {
189:                         switch (parser.next()) {
190:                         case XmlPullParser.END_TAG:
191:                                 depth--;
192:                                 break;
193:                         case XmlPullParser.START_TAG:
194:                                 depth++;
195:                                 break;
196:                         }
197:                 }
198:         }
199:
200:         @Override
201:         public String getPath() {
202:                 return DirectoryPath.getGCPPath();
203:         }
204:
205:         @Override
206:         public String[] getFileList() {
207:                 return FileList.getKMZFileList();
208:         }
209:
210:         @Override
211:         public boolean openFile(String filenameWithPath) {
212:                 return openGCPFile(filenameWithPath);
213:         }
214: }
```

```
    1: package com.droidplanner.file.IO;
    2:
    3: import java.io.BufferedReader;
    4: import java.io.FileInputStream;
    5: import java.io.IOException;
    6: import java.io.InputStreamReader;
    7: import java.util.ArrayList;
    8: import java.util.List;
    9:
   10: import com.MAVLink.Messages.ApmCommands;
   11: import com.droidplanner.dialogs.openfile.OpenFileDialog.FileReader;
   12: import com.droidplanner.drone.variables.Home;
   13: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
   14: import com.droidplanner.file.DirectoryPath;
   15: import com.droidplanner.file.FileList;
   16: import com.droidplanner.file.FileManager;
   17:
   18: public class MissionReader implements FileReader {
   19:         private Home home;
   20:         private List<SpatialCoordItem> waypoints;
   21:
   22:         public MissionReader() {
   23:                 this.waypoints = new ArrayList<SpatialCoordItem>();
   24:         }
   25:
   26:         public boolean openMission(String file) {
   27:                 if (!FileManager.isExternalStorageAvaliable()) {
   28:                         return false;
   29:                 }
   30:                 try {
   31:                         FileInputStream in = new FileInputStream(file);
   32:                         BufferedReader reader = new BufferedReader(
   33:                                         new InputStreamReader(in));
   34:
   35:                         if (!isWaypointFile(reader)) {
   36:                                 in.close();
   37:                                 return false;
   38:                         }
   39:                         parseHomeLine(reader);
   40:                         parseWaypointLines(reader);
   41:
   42:                         in.close();
   43:
   44:                 } catch (Exception e) {
   45:                         e.printStackTrace();
   46:                         return false;
   47:                 }
   48:
   49:                 return true;
   50:         }
   51:
   52:         public Home getHome() {
   53:                 return home;
   54:         }
   55:
   56:         public List<SpatialCoordItem> getWaypoints() {
   57:                 return waypoints;
   58:         }
   59:
   60:         private void parseWaypointLines(BufferedReader reader) throws IOException
{
   61:                 String line;
   62:                 waypoints.clear();
   63:                 while ((line = reader.readLine()) != null) {
   64:                         throw new IllegalArgumentException("NOT implemented"); //T
ODO implement this
   65:                         /*
   66:                         String[] RowData = line.split("\t");
   67:                         Waypoint wp = new Waypoint(Double.valueOf(RowData[8]),
   68:                                         Double.valueOf(RowData[9]), Double.valueOf
(RowData[10]));
   69:                         wp.setNumber(Integer.valueOf(RowData[0]));
   70:                         wp.setFrame(Integer.valueOf(RowData[2]));
   71:                         wp.setCmd(ApmCommands.getCmd(Integer.valueOf(RowData[3])))
;
   72:                         wp.setParameters(Float.valueOf(RowData[4]),
   73:                                         Float.valueOf(RowData[5]), Float.valueOf(R
owData[6]),
   74:                                         Float.valueOf(RowData[7]));
   75:                         wp.setAutoContinue(Integer.valueOf(RowData[11]));
   76:                         waypoints.add(wp);
   77:                         */
   78:                 }
   79:
   80:         }
   81:
   82:         private void parseHomeLine(BufferedReader reader) throws IOException {
   83:                 throw new IllegalArgumentException("NOT implemented"); //TODO impl
ement this
   84:                 /*
   85:                 String[] RowData = reader.readLine().split("\t");
   86:                 home = new Home(Double.valueOf(RowData[8]), Double.valueOf(RowData
[9]),
   87:                                 Double.valueOf(RowData[10]));
   88:                 home.setNumber(Integer.valueOf(RowData[0]));
   89:                 home.setFrame(Integer.valueOf(RowData[2]));
   90:                 home.setCmd(ApmCommands.getCmd(Integer.valueOf(RowData[3])));
   91:                 home.setParameters(Float.valueOf(RowData[4]),
   92:                                 Float.valueOf(RowData[5]), Float.valueOf(RowData[6
]),
   93:                                 Float.valueOf(RowData[7]));
   94:                 home.setAutoContinue(Integer.valueOf(RowData[11]));
   95:                 */
   96:         }
   97:
   98:         private static boolean isWaypointFile(BufferedReader reader)
   99:                         throws IOException {
  100:                 return reader.readLine().contains("QGC WPL 110");
  101:         }
  102:
  103:         @Override
  104:         public String getPath() {
  105:                 return DirectoryPath.getWaypointsPath();
  106:         }
  107:
  108:         @Override
  109:         public String[] getFileList() {
  110:                 return FileList.getWaypointFileList();
  111:         }
  112:
  113:         @Override
  114:         public boolean openFile(String file) {
  115:                 return openMission(file);
  116:         }
  117: }
```

```
    1: package com.droidplanner.file.IO;
    2:
    3: import java.io.FileOutputStream;
    4: import java.io.IOException;
    5: import java.util.List;
    6: import java.util.Locale;
    7:
    8: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
    9: import com.droidplanner.file.FileManager;
   10: import com.droidplanner.file.FileStream;
   11:
   12: public class MissionWriter {
   13:         private SpatialCoordItem home;
   14:         private List<SpatialCoordItem> waypoints;
   15:         private String name = "";
   16:
   17:         public MissionWriter(SpatialCoordItem home, List<SpatialCoordItem> waypoin
ts, String name) {
   18:                 this.home = home;
   19:                 this.waypoints = waypoints;
   20:                 this.name = name;
   21:         }
   22:
   23:         public MissionWriter(SpatialCoordItem home, List<SpatialCoordItem> waypoin
ts) {
   24:                 this(home, waypoints, "waypoints");
   25:         }
   26:
   27:         public boolean saveWaypoints() {
   28:                 try {
   29:                         if (!FileManager.isExternalStorageAvaliable()) {
   30:                                 return false;
   31:                         }
   32:                         FileOutputStream out = FileStream.getWaypointFileStream(na
me);
   33:
   34:                         writeFirstLine(out);
   35:                         writeHomeLine(out);
   36:                         writeWaypointsLines(out);
   37:                         out.close();
   38:
   39:                 } catch (Exception e) {
   40:                         e.printStackTrace();
   41:                         return false;
   42:                 }
   43:                 return true;
   44:         }
   45:
   46:         private void writeFirstLine(FileOutputStream out) throws IOException {
   47:                 out.write(String.format(Locale.ENGLISH, "QGC WPL 110\n").getBytes(
));
   48:         }
   49:
   50:         private void writeHomeLine(FileOutputStream out) throws IOException {
   51:                 throw new IllegalArgumentException("NOT implemented"); //TODO impl
ement this
   52:                 /*
   53:                 out.write(String.format(Locale.ENGLISH,
   54:                                 "0\t1\t0\t16\t0\t0\t0\t0\t%f\t%f\t%f\t1\n",
   55:                                 home.getCoord().latitude, home.getCoord().longitud
e,
   56:                                 home.getHeight()).getBytes());
   57:                                 */
   58:         }
   59:
   60:         private void writeWaypointsLines(FileOutputStream out) throws IOException
{
   61:
   62:                 throw new IllegalArgumentException("NOT implemented");  //TODO imp
lement this
   63:                 /*
   64:                 for (int i = 0; i < waypoints.size(); i++) {
   65:                         Waypoint wp = waypoints.get(i);
   66:                         out.write(String.format(Locale.ENGLISH,
   67:                                         "%d\t0\t%d\t%d\t%f\t%f\t%f\t%f\t%f\t%f\t%f
\t%d\n", i + 1,
   68:                                         wp.getFrame(), wp.getCmd().getType(), wp.g
etParam1(),
   69:                                         wp.getParam2(), wp.getParam3(), wp.getPara
m4(),
   70:                                         wp.getCoord().latitude, wp.getCoord().long
itude,
   71:                                         wp.getHeight(), wp.getAutoContinue()).getB
ytes());
   72:                 }
   73:                 */
   74:         }
   75: }
```

```
1: package com.droidplanner.file.IO;
2:
3: import com.droidplanner.parameters.ParameterMetadata;
4:
5: import java.util.HashMap;
6:
7: public class ParameterMetadataMap extends HashMap<String, ParameterMetadata> {
8: }
```

```
    1: package com.droidplanner.file.IO;
    2:
    3: import android.util.Xml;
    4: import com.droidplanner.parameters.ParameterMetadata;
    5: import org.xmlpull.v1.XmlPullParser;
    6: import org.xmlpull.v1.XmlPullParserException;
    7:
    8: import java.io.IOException;
    9: import java.io.InputStream;
   10:
   11:
   12: /**
   13:  * Parameter metadata parser extracted from parameters
   14:  *
   15:  */
   16: public class ParameterMetadataMapReader {
   17:
   18:     private static final String METADATA_DISPLAYNAME = "DisplayName";
   19:     private static final String METADATA_DESCRIPTION = "Description";
   20:     private static final String METADATA_UNITS = "Units";
   21:     private static final String METADATA_VALUES = "Values";
   22:     private static final String METADATA_RANGE = "Range";
   23:
   24:
   25:     public static ParameterMetadataMap open(InputStream inputStream, String metada
taType) throws XmlPullParserException, IOException {
   26:         try {
   27:             XmlPullParser parser = Xml.newPullParser();
   28:             parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
   29:             parser.setInput(inputStream, null);
   30:             return parseMetadata(parser, metadataType);
   31:
   32:         } finally {
   33:             try { inputStream.close(); } catch (IOException e) { /*nop*/ }
   34:         }
   35:     }
   36:
   37:
   38:     private static ParameterMetadataMap parseMetadata(XmlPullParser parser, String
 metadataType) throws XmlPullParserException, IOException {
   39:         String name;
   40:         boolean parsing = false;
   41:         ParameterMetadata metadata = null;
   42:         ParameterMetadataMap metadataMap = new ParameterMetadataMap();
   43:
   44:         int eventType = parser.getEventType();
   45:         while (eventType != XmlPullParser.END_DOCUMENT) {
   46:
   47:             switch (eventType) {
   48:                 case XmlPullParser.START_TAG:
   49:                     name = parser.getName();
   50:                     // name == metadataType: start collecting metadata(s)
   51:                     // metadata == null: create new metadata w/ name
   52:                     // metadata != null: add to metadata as property
   53:                     if(metadataType.equals(name)) {
   54:                         parsing = true;
   55:                     } else if(parsing) {
   56:                         if(metadata == null) {
   57:                             metadata = new ParameterMetadata();
   58:                             metadata.setName(name);
   59:                         } else {
   60:                             addMetaDataProperty(metadata, name, parser.nextText())
;
   61:                         }
   62:                     }
   63:                     break;
   64:
   65:                 case XmlPullParser.END_TAG:
   66:                     name = parser.getName();
   67:                     // name == metadataType: done
   68:                     // name == metadata.name: add metadata to metadataMap
   69:                     if(metadataType.equals(name)) {
   70:                         return metadataMap;
   71:                     } else if(metadata != null && metadata.getName().equals(name))
{
   72:                         metadataMap.put(metadata.getName(), metadata);
   73:                         metadata = null;
   74:                     }
   75:                     break;
   76:             }
   77:             eventType = parser.next();
   78:         }
   79:         // no metadata
   80:         return null;
   81:     }
   82:
   83:     private static void addMetaDataProperty(ParameterMetadata metaData, String nam
e, String text) {
   84:         if(name.equals(METADATA_DISPLAYNAME))
   85:             metaData.setDisplayName(text);
   86:         else if(name.equals(METADATA_DESCRIPTION))
   87:             metaData.setDescription(text);
   88:
   89:         else if(name.equals(METADATA_UNITS))
   90:             metaData.setUnits(text);
   91:         else if(name.equals(METADATA_RANGE))
   92:             metaData.setRange(text);
   93:         else if(name.equals(METADATA_VALUES))
   94:             metaData.setValues(text);
   95:     }
   96: }
```

```java
package com.droidplanner.file.IO;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

import com.droidplanner.file.DirectoryPath;
import com.droidplanner.file.FileList;
import com.droidplanner.file.FileManager;
import com.droidplanner.parameters.Parameter;

public class ParameterReader implements
                com.droidplanner.dialogs.openfile.OpenFileDialog.FileReader {
        private List<Parameter> parameters;

        public ParameterReader() {
                this.parameters = new ArrayList<Parameter>();
        }

        public boolean openFile(String itemList) {
                if (!FileManager.isExternalStorageAvaliable()) {
                        return false;
                }
                try {
                        FileInputStream in = new FileInputStream(itemList);
                        BufferedReader reader = new BufferedReader(
                                        new InputStreamReader(in));

                        if (!isParameterFile(reader)) {
                                in.close();
                                return false;
                        }
                        parseWaypointLines(reader);

                        in.close();

                } catch (Exception e) {
                        e.printStackTrace();
                        return false;
                }

                return true;
        }

        private void parseWaypointLines(BufferedReader reader) throws IOException
{
                String line;
                parameters.clear();
                while ((line = reader.readLine()) != null) {
                        try {
                                parseLine(line);
                        } catch (Exception e) {
                        }
                }
        }

        private void parseLine(String line) throws Exception {
                String[] RowData = splitLine(line);

                String name = RowData[0];
                Double value = Double.valueOf(RowData[1]);

                Parameter.checkParameterName(name);

                parameters.add(new Parameter(name, value));
        }

        private String[] splitLine(String line) throws Exception {
                String[] RowData = line.split(",");
                if (RowData.length != 2) {
                        throw new Exception("Invalid Length");
                }
                RowData[0] = RowData[0].trim();
                return RowData;
        }

        private static boolean isParameterFile(BufferedReader reader)
                        throws IOException {
                return reader.readLine().contains("#NOTE");
        }

        public List<Parameter> getParameters() {
                return parameters;
        }

        @Override
        public String getPath() {
                return DirectoryPath.getParametersPath();
        }

        @Override
        public String[] getFileList() {
                return FileList.getParametersFileList();
        }
}
```

```
    1: package com.droidplanner.file.IO;
    2:
    3: import java.io.FileOutputStream;
    4: import java.io.IOException;
    5: import java.util.List;
    6: import java.util.Locale;
    7:
    8: import com.droidplanner.file.FileManager;
    9: import com.droidplanner.file.FileStream;
   10: import com.droidplanner.parameters.Parameter;
   11:
   12: public class ParameterWriter {
   13:         private List<Parameter> parameterList;
   14:
   15:         public ParameterWriter(List<Parameter> param) {
   16:                 this.parameterList = param;
   17:         }
   18:
   19:         public boolean saveParametersToFile() {
   20:                 try {
   21:                         if (!FileManager.isExternalStorageAvaliable()) {
   22:                                 return false;
   23:                         }
   24:                         FileOutputStream out = FileStream.getParameterFileStream()
;
   25:
   26:                         writeFirstLine(out);
   27:
   28:                         writeWaypointsLines(out);
   29:                         out.close();
   30:
   31:                 } catch (Exception e) {
   32:                         e.printStackTrace();
   33:                         return false;
   34:                 }
   35:                 return true;
   36:         }
   37:
   38:         private void writeFirstLine(FileOutputStream out) throws IOException {
   39:                 out.write((new String("#NOTE: " + FileManager.getTimeStamp() + "\n
")
   40:                                 .getBytes()));
   41:         }
   42:
   43:         private void writeWaypointsLines(FileOutputStream out) throws IOException
{
   44:                 for (Parameter param : parameterList) {
   45:                         out.write(String.format(Locale.ENGLISH, "%s , %f\n", param
.name,
   46:                                         param.value).getBytes());
   47:                 }
   48:         }
   49: }
```

```
    1: package com.droidplanner.fragments.checklist;
    2:
    3: import java.util.ArrayList;
    4: import java.util.HashMap;
    5: import java.util.List;
    6:
    7: import com.droidplanner.R;
    8: import com.droidplanner.checklist.CheckListAdapter;
    9: import com.droidplanner.checklist.CheckListAdapter.OnCheckListItemUpdateListener;
   10: import com.droidplanner.checklist.CheckListItem;
   11: import com.droidplanner.checklist.CheckListXmlParser;
   12:
   13: import android.os.Bundle;
   14: import android.support.v4.app.DialogFragment;
   15: import android.view.LayoutInflater;
   16: import android.view.View;
   17: import android.view.ViewGroup;
   18: import android.view.WindowManager.LayoutParams;
   19: import android.widget.ExpandableListView;
   20:
   21: public class ListXmlFragment extends DialogFragment implements
   22:                 OnCheckListItemUpdateListener {
   23:         private List<String> listDataHeader;
   24:         private List<CheckListItem> checkItemList;
   25:         private HashMap<String, List<CheckListItem>> listDataChild;
   26:         private CheckListAdapter listAdapter;
   27:
   28:         public ListXmlFragment() {
   29:                 // TODO Auto-generated constructor stub
   30:         }
   31:
   32:         @Override
   33:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   34:                         Bundle savedInstanceState) {
   35:                 View view = inflater.inflate(R.layout.layout_checklist, container)
;
   36:                 ExpandableListView expListView = (ExpandableListView) view
   37:                                 .findViewById(R.id.expListView);
   38:
   39:                 getXmlListItems();
   40:                 setDialogOptions();
   41:
   42: //              listAdapter = new CheckListAdapter(inflater, listDataHeader,
   43: //                              listDataChild);
   44: //              listAdapter.setOnCheckListItemUpdateListener(this);
   45: //              expListView.setAdapter(listAdapter);
   46:                 return view;
   47:         }
   48:
   49:         private void getXmlListItems() {
   50:                 CheckListXmlParser xml = new CheckListXmlParser();
   51:                 xml.getListItemsFromResource(getActivity(), R.xml.checklist_defaul
t);
   52:                 listDataHeader = xml.getCategories();
   53:                 checkItemList = xml.getCheckListItems();
   54:
   55:                 listDataChild = new HashMap<String, List<CheckListItem>>();
   56:                 List<CheckListItem> cli;
   57:
   58:                 for (int h = 0; h < listDataHeader.size(); h++) {
   59:                         cli = new ArrayList<CheckListItem>();
   60:                         for (int i = 0; i < checkItemList.size(); i++) {
   61:                                 CheckListItem c = checkItemList.get(i);
   62:                                 if (c.getCategoryIndex() == h)
   63:                                         cli.add(c);
   64:                         }
   65:                         listDataChild.put(listDataHeader.get(h), cli);
   66:                 }
   67:         }
   68:
   69:         private void setListView(ExpandableListView expListView) {
   70:                 // setting list adapter
   71:
   72:         }
   73:
   74:         private void setDialogOptions() {
   75:
   76:                 getDialog().setTitle("Pre-Flight Checklist");
   77:                 getDialog().getWindow().setSoftInputMode(
   78:                 LayoutParams.SOFT_INPUT_STATE_ALWAYS_VISIBLE);
   79:
   80:                 // getDialog().getWindow().setBackgroundDrawableResource(R.drawabl
e.round_dialog);
   81:                 setStyle(DialogFragment.STYLE_NORMAL, android.R.style.Theme_Holo);
   82:         }
   83:
   84:
   85:         @Override
   86:         public void onSelectUpdate(CheckListItem checkListItem, int selectId) {
   87:                 // TODO Auto-generated method stub
   88:
   89:         }
   90:
   91:         @Override
   92:         public void onCheckBoxUpdate(CheckListItem checkListItem, boolean isChecke
d) {
   93:                 // TODO Auto-generated method stub
   94:
   95:         }
   96:
   97:         @Override
   98:         public void onSwitchUpdate(CheckListItem checkListItem, boolean isSwitched
) {
   99:                 // TODO Auto-generated method stub
  100:
  101:         }
  102:
  103:         @Override
  104:         public void onToggleUpdate(CheckListItem checkListItem, boolean isToggled)
 {
  105:                 // TODO Auto-generated method stub
  106:
  107:         }
  108:
  109:         @Override
  110:         public void onValueUpdate(CheckListItem checkListItem, String newValue) {
  111:                 // TODO Auto-generated method stub
  112:
  113:         }
  114:
  115:         @Override
  116:         public void onRadioGroupUpdate(CheckListItem checkListItem, int checkId) {
  117:                 // TODO Auto-generated method stub
  118:
  119:         }
  120: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import android.app.Activity;
    4: import android.app.Fragment;
    5: import android.os.Bundle;
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.View.OnClickListener;
    9: import android.view.ViewGroup;
   10: import android.widget.RadioButton;
   11:
   12: import com.droidplanner.R;
   13:
   14: public class EditorToolsFragment extends Fragment implements OnClickListener {
   15:
   16:         public enum EditorTools {
   17:                 MARKER, DRAW, POLY, TRASH
   18:         }
   19:
   20:         public interface OnEditorToolSelected {
   21:                 public void editorToolChanged(EditorTools tools);
   22:         }
   23:
   24:         private OnEditorToolSelected listner;
   25:         private RadioButton buttonDraw;
   26:         private RadioButton buttonMarker;
   27:         private RadioButton buttonPoly;
   28:         private RadioButton buttonTrash;
   29:         private EditorTools tool = EditorTools.MARKER;
   30:
   31:         @Override
   32:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   33:                         Bundle savedInstanceState) {
   34:                 View view = inflater.inflate(R.layout.fragment_editor_control,
   35:                                 container, false);
   36:                 buttonDraw = (RadioButton) view.findViewById(R.id.editor_tools_dra
w);
   37:                 buttonMarker = (RadioButton) view
   38:                                 .findViewById(R.id.editor_tools_marker);
   39:                 buttonPoly = (RadioButton) view.findViewById(R.id.editor_tools_pol
y);
   40:                 buttonTrash = (RadioButton) view.findViewById(R.id.editor_tools_tr
ash);
   41:
   42:                 buttonDraw.setOnClickListener(this);
   43:                 buttonMarker.setOnClickListener(this);
   44:                 buttonPoly.setOnClickListener(this);
   45:                 buttonTrash.setOnClickListener(this);
   46:                 return view;
   47:         }
   48:
   49:         @Override
   50:         public void onAttach(Activity activity) {
   51:                 super.onAttach(activity);
   52:                 listner = (OnEditorToolSelected) activity;
   53:         }
   54:
   55:         @Override
   56:         public void onClick(View v) {
   57:                 switch (v.getId()) {
   58:                 case R.id.editor_tools_marker:
   59:                         tool = EditorTools.MARKER;
   60:                         break;
   61:                 case R.id.editor_tools_draw:
   62:                         tool = EditorTools.DRAW;
   63:                         break;
   64:                 case R.id.editor_tools_poly:
   65:                         tool = EditorTools.POLY;
   66:                         break;
   67:                 case R.id.editor_tools_trash:
   68:                         tool = EditorTools.TRASH;
   69:                         break;
   70:                 }
   71:                 listner.editorToolChanged(getTool());
   72:         }
   73:
   74:         public EditorTools getTool() {
   75:                 return tool;
   76:         }
   77:
   78:         public void setTool(EditorTools marker) {
   79:                 RadioButton selected = null;
   80:                 buttonDraw.setChecked(false);
   81:                 buttonMarker.setChecked(false);
   82:                 buttonPoly.setChecked(false);
   83:                 buttonTrash.setChecked(false);
   84:                 switch (marker) {
   85:                 case DRAW:
   86:                         selected = buttonDraw;
   87:                         break;
   88:                 case MARKER:
   89:                         selected = buttonMarker;
   90:                         break;
   91:                 case POLY:
   92:                         selected = buttonPoly;
   93:                         break;
   94:                 case TRASH:
   95:                         selected = buttonTrash;
   96:                         break;
   97:                 }
   98:                 selected.setChecked(true);
   99:                 onClick(selected);
  100:
  101:         }
  102:
  103: }
```

```
package com.droidplanner.fragments;

import java.util.List;

import android.content.Context;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.droidplanner.drone.DroneInterfaces.MapUpdatedListner;
import com.droidplanner.drone.variables.GuidedPoint;
import com.droidplanner.drone.variables.GuidedPoint.OnGuidedListener;
import com.droidplanner.fragments.helpers.DroneMap;
import com.droidplanner.fragments.helpers.MapPath;
import com.droidplanner.fragments.markers.DroneMarker;
import com.google.android.gms.maps.GoogleMap.OnMapLongClickListener;
import com.google.android.gms.maps.GoogleMap.OnMarkerClickListener;
import com.google.android.gms.maps.GoogleMap.OnMarkerDragListener;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;

public class FlightMapFragment extends DroneMap implements
                OnMapLongClickListener, OnMarkerClickListener, OnMarkerDragListene
r, OnGuidedListener, MapUpdatedListner {
        private Polyline flightPath;
        private MapPath droneLeashPath;
        private int maxFlightPathSize;
        public boolean isAutoPanEnabled;
        private boolean isGuidedModeEnabled;

        public boolean hasBeenZoomed = false;

        public DroneMarker droneMarker;

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup viewGroup,
                        Bundle bundle) {
                View view = super.onCreateView(inflater, viewGroup, bundle);

                droneMarker = new DroneMarker(this);
                droneLeashPath = new MapPath(mMap,Color.WHITE,5);

                addFlightPathToMap();
                getPreferences();


                drone.setMapListner(this);
                mMap.setOnMapLongClickListener(this);
                mMap.setOnMarkerDragListener(this);
                mMap.setOnMarkerClickListener(this);

                drone.setGuidedPointListner(this);
                return view;
        }

        private void getPreferences() {
                Context context = this.getActivity();
                SharedPreferences prefs = PreferenceManager
                                .getDefaultSharedPreferences(context);
                maxFlightPathSize = Integer.valueOf(prefs.getString(
                                "pref_max_fligth_path_size", "0"));
                isGuidedModeEnabled = prefs.getBoolean("pref_guided_mode_enabled",
                                false);
                isAutoPanEnabled = prefs.getBoolean("pref_auto_pan_enabled", false
);
        }

        @Override
        public void update() {
                super.update();
        }

        public void addFlithPathPoint(LatLng position) {
                if (maxFlightPathSize > 0) {
                        List<LatLng> oldFlightPath = flightPath.getPoints();
                        if (oldFlightPath.size() > maxFlightPathSize) {
                                oldFlightPath.remove(0);
                        }
                        oldFlightPath.add(position);
                        flightPath.setPoints(oldFlightPath);
                }
        }

        public void clearFlightPath() {
                List<LatLng> oldFlightPath = flightPath.getPoints();
                oldFlightPath.clear();
                flightPath.setPoints(oldFlightPath);
        }

        private void addFlightPathToMap() {
                PolylineOptions flightPathOptions = new PolylineOptions();
                flightPathOptions.color(Color.argb(180, 0, 0, 200)).width(2).zInde
x(1);

                flightPath = mMap.addPolyline(flightPathOptions);
        }

        @Override
        public void onMapLongClick(LatLng coord) {
                getPreferences();
                if (isGuidedModeEnabled)
                        drone.guidedPoint.newGuidedPointWithCurrentAlt(coord);
        }

        @Override
        public void onMarkerDragStart(Marker marker){
        }

        @Override
        public void onMarkerDrag(Marker marker){
        }


        @Override
        public void onMarkerDragEnd(Marker marker){
                drone.guidedPoint.newGuidedPointwithLastAltitude(marker.getPositio
n());
        }

        @Override
        public boolean onMarkerClick(Marker marker){
                drone.guidedPoint.newGuidedPointWithCurrentAlt(marker.getPosition(
));
                return true;
        }

        @Override
        public void onGuidedPoint() {
                GuidedPoint guidedPoint = drone.guidedPoint;
```

```
130:                 markers.updateMarker(guidedPoint, true, context);
131:                 droneLeashPath.update(guidedPoint);
132:         }
133:
134:         @Override
135:         public void onDroneUpdate() {
136:                 droneMarker.onDroneUpdate();
137:                 droneLeashPath.update(drone.guidedPoint);
138:         }
139:
140:         @Override
141:         public void onDroneTypeChanged() {
142:                 droneMarker.onDroneTypeChanged();
143:         }
144: }
```

```
    1: package com.droidplanner.fragments.helpers;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.graphics.Color;
    7:
    8: import com.droidplanner.drone.variables.mission.survey.SurveyData;
    9: import com.droidplanner.helpers.geoTools.GeoTools;
   10: import com.google.android.gms.maps.GoogleMap;
   11: import com.google.android.gms.maps.model.LatLng;
   12: import com.google.android.gms.maps.model.Polygon;
   13: import com.google.android.gms.maps.model.PolygonOptions;
   14:
   15: public class CameraGroundOverlays {
   16:         public ArrayList<Polygon> cameraOverlays = new ArrayList<com.google.androi
d.gms.maps.model.Polygon>();
   17:         private GoogleMap mMap;
   18:
   19:         public CameraGroundOverlays(GoogleMap mMap) {
   20:                 this.mMap = mMap;
   21:         }
   22:
   23:         public void addOverlays(List<LatLng> cameraLocations,
   24:                         SurveyData surveyData) {
   25:                 for (LatLng latLng : cameraLocations) {
   26:                         addOneFootprint(latLng, surveyData);
   27:                 }
   28:         }
   29:
   30:
   31:         public void removeAll() {
   32:                 for (com.google.android.gms.maps.model.Polygon overlay : cameraOve
rlays) {
   33:                         overlay.remove();
   34:                 }
   35:                 cameraOverlays.clear();
   36:         }
   37:
   38:         private void addOneFootprint(LatLng latLng, SurveyData surveyData) {
   39:                 double lng = surveyData.getLateralFootPrint().valueInMeters();
   40:                 double lateral = surveyData.getLongitudinalFootPrint().valueInMete
rs();
   41:                 double halfDiag = Math.hypot(lng, lateral) / 2;
   42:                 double angle = Math.toDegrees(Math.atan(lng / lateral));
   43:                 Double orientation = surveyData.getAngle();
   44:                 addRectangleOverlay(latLng, halfDiag, angle, orientation);
   45:
   46:         }
   47:
   48:         private void addRectangleOverlay(LatLng center, double halfDiagonal,
   49:                         double centerAngle, Double orientation) {
   50:                 cameraOverlays.add(mMap.addPolygon(new PolygonOptions()
   51:                 .add(GeoTools.newCoordFromBearingAndDistance(center,
   52:                                 orientation - centerAngle, halfDiagonal),
   53:                                 GeoTools.newCoordFromBearingAndDistance(center,
   54:                                                 orientation + centerAngle, halfDia
gonal),
   55:                                                 GeoTools.newCoordFromBearingAndDis
tance(center,
   56:                                                                 orientation + 180
- centerAngle, halfDiagonal),
   57:                                                                 GeoTools.newCoordF
romBearingAndDistance(center,
   58:                                                                                 or
ientation + 180 + centerAngle, halfDiagonal))
   59:                                                                                 .f
illColor(Color.argb(40, 0, 0, 127)).strokeWidth(1)
   60:                                                                                 .s
trokeColor(Color.argb(127, 0, 0, 255))));
   61:         }
   62:
   63: }
```

```
 1: package com.droidplanner.fragments.helpers;
 2:
 3: import android.app.Activity;
 4: import android.content.Context;
 5: import android.os.Bundle;
 6: import android.view.LayoutInflater;
 7: import android.view.View;
 8: import android.view.ViewGroup;
 9:
10: import com.droidplanner.DroidPlannerApp;
11: import com.droidplanner.DroidPlannerApp.OnWaypointChangedListner;
12: import com.droidplanner.drone.Drone;
13: import com.droidplanner.drone.variables.Home;
14: import com.droidplanner.drone.variables.mission.Mission;
15: import com.droidplanner.fragments.markers.MarkerManager;
16: import com.google.android.gms.maps.GoogleMap;
17: import com.google.android.gms.maps.model.LatLng;
18:
19: public abstract class DroneMap extends OfflineMapFragment implements OnWaypointCha
ngedListner {
20:         public GoogleMap mMap;
21:         protected MarkerManager markers;
22:         protected MapPath missionPath;
23:         public Drone drone;
24:         public Mission mission;
25:         protected Context context;
26:
27:         @Override
28:         public View onCreateView(LayoutInflater inflater, ViewGroup viewGroup,
29:                         Bundle bundle) {
30:                 View view = super.onCreateView(inflater, viewGroup, bundle);
31:                 drone = ((DroidPlannerApp) getActivity().getApplication()).drone;

32:                 mission = drone.mission;
33:                 mMap = getMap();
34:                 markers = new MarkerManager(mMap);
35:                 missionPath = new MapPath(mMap);
36:                 mission.addOnMissionUpdateListner(this);
37:                 return view;
38:         }
39:
40:         @Override
41:         public void onAttach(Activity activity) {
42:                 super.onAttach(activity);
43:                 context = activity.getApplicationContext();
44:         }
45:
46:         @Override
47:         public void onDestroy() {
48:                 super.onDestroy();
49:                 mission.removeOnMissionUpdateListner(this);
50:         }
51:
52:         public LatLng getMyLocation() {
53:                 if (mMap.getMyLocation() != null) {
54:                         return new LatLng(mMap.getMyLocation().getLatitude(), mMap
55:                                         .getMyLocation().getLongitude());
56:                 } else {
57:                         return null;
58:                 }
59:         }
60:
61:         public void update() {
62:                 markers.clean();
63:
64:                 Home home = drone.home.getHome();
65:                 if (home.isValid()) {
66:                         markers.updateMarker(home, false, context);

67:                 }
68:
69:                 markers.updateMarkers(mission.getMarkers(), true, context);
70:
71:                 missionPath.update(mission);
72:         }
73:
74:         @Override
75:         public void onMissionUpdate() {
76:                 update();
77:         }
78:
79: }
```

```
    1: package com.droidplanner.fragments.helpers;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.app.Fragment;
    7: import android.gesture.GestureOverlayView;
    8: import android.gesture.GestureOverlayView.OnGestureListener;
    9: import android.graphics.Point;
   10: import android.os.Bundle;
   11: import android.view.LayoutInflater;
   12: import android.view.MotionEvent;
   13: import android.view.View;
   14: import android.view.ViewGroup;
   15:
   16: import com.droidplanner.R;
   17: import com.droidplanner.helpers.geoTools.Simplify;
   18:
   19: public class GestureMapFragment extends Fragment implements OnGestureListener {
   20:         private static final int TOLERANCE = 15;
   21:         private static final int STROKE_WIDTH = 3;
   22:
   23:         private double toleranceInPixels;
   24:
   25:         public interface OnPathFinishedListner {
   26:
   27:                 void onPathFinished(List<Point> path);                          {
   28:         }
   29:
   30:         private GestureOverlayView overlay;
   31:         private OnPathFinishedListner listner;
   32:
   33:         @Override
   34:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   35:                         Bundle savedInstanceState) {
   36:                 View view = inflater.inflate(R.layout.fragment_gesture_map, contai
ner,
   37:                                 false);
   38:                 overlay = (GestureOverlayView) view.findViewById(R.id.overlay1);
   39:                 overlay.addOnGestureListener(this);
   40:                 overlay.setEnabled(false);
   41:
   42:                 overlay.setGestureStrokeWidth(scaleDpToPixels(STROKE_WIDTH));
   43:                 toleranceInPixels = scaleDpToPixels(TOLERANCE);
   44:                 return view;
   45:         }
   46:
   47:         private int scaleDpToPixels(double value) {
   48:                 final float scale = getResources().getDisplayMetrics().density;
   49:                 return (int) Math.round(value*scale);
   50:         }
   51:
   52:         public void enableGestureDetection() {
   53:                 overlay.setEnabled(true);
   54:         }
   55:
   56:         public void disableGestureDetection() {
   57:                 overlay.setEnabled(false);
   58:         }
   59:
   60:         public void setOnPathFinishedListner(OnPathFinishedListner listner) {
   61:                 this.listner = listner;
   62:         }
   63:
   64:         @Override
   65:         public void onGestureEnded(GestureOverlayView arg0, MotionEvent arg1) {
   66:                 overlay.setEnabled(false);
   67:                 List<Point> path = decodeGesture();
   68:                 if (path.size() > 1) {
   69:                         path = Simplify.simplify(path, toleranceInPixels);
   70:                 }
   71:                 listner.onPathFinished(path);
   72:         }
   73:
   74:         private List<Point> decodeGesture() {
   75:                 List<Point> path = new ArrayList<Point>();
   76:                 extractPathFromGesture(path);
   77:                 return path;
   78:         }
   79:
   80:         private void extractPathFromGesture(List<Point> path) {
   81:                 float[] points = overlay.getGesture().getStrokes().get(0).points;
   82:                 for (int i = 0; i < points.length; i += 2) {
   83:                         path.add(new Point((int) points[i], (int) points[i + 1]));
   84:                 }
   85:         }
   86:
   87:         @Override
   88:         public void onGesture(GestureOverlayView arg0, MotionEvent arg1) {
   89:         }
   90:
   91:         @Override
   92:         public void onGestureCancelled(GestureOverlayView arg0, MotionEvent arg1)
{
   93:         }
   94:
   95:         @Override
   96:         public void onGestureStarted(GestureOverlayView arg0, MotionEvent arg1) {
   97:         }
   98:
   99:
  100: }
```

```
 1: package com.droidplanner.fragments.helpers;
 2:
 3: import com.droidplanner.drone.variables.GuidedPoint;
 4: import com.google.android.gms.maps.model.LatLng;
 5:
 6: /**
 7:  * Created with IntelliJ IDEA.
 8:  * User: rgayle
 9:  * Date: 2013-10-07
10:  * Time: 12:30 AM
11:  * To change this template use File | Settings | File Templates.
12:  */
13: public interface GuidePointListener {
14:         void OnGuidePointMoved();
15: }
```

```
 1: package com.droidplanner.fragments.helpers;
 2:
 3: import java.util.List;
 4:
 5: import android.graphics.Color;
 6:
 7: import com.google.android.gms.maps.GoogleMap;
 8: import com.google.android.gms.maps.model.LatLng;
 9: import com.google.android.gms.maps.model.Polyline;
10: import com.google.android.gms.maps.model.PolylineOptions;
11:
12: public class MapPath {
13:         public interface PathSource {
14:                 public List<LatLng> getPathPoints();
15:         }
16:
17:         public Polyline missionPath;
18:         private GoogleMap mMap;
19:         private float width;
20:         private int color;
21:
22:         public MapPath(GoogleMap mMap, int color, float width) {
23:                 this.mMap = mMap;
24:                 this.color = color;
25:                 this.width = width;
26:         }
27:
28:         public MapPath(GoogleMap mMap) {
29:                 this(mMap, Color.YELLOW, 5);
30:         }
31:
32:         public void update(PathSource pathSource) {
33:                 addToMapIfNeeded();
34:                 List<LatLng> newPath = pathSource.getPathPoints();
35:                 missionPath.setPoints(newPath);
36:         }
37:
38:         private void addToMapIfNeeded() {
39:                 if (missionPath == null) {
40:                         PolylineOptions flightPath = new PolylineOptions();
41:                         flightPath.color(color).width(width);
42:                         missionPath = mMap.addPolyline(flightPath);
43:                 }
44:         }
45: }
```

```
    1: package com.droidplanner.fragments.helpers;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.graphics.Point;
    7:
    8: import com.google.android.gms.maps.GoogleMap;
    9: import com.google.android.gms.maps.Projection;
   10: import com.google.android.gms.maps.model.LatLng;
   11:
   12: public class MapProjection {
   13:
   14:         public static List<LatLng> projectPathIntoMap(List<Point> path,GoogleMap m
ap) {
   15:                 List<LatLng> coords = new ArrayList<LatLng>();
   16:                 Projection projection = map.getProjection();
   17:
   18:                 for (Point point : path) {
   19:                         coords.add(projection.fromScreenLocation(point));
   20:                 }
   21:
   22:                 return coords;
   23:         }
   24: }
```

```
    1: package com.droidplanner.fragments.helpers;
    2:
    3: import java.util.List;
    4:
    5: import android.content.Context;
    6: import android.content.SharedPreferences;
    7: import android.os.Bundle;
    8: import android.preference.PreferenceManager;
    9: import android.view.LayoutInflater;
   10: import android.view.View;
   11: import android.view.ViewGroup;
   12:
   13: import com.droidplanner.DroidPlannerApp;
   14: import com.droidplanner.drone.DroneInterfaces;
   15: import com.droidplanner.helpers.LocalMapTileProvider;
   16: import com.google.android.gms.maps.CameraUpdate;
   17: import com.google.android.gms.maps.CameraUpdateFactory;
   18: import com.google.android.gms.maps.GoogleMap;
   19: import com.google.android.gms.maps.MapFragment;
   20: import com.google.android.gms.maps.UiSettings;
   21: import com.google.android.gms.maps.model.LatLng;
   22: import com.google.android.gms.maps.model.LatLngBounds;
   23: import com.google.android.gms.maps.model.TileOverlay;
   24: import com.google.android.gms.maps.model.TileOverlayOptions;
   25:
   26: public class OfflineMapFragment extends MapFragment
   27:                 implements DroneInterfaces.MapConfigListener {
   28:
   29:         public static final String PREF_MAP_TYPE = "pref_map_type";
   30:
   31:         public static final String MAP_TYPE_SATELLITE = "Satellite";
   32:         public static final String MAP_TYPE_HYBRID = "Hybrid";
   33:         public static final String MAP_TYPE_NORMAL = "Normal";
   34:         public static final String MAP_TYPE_TERRAIN = "Terrain";
   35:
   36:         private GoogleMap mMap;
   37:
   38:         @Override
   39:         public View onCreateView(LayoutInflater inflater, ViewGroup viewGroup,
   40:                         Bundle bundle) {
   41:                 View view = super.onCreateView(inflater, viewGroup, bundle);
   42:
   43:                 ((DroidPlannerApp) getActivity().getApplication()).drone.setMapCon
figListener(this);
   44:
   45:                 setupMap();
   46:                 return view;
   47:         }
   48:
   49:         private void setupMap() {
   50:                 mMap = getMap();
   51:                 if (isMapLayoutFinished()) { // TODO it should wait for the map la
yout
   52:                                                                                 //
 before setting it up, instead of just
   53:                                                                                 //
 skipping the setup
   54:                         setupMapUI();
   55:                         setupMapOverlay();
   56:                 }
   57:         }
   58:
   59:         private void setupMapUI() {
   60:                 mMap.setMyLocationEnabled(true);
   61:                 UiSettings mUiSettings = mMap.getUiSettings();
   62:                 mUiSettings.setMyLocationButtonEnabled(true);
   63:                 mUiSettings.setCompassEnabled(true);
   64:                 mUiSettings.setTiltGesturesEnabled(false);
   65:         }
   66:
   67:         private void setupMapOverlay() {
   68:                 if (isOfflineMapEnabled()) {
   69:                         setupOfflineMapOverlay();
   70:                 } else {
   71:                         setupOnlineMapOverlay();
   72:                 }
   73:         }
   74:
   75:         private boolean isOfflineMapEnabled() {
   76:                 Context context = this.getActivity();
   77:                 SharedPreferences prefs = PreferenceManager
   78:                                 .getDefaultSharedPreferences(context);
   79:                 return prefs.getBoolean("pref_advanced_use_offline_maps", false);
   80:         }
   81:
   82:         private void setupOnlineMapOverlay() {
   83:                 mMap.setMapType(getMapType());
   84:         }
   85:
   86:         private int getMapType() {
   87:                 SharedPreferences prefs = PreferenceManager
   88:                                 .getDefaultSharedPreferences(getActivity());
   89:                 String mapType = prefs.getString(PREF_MAP_TYPE, "");
   90:
   91:                 if (mapType.equalsIgnoreCase(MAP_TYPE_SATELLITE)) {
   92:                         return GoogleMap.MAP_TYPE_SATELLITE;
   93:                 }
   94:                 if (mapType.equalsIgnoreCase(MAP_TYPE_HYBRID)) {
   95:                         return GoogleMap.MAP_TYPE_HYBRID;
   96:                 }
   97:                 if (mapType.equalsIgnoreCase(MAP_TYPE_NORMAL)) {
   98:                         return GoogleMap.MAP_TYPE_NORMAL;
   99:                 }
  100:                 if (mapType.equalsIgnoreCase(MAP_TYPE_TERRAIN)) {
  101:                         return GoogleMap.MAP_TYPE_TERRAIN;
  102:                 } else {
  103:                         return GoogleMap.MAP_TYPE_SATELLITE;
  104:                 }
  105:         }
  106:
  107:         private void setupOfflineMapOverlay() {
  108:                 mMap.setMapType(GoogleMap.MAP_TYPE_NONE);
  109:                 TileOverlay tileOverlay = mMap.addTileOverlay(new TileOverlayOptio
ns()
  110:                                                                                 .til
eProvider(new LocalMapTileProvider()));
  111:                 tileOverlay.setZIndex(-1);
  112:                 tileOverlay.clearTileCache();
  113:         }
  114:
  115:         public void zoomToExtents(List<LatLng> pointsList) {
  116:                 if (!pointsList.isEmpty()) {
  117:                         LatLngBounds bounds = getBounds(pointsList);
  118:                         CameraUpdate animation;
  119:                         if (isMapLayoutFinished())
  120:                                 animation = CameraUpdateFactory.newLatLngBounds(bo
unds, 100);
  121:                         else
  122:                                 animation = CameraUpdateFactory.newLatLngBounds(bo
unds, 480,
  123:                                                                         360, 100);
  124:                         getMap().animateCamera(animation);
  125:                 }
  126:         }
```

```
127:
128:         protected void clearMap() {
129:                 GoogleMap mMap = getMap();
130:                 mMap.clear();
131:                 setupMapOverlay();
132:         }
133:
134:         private LatLngBounds getBounds(List<LatLng> pointsList) {
135:                 LatLngBounds.Builder builder = new LatLngBounds.Builder();
136:                 for (LatLng point : pointsList) {
137:                         builder.include(point);
138:                 }
139:                 return builder.build();
140:         }
141:
142:         public double getMapRotation() {
143:                 if (isMapLayoutFinished()) {
144:                         return mMap.getCameraPosition().bearing;
145:                 } else {
146:                         return 0;
147:                 }
148:         }
149:
150:         private boolean isMapLayoutFinished() {
151:                 return getMap() != null;
152:         }
153:
154:         @Override
155:         public void onMapTypeChanged() {
156:                 setupMap();
157:         }
158: }
```

```
 1: package com.droidplanner.fragments.helpers;
 2:
 3: import com.droidplanner.drone.variables.mission.MissionItem;
 4: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
 5: import com.droidplanner.polygon.PolygonPoint;
 6: import com.google.android.gms.maps.model.LatLng;
 7:
 8: public interface OnMapInteractionListener {
 9:
10:         public void onAddPoint(LatLng point);
11:
12:         public void onMoveHome(LatLng coord);
13:
14:         public void onMoveWaypoint(SpatialCoordItem waypoint, LatLng latLng);
15:
16:         public void onMovingWaypoint(SpatialCoordItem source, LatLng latLng);
17:
18:         public void onMovePolygonPoint(PolygonPoint source, LatLng newCoord);
19:
20:         public void onMapClick(LatLng point);
21:
22:         public boolean onMarkerClick(MissionItem missionItem);
23: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import android.content.res.Resources;
    4: import android.graphics.Bitmap;
    5: import android.graphics.BitmapFactory;
    6: import android.graphics.Matrix;
    7:
    8: import com.MAVLink.Messages.enums.MAV_TYPE;
    9: import com.droidplanner.R.drawable;
   10: import com.google.android.gms.maps.model.BitmapDescriptor;
   11: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   12:
   13: public class DroneBitmaps {
   14:
   15:         public static final int DRONE_MIN_ROTATION = 5;
   16:         private BitmapDescriptor[] droneBitmaps;
   17:         private Resources resources;
   18:
   19:         public DroneBitmaps(Resources resources, int type) {
   20:                 this.resources = resources;
   21:                 buildBitmaps(type);
   22:         }
   23:
   24:         public BitmapDescriptor getIcon(double yaw) {
   25:                 int index = (int) (yaw / DRONE_MIN_ROTATION);
   26:                 return droneBitmaps[index];
   27:         }
   28:
   29:         private void buildBitmaps(int type) {
   30:                 int count = 360 / DRONE_MIN_ROTATION;
   31:                 droneBitmaps = new BitmapDescriptor[count];
   32:                 for (int i = 0; i < count; i++) {
   33:                         droneBitmaps[i] = generateIcon(i * DRONE_MIN_ROTATION, typ
e);
   34:                 }
   35:         }
   36:
   37:         private BitmapDescriptor generateIcon(float heading, int type) {
   38:                 Bitmap planeBitmap = getBitmap(type);
   39:                 Matrix matrix = new Matrix();
   40:                 matrix.postRotate(heading);
   41:                 return BitmapDescriptorFactory.fromBitmap(Bitmap.createBitmap(
   42:                                 planeBitmap, 0, 0, planeBitmap.getWidth(),
   43:                                 planeBitmap.getHeight(), matrix, true));
   44:         }
   45:
   46:         private Bitmap getBitmap(int type) {
   47:                 switch (type) {
   48:                 case MAV_TYPE.MAV_TYPE_GROUND_ROVER:
   49:                         return BitmapFactory.decodeResource(resources, drawable.ro
ver);
   50:                 case MAV_TYPE.MAV_TYPE_TRICOPTER:
   51:                 case MAV_TYPE.MAV_TYPE_QUADROTOR:
   52:                 case MAV_TYPE.MAV_TYPE_HEXAROTOR:
   53:                 case MAV_TYPE.MAV_TYPE_OCTOROTOR:
   54:                 case MAV_TYPE.MAV_TYPE_HELICOPTER:
   55:                         return BitmapFactory.decodeResource(resources, drawable.qu
ad);
   56:                 case MAV_TYPE.MAV_TYPE_FIXED_WING:
   57:                 default:
   58:                         return BitmapFactory.decodeResource(resources, drawable.pl
ane);
   59:                 }
   60:         }
   61: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import com.droidplanner.drone.DroneInterfaces.MapUpdatedListner;
    4: import com.droidplanner.fragments.FlightMapFragment;
    5: import com.google.android.gms.maps.CameraUpdateFactory;
    6: import com.google.android.gms.maps.model.LatLng;
    7: import com.google.android.gms.maps.model.Marker;
    8: import com.google.android.gms.maps.model.MarkerOptions;
    9:
   10: public class DroneMarker implements MapUpdatedListner {
   11:         private static final int ZOOM_LEVEL = 18;
   12:
   13:         private Marker droneMarker;
   14:         private FlightMapFragment flightMapFragment;
   15:         private DroneBitmaps bitmaps;
   16:
   17:         public DroneMarker(FlightMapFragment flightMapFragment) {
   18:                 this.flightMapFragment = flightMapFragment;
   19:                 updateDroneMarkers();
   20:         }
   21:
   22:         private void updatePosition(double yaw, LatLng coord) {
   23:                 // This ensure the 0 to 360 range
   24:                 double correctHeading = (yaw - flightMapFragment.getMapRotation()
+ 360) % 360;
   25:                 try {
   26:                         droneMarker.setVisible(true);
   27:                         droneMarker.setPosition(coord);
   28:                         droneMarker.setIcon(bitmaps.getIcon(correctHeading));
   29:
   30:                         animateCamera(coord);
   31:                 } catch (Exception e) {
   32:                 }
   33:         }
   34:
   35:         private void animateCamera(LatLng coord) {
   36:                 if (!flightMapFragment.hasBeenZoomed) {
   37:                         flightMapFragment.hasBeenZoomed = true;
   38:                         flightMapFragment.mMap.animateCamera(CameraUpdateFactory
   39:                                         .newLatLngZoom(coord, ZOOM_LEVEL));
   40:                 }
   41:                 if (flightMapFragment.isAutoPanEnabled) {
   42:                         flightMapFragment.mMap.animateCamera(CameraUpdateFactory
   43:                                         .newLatLngZoom(droneMarker.getPosition(),
ZOOM_LEVEL));
   44:                 }
   45:         }
   46:
   47:         public void updateDroneMarkers() {
   48:                 if (droneMarker!=null) {
   49:                         droneMarker.remove();
   50:                 }
   51:                 buildBitmaps();
   52:                 addMarkerToMap();
   53:         }
   54:
   55:         private void addMarkerToMap() {
   56:                 droneMarker = flightMapFragment.mMap.addMarker(new MarkerOptions()
   57:                                 .anchor((float) 0.5, (float) 0.5).position(new Lat
Lng(0, 0))
   58:                                 .icon(bitmaps.getIcon(0)).visible(false));
   59:         }
   60:
   61:         private void buildBitmaps() {
   62:                 bitmaps = new DroneBitmaps(flightMapFragment.getResources(),
   63:                                 flightMapFragment.drone.type.getType());
   64:         }
   65:
   66:         public void onDroneUpdate() {
   67:                 updatePosition(flightMapFragment.drone.orientation.getYaw(),
   68:                                 flightMapFragment.drone.GPS.getPosition());
   69:                 flightMapFragment.addFlithPathPoint(flightMapFragment.drone.GPS
   70:                                 .getPosition());
   71:         }
   72:
   73:         @Override
   74:         public void onDroneTypeChanged() {
   75:                 updateDroneMarkers();
   76:         }
   77: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import com.droidplanner.R;
    4: import com.droidplanner.gcp.Gcp;
    5: import com.google.android.gms.maps.model.BitmapDescriptor;
    6: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
    7: import com.google.android.gms.maps.model.Marker;
    8: import com.google.android.gms.maps.model.MarkerOptions;
    9:
   10: public class GcpMarker {
   11:
   12:         public static MarkerOptions build(Gcp gcp) {
   13:                 return new MarkerOptions().position(gcp.coord).title(String.valueO
f(0))
   14:                                 .icon(getIcon(gcp)).anchor((float) 0.5, (float) 0.
5);
   15:         }
   16:
   17:         public static void update(Marker marker, Gcp gcp) {
   18:                 marker.setPosition(gcp.coord);
   19:                 marker.setTitle(String.valueOf(0));
   20:                 marker.setIcon(getIcon(gcp));
   21:         }
   22:
   23:         private static BitmapDescriptor getIcon(Gcp gcp) {
   24:                 if (gcp.isMarked) {
   25:                         return BitmapDescriptorFactory
   26:                                         .fromResource(R.drawable.placemark_circle_
red);
   27:                 } else {
   28:                         return BitmapDescriptorFactory
   29:                                         .fromResource(R.drawable.placemark_circle_
blue);
   30:                 }
   31:         }
   32: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import com.google.android.gms.maps.model.LatLng;
    4: import com.google.android.gms.maps.model.MarkerOptions;
    5:
    6: public class GenericMarker {
    7:
    8:         public static MarkerOptions build(LatLng coord) {
    9:                 return new MarkerOptions().position(coord).draggable(true).anchor(
0.5f, 0.5f);
   10:         }
   11:
   12: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3:
    4: import android.content.Context;
    5:
    6: import com.droidplanner.R;
    7: import com.droidplanner.drone.variables.GuidedPoint;
    8: import com.droidplanner.fragments.markers.helpers.MarkerWithText;
    9: import com.droidplanner.helpers.units.Altitude;
   10: import com.google.android.gms.maps.model.BitmapDescriptor;
   11: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
   12: import com.google.android.gms.maps.model.Marker;
   13: import com.google.android.gms.maps.model.MarkerOptions;
   14:
   15: public class GuidedMarker {
   16:         public static MarkerOptions build(GuidedPoint guidedPoint, Altitude altitu
de, Context context) {
   17:                 return new MarkerOptions()
   18:                                 .position(guidedPoint.getCoord())
   19:                                 .icon(getIcon(guidedPoint, altitude, context))
   20:                                 .anchor(0.5f, 0.5f);
   21:         }
   22:
   23:         public static void update(Marker marker, GuidedPoint guidedPoint, Altitude
 altitude, Context context) {
   24:                 if (guidedPoint.isValid()) {
   25:                         marker.setPosition(guidedPoint.getCoord());
   26:                         marker.setIcon(getIcon(guidedPoint, altitude, context));
   27:                         marker.setVisible(true);
   28:                 }else{
   29:                         marker.setVisible(false);
   30:                 }
   31:         }
   32:
   33:         private static BitmapDescriptor getIcon(GuidedPoint guidedPoint, Altitude
altitude, Context context)
   34:         {
   35:                 return BitmapDescriptorFactory.fromBitmap(MarkerWithText.getMarker
WithTextAndDetail(R.drawable.ic_wp_map_on,
   36:                         "Guided",  "", context));
   37:         }
   38: }
```

```
    1: package com.droidplanner.fragments.markers.helpers;
    2:
    3: import android.content.Context;
    4: import android.content.res.Resources;
    5: import android.graphics.*;
    6:
    7: import com.droidplanner.R;
    8:
    9: public class MarkerWithText {
   10:
   11:         private static final int RECT_PADDING = 6;
   12:
   13:
   14:         public static Bitmap getMarkerWithText(int color, String text, Context con
text) {
   15:                 return drawTextToBitmap(context, R.drawable.ic_marker_white, color
, text);
   16:         }
   17:
   18:         /**
   19:          * Copied from:
   20:          * http://stackoverflow.com/questions/18335642/how-to-draw-text-in-default
-marker-of-google-map-v2?lq=1
   21:          */
   22:         private static Bitmap drawTextToBitmap(Context gContext, int gResId,
   23:                                                int color, String gText) {
   24:                 Resources resources = gContext.getResources();
   25:                 float scale = resources.getDisplayMetrics().density;
   26:                 Bitmap bitmap = BitmapFactory.decodeResource(resources, gResId);
   27:
   28:                 android.graphics.Bitmap.Config bitmapConfig = bitmap.getConfig();
   29:                 if (bitmapConfig == null) {
   30:                         bitmapConfig = android.graphics.Bitmap.Config.ARGB_8888;
   31:                 }
   32:                 bitmap = bitmap.copy(bitmapConfig, true);
   33:
   34:                 // copy bitmap to canvas, replace white with colour
   35:                 Paint paint = new Paint();
   36:                 paint.setColorFilter(new LightingColorFilter(0x000000, color));
   37:                 Canvas canvas = new Canvas(bitmap);
   38:                 canvas.drawBitmap(bitmap, 0, 0, paint);
   39:
   40:                 paint = new Paint(Paint.ANTI_ALIAS_FLAG);
   41:                 paint.setColor(Color.BLACK);
   42:                 paint.setTextSize((int) (15 * scale));
   43:                 paint.setShadowLayer(1f, 0f, 1f, Color.WHITE);
   44:
   45:                 Rect bounds = new Rect();
   46:                 paint.getTextBounds(gText, 0, gText.length(), bounds);
   47:                 int x = (bitmap.getWidth() - bounds.width()) / 2;
   48:                 int y = (bitmap.getHeight() + bounds.height()) * 5/12; // At 5/12
from the top so it stays on the center
   49:
   50:                 canvas.drawText(gText, x, y, paint);
   51:
   52:                 return bitmap;
   53:         }
   54:
   55:
   56:         public static Bitmap getMarkerWithTextAndDetail(int gResId, String text, S
tring detail, Context context) {
   57:                 return drawTextAndDetailToBitmap(context, gResId, text, detail);
   58:         }
   59:
   60:         /**
   61:          * Based on:
   62:          * http://stackoverflow.com/questions/18335642/how-to-draw-text-in-default
-marker-of-google-map-v2?lq=1
   63:          */
   64:         private static Bitmap drawTextAndDetailToBitmap(Context gContext, int gRes
Id, String gText, String gDetail) {
   65:                 Resources resources = gContext.getResources();
   66:                 float scale = resources.getDisplayMetrics().density;
   67:                 Bitmap bitmap = BitmapFactory.decodeResource(resources, gResId);
   68:
   69:                 android.graphics.Bitmap.Config bitmapConfig = bitmap.getConfig();
   70:                 if (bitmapConfig == null) {
   71:                         bitmapConfig = android.graphics.Bitmap.Config.ARGB_8888;
   72:                 }
   73:                 bitmap = bitmap.copy(bitmapConfig, true);
   74:
   75:                 // copy bitmap to canvas, replace white with colour
   76:                 Paint paint = new Paint();
   77:                 Canvas canvas = new Canvas(bitmap);
   78:                 canvas.drawBitmap(bitmap, 0, 0, paint);
   79:
   80:                 paint = new Paint(Paint.ANTI_ALIAS_FLAG);
   81:                 paint.setColor(Color.BLACK);
   82:                 paint.setTextSize((int) (15 * scale));
   83:                 paint.setFakeBoldText(true);
   84:                 paint.setShadowLayer(1f, 0f, 1f, Color.TRANSPARENT);
   85:
   86:                 Rect bounds = new Rect();
   87:                 paint.getTextBounds(gText, 0, gText.length(), bounds);
   88:                 bounds.offsetTo(0, bounds.height() / 2);
   89:
   90:                 // paint and bounds for details
   91:                 Paint dpaint = new Paint(Paint.ANTI_ALIAS_FLAG);
   92:                 dpaint.setColor(Color.BLACK);
   93:                 dpaint.setTextSize((int) (10 * scale));
   94:                 paint.setFakeBoldText(true);
   95:                 dpaint.setShadowLayer(1f, 0f, 1f, Color.WHITE);
   96:
   97:                 Rect dbounds = new Rect();
   98:                 dpaint.getTextBounds(gDetail, 0, gDetail.length(), dbounds);
   99:                 dbounds.offsetTo(0, bounds.bottom + 2);
  100:
  101:
  102:                 // include text and detail bounds
  103:                 Rect brect = new Rect(bounds);
  104:                 brect.union(dbounds);
  105:
  106:                 // position and inflate w/ padding
  107:                 int x = (bitmap.getWidth() - brect.width()) / 2;
  108:                 int y = bounds.top + (bitmap.getHeight() - brect.height()) / 2;
  109:                 brect.offsetTo(x, y - (bounds.height()));
  110:                 brect.set(brect.left - RECT_PADDING, brect.top - RECT_PADDING, bre
ct.right + RECT_PADDING, brect.bottom + RECT_PADDING);
  111:
  112:
  113:                 // draw text
  114:                 x = (bitmap.getWidth() - bounds.width()) / 2;
  115:                 y = bounds.top + (bitmap.getHeight() - (bounds.height() + dbounds.
height())) / 2;
  116:                 canvas.drawText(gText, x, y, paint);
  117:
  118:                 // draw detail
  119:                 x = (bitmap.getWidth() - dbounds.width()) / 2;
  120:                 y = y + bounds.height() + 2;
  121:                 canvas.drawText(gDetail, x, y, dpaint);
  122:
  123:                 return bitmap;
  124:         }
  125: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import com.droidplanner.R.drawable;
    4: import com.droidplanner.drone.variables.Home;
    5: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
    6: import com.google.android.gms.maps.model.Marker;
    7: import com.google.android.gms.maps.model.MarkerOptions;
    8:
    9: public class HomeMarker {
   10:         public static MarkerOptions build(Home home) {
   11:                 return new MarkerOptions()
   12:                                 .position(home.getCoord())
   13:                                 .visible(home.isValid())
   14:                                 .title("Home")
   15:                                 .snippet(home.getAltitude().toString())
   16:                                 .anchor((float) 0.5, (float) 0.5)
   17:                                 .icon(BitmapDescriptorFactory
   18:                                                 .fromResource(drawable.ic_menu_hom
e)).title("Home");
   19:         }
   20:
   21:         public static void update(Marker marker, Home home) {
   22:                 marker.setVisible(home.isValid());
   23:                 marker.setPosition(home.getCoord());
   24:                 marker.setSnippet("Home "+ home.getAltitude());
   25:         }
   26:
   27: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import java.util.ArrayList;
    4: import java.util.HashMap;
    5: import java.util.List;
    6:
    7: import android.content.Context;
    8:
    9: import com.google.android.gms.maps.GoogleMap;
   10: import com.google.android.gms.maps.model.Marker;
   11: import com.google.android.gms.maps.model.MarkerOptions;
   12:
   13: public class MarkerManager {
   14:         public GoogleMap mMap;
   15:         public HashMap<Marker, MarkerSource> hashMap = new HashMap<Marker, MarkerS
ource>();
   16:
   17:         public interface MarkerSource {
   18:                 MarkerOptions build(Context context);
   19:
   20:                 void update(Marker marker, Context context);
   21:         }
   22:
   23:         public MarkerManager(GoogleMap map) {
   24:                 this.mMap = map;
   25:         }
   26:
   27:         public void clean() {
   28:                 List<MarkerSource> emptyList = new ArrayList<MarkerSource>();
   29:                 removeOldMarkers(emptyList);
   30:         }
   31:
   32:         public <T> void updateMarkers(List<T> list, boolean draggable,
   33:                         Context context) {
   34:                 for (T object : list) {
   35:                         updateMarker((MarkerSource) object, draggable, context);
   36:                 }
   37:         }
   38:
   39:         public void updateMarker(MarkerSource source, boolean draggable,
   40:                         Context context) {
   41:
   42:                 if (hashMap.containsValue(source)) {
   43:                         Marker marker = getMarkerFromSource(source);
   44:                         source.update(marker, context);
   45:                         marker.setDraggable(draggable);
   46:                 } else {
   47:                         addMarker(source, draggable, context);
   48:                 }
   49:         }
   50:
   51:         private <T> void removeOldMarkers(List<T> list) {
   52:                 List<MarkerSource> toRemove = new ArrayList<MarkerSource>();
   53:                 for (Marker marker : hashMap.keySet()) {
   54:                         MarkerSource object = getSourceFromMarker(marker);
   55:                         if (!list.contains(object)) {
   56:                                 toRemove.add(object);
   57:                         }
   58:                 }
   59:                 for (MarkerSource markerSource : toRemove) {
   60:                         removeMarker(markerSource);
   61:                 }
   62:         }
   63:
   64:         private boolean removeMarker(MarkerSource object) {
   65:                 if (hashMap.containsValue(object)) {
   66:                         Marker marker = getMarkerFromSource(object);
   67:                         hashMap.remove(marker);
   68:                         marker.remove();
   69:                         return true;
   70:                 } else {
   71:                         return false;
   72:                 }
   73:         }
   74:
   75:         private void addMarker(MarkerSource object, boolean draggable,
   76:                         Context context) {
   77:                 Marker marker = mMap.addMarker(object.build(context));
   78:                 marker.setDraggable(draggable);
   79:                 hashMap.put(marker, object);
   80:         }
   81:
   82:         public Marker getMarkerFromSource(MarkerSource object) {
   83:                 for (Marker marker : hashMap.keySet()) {
   84:                         if (getSourceFromMarker(marker) == object) {
   85:                                 return marker;
   86:                         }
   87:                 }
   88:                 return null;
   89:         }
   90:
   91:         public MarkerSource getSourceFromMarker(Marker marker) {
   92:                 return hashMap.get(marker);
   93:         }
   94:
   95: }
```

```
    1: package com.droidplanner.fragments.markers;
    2:
    3: import com.droidplanner.polygon.PolygonPoint;
    4: import com.google.android.gms.maps.model.BitmapDescriptorFactory;
    5: import com.google.android.gms.maps.model.Marker;
    6: import com.google.android.gms.maps.model.MarkerOptions;
    7:
    8: public class PolygonMarker {
    9:
   10:         public static MarkerOptions build(PolygonPoint wp) {
   11:                 return new MarkerOptions()
   12:                                 .position(wp.coord)
   13:                                 .draggable(true)
   14:                                 .title("Poly")
   15:                                 // TODO fix constant
   16:                                 .icon(BitmapDescriptorFactory
   17:                                                 .defaultMarker(BitmapDescriptorFac
tory.HUE_BLUE));
   18:         }
   19:
   20:         public static void update(Marker marker, PolygonPoint wp) {
   21:                 marker.setPosition(wp.coord);
   22:                 marker.setTitle("Poly");// TODO fix constant
   23:                 marker.setIcon(BitmapDescriptorFactory
   24:                                 .defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
   25:         }
   26:
   27: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.content.Context;
    4: import android.view.View;
    5: import android.view.ViewGroup;
    6: import android.widget.ArrayAdapter;
    7: import android.widget.TextView;
    8:
    9: class AdapterMissionItens extends ArrayAdapter<MissionItemTypes> {
   10:
   11:         public AdapterMissionItens(Context context, int resource,
   12:                         MissionItemTypes[] objects) {
   13:                 super(context, resource, objects);
   14:         }
   15:
   16:         @Override
   17:         public View getView(int position, View convertView, ViewGroup parent) {
   18:                 View view = super.getView(position, convertView, parent);
   19:                 ((TextView) view).setText(getItem(position).getName());
   20:                 return view;
   21:         }
   22:
   23:         @Override
   24:         public View getDropDownView(int position, View convertView, ViewGroup pare
nt) {
   25:                 return getView(position, convertView, parent);
   26:         }
   27:
   28: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.app.Activity;
    4: import android.app.Fragment;
    5: import android.os.Bundle;
    6: import android.util.Log;
    7: import android.view.LayoutInflater;
    8: import android.view.View;
;
    9: import android.view.ViewGroup;
   10: import android.widget.AdapterView;
   11: import android.widget.AdapterView.OnItemSelectedListener;
   12:
   13: import com.droidplanner.DroidPlannerApp;
   14: import com.droidplanner.R;
   15: import com.droidplanner.drone.variables.mission.Mission;
   16: import com.droidplanner.drone.variables.mission.MissionItem;
   17: import com.droidplanner.fragments.mission.MissionItemTypes.InvalidItemException;
   18: import com.droidplanner.widgets.spinners.SpinnerSelfSelect;
   19:
   20: public abstract class MissionDetailFragment extends Fragment implements
   21:                 OnItemSelectedListener {
   22:
   23:         public interface OnWayPointTypeChangeListener{
   24:                 public void onWaypointTypeChanged(MissionItem newItem, MissionItem
 oldItem);
   25:         }
   26:
   27:         protected abstract int getResource();
   28:
   29:         protected SpinnerSelfSelect typeSpinner;
   30:         protected AdapterMissionItens commandAdapter;
   31:         protected Mission mission;
   32:         private OnWayPointTypeChangeListener mListner;
   33:
   34:         protected MissionItem item;
   35:
   36:
   37:         @Override
   38:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   39:                         Bundle savedInstanceState) {
   40:                 View view = inflater.inflate(getResource(), null);
   41:                 setupViews(view);
   42:                 return view;
   43:         }
   44:
   45:         protected void setupViews(View view) {
   46:                 typeSpinner = (SpinnerSelfSelect) view
   47:                                 .findViewById(R.id.spinnerWaypointType);
   48:                 commandAdapter = new AdapterMissionItens(this.getActivity(),
   49:                                 android.R.layout.simple_list_item_1, MissionItemTy
pes.values());
   50:                 typeSpinner.setAdapter(commandAdapter);
   51:                 typeSpinner.setOnItemSelectedListener(this);
   52:         }
   53:
   54:         @Override
   55:         public void onAttach(Activity activity) {
   56:                 super.onAttach(activity);
   57:                 mission = ((DroidPlannerApp) getActivity().getApplication()).drone
.mission;
   58:                 mListner = (OnWayPointTypeChangeListener) activity;
   59:         }
   60:
   61:
   62:         @Override
   63:         public void onItemSelected(AdapterView<?> arg0, View v, int position,
   64:                         long id) {
   65:
   66:                 MissionItemTypes selected = commandAdapter.getItem(position);
   67:                 try {
   68:                         MissionItem newItem = selected.getNewItem(getItem());
   69:                         if (!newItem.getClass().equals(getItem().getClass())) {
   70:                                 Log.d("CLASS", "Diferent waypoint Classes");
   71:                                 mListner.onWaypointTypeChanged(newItem, getItem())
   72:                         }
   73:                 } catch (InvalidItemException e) {
   74:                 }
   75:         }
   76:
   77:         @Override
   78:         public void onNothingSelected(AdapterView<?> arg0) {
   79:         }
   80:
   81:         public MissionItem getItem() {
   82:                 return item;
   83:         }
   84:
   85:         public void setItem(MissionItem item) {
   86:                 this.item = item;
   87:         }
   88:
   89: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import java.security.InvalidParameterException;
    4:
    5: import com.droidplanner.drone.variables.mission.MissionItem;
    6: import com.droidplanner.drone.variables.mission.commands.ReturnToHome;
    7: import com.droidplanner.drone.variables.mission.survey.Survey;
    8: import com.droidplanner.drone.variables.mission.waypoints.Land;
    9: import com.droidplanner.drone.variables.mission.waypoints.LoiterInfinite;
   10: import com.droidplanner.drone.variables.mission.waypoints.LoiterTime;
   11: import com.droidplanner.drone.variables.mission.waypoints.LoiterTurns;
   12: import com.droidplanner.drone.variables.mission.waypoints.RegionOfInterest;
   13: import com.droidplanner.drone.variables.mission.waypoints.Takeoff;
   14: import com.droidplanner.drone.variables.mission.waypoints.Waypoint;
   15:
   16: public enum MissionItemTypes {
   17:         WAYPOINT("Waypoint"), LOITER("Loiter"), LOITERN("LoiterN"), LOITERT(
   18:                         "LoiterT"), RTL("RTL"), LAND("Land"), TAKEOFF("Takeoff"),
ROI("ROI"), SURVEY("Survey");
   19:
   20:         private final String name;
   21:
   22:         private MissionItemTypes(String name) {
   23:                 this.name = name;
   24:         }
   25:
   26:         public String getName() {
   27:                 return name;
   28:         }
   29:
   30:         public MissionItem getNewItem(MissionItem item) throws InvalidItemExceptio
n {
   31:                 switch (this) {
   32:                 case LAND:
   33:                         return new Land(item);
   34:                 case LOITER:
   35:                         return new LoiterInfinite(item);
   36:                 case LOITERN:
   37:                         return new LoiterTurns(item);
   38:                 case LOITERT:
   39:                         return new LoiterTime(item);
   40:                 case ROI:
   41:                         return new RegionOfInterest(item);
   42:                 case RTL:
   43:                         return new ReturnToHome(item);
   44:                 case TAKEOFF:
   45:                         return new Takeoff(item);
   46:                 case WAYPOINT:
   47:                         return new Waypoint(item);
   48:                 case SURVEY:
   49:                         throw new InvalidItemException();
   50:                 }
   51:                 throw new InvalidParameterException();
   52:         }
   53:
   54:         class InvalidItemException extends Exception{
   55:                 private static final long serialVersionUID = 1L;
   56:
   57:         }
   58: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4:
    5: import com.droidplanner.R;
    6: import com.droidplanner.drone.variables.mission.waypoints.Land;
    7: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
    8: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
    9:
   10: public class MissionLandFragment extends MissionDetailFragment implements
   11:                 OnTextSeekBarChangedListner {
   12:
   13:         private SeekBarWithText yawSeekBar;
   14:
   15:         @Override
   16:         protected int getResource() {
   17:                 return R.layout.fragment_detail_land;
   18:         }
   19:
   20:         @Override
   21:         protected void setupViews(View view) {
   22:                 super.setupViews(view);
   23:                 Land item = (Land) this.item;
   24:                 typeSpinner.setSelection(commandAdapter
   25:                                 .getPosition(MissionItemTypes.LAND));
   26:                 yawSeekBar = (SeekBarWithText) view.findViewById(R.id.waypointAngl
e);
   27:                 yawSeekBar.setValue(item.getYawAngle());
   28:                 yawSeekBar.setOnChangedListner(this);
   29:         }
   30:
   31:         @Override
   32:         public void onSeekBarChanged() {
   33:                 Land item = (Land) this.item;
   34:                 item.setYawAngle((float) yawSeekBar.getValue());
   35:         }
   36:
   37: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4: import android.widget.CheckBox;
    5: import android.widget.CompoundButton;
    6: import android.widget.CompoundButton.OnCheckedChangeListener;
    7:
    8: import com.droidplanner.R;
    9: import com.droidplanner.drone.variables.mission.waypoints.Loiter;
   10: import com.droidplanner.drone.variables.mission.waypoints.LoiterInfinite;
   11: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
   12: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   13:
   14: public class MissionLoiterFragment extends MissionDetailFragment implements
   15:                 OnTextSeekBarChangedListner, OnCheckedChangeListener{
   16:
   17:         private SeekBarWithText loiterRadiusSeekBar;
   18:         private CheckBox loiterCCW;
   19:         private SeekBarWithText yawSeekBar;
   20:
   21:         @Override
   22:         protected int getResource() {
   23:                 return R.layout.fragment_detail_loiter;
   24:         }
   25:
   26:
   27:         @Override
   28:         protected void setupViews(View view) {
   29:                 super.setupViews(view);
   30:                 typeSpinner.setSelection(commandAdapter.getPosition(MissionItemTyp
es.LOITER));
   31:
   32:                 LoiterInfinite item = (LoiterInfinite) this.item;
   33:
   34:
   35:                 loiterCCW = (CheckBox) view.findViewById(R.string.loiter_ccw);
   36:                 loiterCCW.setChecked(item.isOrbitCCW());
   37:                 loiterCCW.setOnCheckedChangeListener(this);
   38:
   39:
   40:                 loiterRadiusSeekBar = (SeekBarWithText) view
   41:                                 .findViewById(R.id.loiterRadius);
   42:                 loiterRadiusSeekBar .setOnChangedListner(this);
   43:                 loiterRadiusSeekBar.setAbsValue(item.getOrbitalRadius());
   44:
   45:                 yawSeekBar = (SeekBarWithText) view
   46:                                 .findViewById(R.id.waypointAngle);
   47:                 yawSeekBar.setValue(item.getYawAngle());
   48:                 yawSeekBar.setOnChangedListner(this);
   49:         }
   50:
   51:
   52:
   53:         @Override
   54:     public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
   55:                 ((Loiter) item).setOrbitCCW(isChecked);
   56:     }
   57:
   58:         @Override
   59:         public void onSeekBarChanged() {
   60:                 ((Loiter) item).setOrbitalRadius(loiterRadiusSeekBar.getValue());
   61:                 ((Loiter) item).setYawAngle(yawSeekBar.getValue());
   62:         }
   63: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4: import android.widget.CheckBox;
    5: import android.widget.CompoundButton;
    6: import android.widget.CompoundButton.OnCheckedChangeListener;
    7:
    8: import com.droidplanner.R;
    9: import com.droidplanner.drone.variables.mission.waypoints.Loiter;
   10: import com.droidplanner.drone.variables.mission.waypoints.LoiterTurns;
   11: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
   12: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   13:
   14: public class MissionLoiterNFragment extends MissionDetailFragment implements
   15:                 OnTextSeekBarChangedListner, OnCheckedChangeListener {
   16:
   17:
   18:         private SeekBarWithText altitudeSeekBar;
   19:         private SeekBarWithText loiterTurnSeekBar;
   20:         private SeekBarWithText loiterRadiusSeekBar;
   21:         private CheckBox loiterCCW;
   22:         private SeekBarWithText yawSeekBar;
   23:
   24:         @Override
   25:         protected int getResource() {
   26:                 return R.layout.fragment_detail_loitern;
   27:         }
   28:
   29:         @Override
   30:         protected void setupViews(View view) {
   31:                 super.setupViews(view);
   32:                 typeSpinner.setSelection(commandAdapter.getPosition(MissionItemTyp
es.LOITERN));
   33:
   34:                 LoiterTurns item = (LoiterTurns) this.item;
   35:                 loiterCCW = (CheckBox) view.findViewById(R.string.loiter_ccw);
   36:                 loiterCCW.setChecked(item.isOrbitCCW());
   37:                 loiterCCW.setOnCheckedChangeListener(this);
   38:
   39:                 altitudeSeekBar = (SeekBarWithText) view
   40:                                 .findViewById(R.id.altitudeView);
   41:                 altitudeSeekBar.setValue(item.getAltitude().valueInMeters());
   42:                 altitudeSeekBar.setOnChangedListner(this);
   43:
   44:                 loiterTurnSeekBar = (SeekBarWithText) view
   45:                                 .findViewById(R.id.loiterTurn);
   46:                 loiterTurnSeekBar.setOnChangedListner(this);
   47:                 loiterTurnSeekBar.setValue(item.getTurns());
   48:
   49:                 loiterRadiusSeekBar = (SeekBarWithText) view
   50:                                 .findViewById(R.id.loiterRadius);
   51:                 loiterRadiusSeekBar.setAbsValue(item.getOrbitalRadius());
   52:                 loiterRadiusSeekBar .setOnChangedListner(this);
   53:
   54:                 yawSeekBar = (SeekBarWithText) view
   55:                                 .findViewById(R.id.waypointAngle);
   56:                 yawSeekBar.setValue(item.getYawAngle());
   57:                 yawSeekBar.setOnChangedListner(this);
   58:         }
   59:
   60:
   61:     @Override
   62:     public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
   63:                 ((Loiter) item).setOrbitCCW(isChecked);
   64:     }
   65:
   66:
   67:         @Override
   68:         public void onSeekBarChanged() {
   69:                 LoiterTurns item = (LoiterTurns) this.item;
   70:                 item.getAltitude().set(altitudeSeekBar.getValue());
   71:                 item.setTurns((int)loiterTurnSeekBar.getValue());
   72:                 item.setOrbitalRadius(loiterRadiusSeekBar.getValue());
   73:                 item.setYawAngle(yawSeekBar.getValue());
   74:         }
   75:
   76: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4: import android.widget.CheckBox;
    5: import android.widget.CompoundButton;
    6: import android.widget.CompoundButton.OnCheckedChangeListener;
    7:
    8: import com.droidplanner.R;
    9: import com.droidplanner.drone.variables.mission.waypoints.Loiter;
   10: import com.droidplanner.drone.variables.mission.waypoints.LoiterTime;
   11: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
   12: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   13:
   14: public class MissionLoiterTFragment extends MissionDetailFragment implements
   15:                 OnTextSeekBarChangedListner, OnCheckedChangeListener {
   16:         private SeekBarWithText altitudeSeekBar;
   17:         private SeekBarWithText loiterTimeSeekBar;
   18:         private SeekBarWithText loiterRadiusSeekBar;
   19:         private CheckBox loiterCCW;
   20:         private SeekBarWithText yawSeekBar;
   21:
   22:         @Override
   23:         protected int getResource() {
   24:                 return R.layout.fragment_detail_loitert;
   25:         }
   26:
   27:
   28:         @Override
   29:         protected void setupViews(View view) {
   30:                 super.setupViews(view);
   31:                 typeSpinner.setSelection(commandAdapter.getPosition(MissionItemTyp
es.LOITERT));
   32:
   33:                 LoiterTime item = (LoiterTime) this.item;
   34:
   35:                 loiterCCW = (CheckBox) view.findViewById(R.string.loiter_ccw);
   36:                 loiterCCW.setChecked(item.isOrbitCCW());
   37:                 loiterCCW.setOnCheckedChangeListener(this);
   38:
   39:
   40:                 altitudeSeekBar = (SeekBarWithText) view
   41:                                 .findViewById(R.id.altitudeView);
   42:                 altitudeSeekBar.setValue(item.getAltitude().valueInMeters());
   43:                 altitudeSeekBar.setOnChangedListner(this);
   44:
   45:                 loiterTimeSeekBar = (SeekBarWithText) view
   46:                                 .findViewById(R.id.loiterTime);
   47:                 loiterTimeSeekBar .setOnChangedListner(this);
   48:                 loiterTimeSeekBar.setValue(item.getTime());
   49:
   50:                 loiterRadiusSeekBar = (SeekBarWithText) view
   51:                                 .findViewById(R.id.loiterRadius);
   52:                 loiterRadiusSeekBar.setAbsValue(item.getOrbitalRadius());
   53:                 loiterRadiusSeekBar .setOnChangedListner(this);
   54:
   55:                 yawSeekBar = (SeekBarWithText) view
   56:                                 .findViewById(R.id.waypointAngle);
   57:                 yawSeekBar.setValue(item.getYawAngle());
   58:                 yawSeekBar.setOnChangedListner(this);
   59:         }
   60:
   61:     @Override
   62:     public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
   63:                 ((Loiter) item).setOrbitCCW(isChecked);
   64:     }
   65:
   66:
   67:         @Override
   68:         public void onSeekBarChanged() {
   69:                 LoiterTime item = (LoiterTime) this.item;
   70:                 item.getAltitude().set(altitudeSeekBar.getValue());
   71:                 item.setTime(loiterTimeSeekBar.getValue());
   72:                 item.setOrbitalRadius(loiterRadiusSeekBar.getValue());
   73:                 item.setYawAngle(yawSeekBar.getValue());
   74:         }
   75:
   76:
   77: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4:
    5: import com.droidplanner.R;
    6: import com.droidplanner.drone.variables.mission.waypoints.RegionOfInterest;
    7: import com.droidplanner.helpers.units.Altitude;
    8: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
    9: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   10:
   11: public class MissionRegionOfInterestFragment extends MissionDetailFragment
   12:                 implements OnTextSeekBarChangedListner {
   13:
   14:         private SeekBarWithText altitudeSeekBar;
   15:
   16:         @Override
   17:         protected int getResource() {
   18:                 return R.layout.fragment_detail_roi;
   19:         }
   20:
   21:         @Override
   22:         protected void setupViews(View view) {
   23:                 super.setupViews(view);
   24:                 typeSpinner.setSelection(commandAdapter
   25:                                 .getPosition(MissionItemTypes.ROI));
   26:
   27:                 altitudeSeekBar = (SeekBarWithText) view
   28:                                 .findViewById(R.id.altitudeView);
   29:                 altitudeSeekBar.setValue(((RegionOfInterest) item).getAltitude()
   30:                                 .valueInMeters());
   31:                 altitudeSeekBar.setOnChangedListner(this);
   32:         }
   33:
   34:         @Override
   35:         public void onSeekBarChanged() {
   36:                 ((RegionOfInterest) item).setAltitude(new Altitude(altitudeSeekBar
   37:                                 .getValue()));
   38:         }
   39:
   40: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4:
    5: import com.droidplanner.R;
    6: import com.droidplanner.drone.variables.mission.commands.ReturnToHome;
    7: import com.droidplanner.helpers.units.Altitude;
    8: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
    9: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   10:
   11: public class MissionRTLFragment extends MissionDetailFragment implements
   12:                 OnTextSeekBarChangedListner {
   13:         private SeekBarWithText altitudeSeekBar;
   14:
   15:         @Override
   16:         protected int getResource() {
   17:                 return R.layout.fragment_detail_rtl;
   18:         }
   19:
   20:
   21:         @Override
   22:         protected void setupViews(View view) {
   23:                 super.setupViews(view);
   24:                 typeSpinner.setSelection(commandAdapter.getPosition(MissionItemTyp
es.RTL));
   25:
   26:                 altitudeSeekBar = (SeekBarWithText) view
   27:                                 .findViewById(R.id.altitudeView);
   28:                 altitudeSeekBar.setValue(((ReturnToHome) item).getHeight().valueIn
Meters());
   29:                 altitudeSeekBar.setOnChangedListner(this);
   30:         }
   31:
   32:         @Override
   33:         public void onSeekBarChanged() {
   34:                 ((ReturnToHome) item).setHeight(new Altitude(altitudeSeekBar.getVa
lue()));
   35:         }
   36:
   37: }
```

```
 1: package com.droidplanner.fragments.mission;
 2:
 3: import android.view.View;
 4:
 5: import com.droidplanner.R;
 6:
 7: public class MissionSurveyFragment extends MissionDetailFragment {
 8:
 9:         @Override
10:         protected int getResource() {
11:                 return R.layout.fragment_detail_survey;
12:         }
13:
14:         @Override
15:         protected void setupViews(View view) {
16:                 super.setupViews(view);
17:                 typeSpinner.setSelection(commandAdapter
18:                                 .getPosition(MissionItemTypes.SURVEY));
19:         }
20: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4:
    5: import com.droidplanner.R;
    6: import com.droidplanner.drone.variables.mission.waypoints.Takeoff;
    7: import com.droidplanner.helpers.units.Altitude;
    8: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
    9: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   10:
   11: public class MissionTakeoffFragment extends MissionDetailFragment implements
   12:                 OnTextSeekBarChangedListner {
   13:         private SeekBarWithText altitudeSeekBar;
   14:         private SeekBarWithText angleSeekBar;
   15:         private SeekBarWithText yawSeekBar;
   16:
   17:         @Override
   18:         protected int getResource() {
   19:                 return R.layout.fragment_detail_takeoff;
   20:         }
   21:
   22:         @Override
   23:         protected void setupViews(View view) {
   24:                 super.setupViews(view);
   25:                 typeSpinner.setSelection(commandAdapter.getPosition(MissionItemTyp
es.TAKEOFF));
   26:
   27:                 Takeoff item = (Takeoff) this.item;
   28:
   29:                 altitudeSeekBar = (SeekBarWithText) view
   30:                                 .findViewById(R.id.altitudeView);
   31:                 altitudeSeekBar.setValue(item.getAltitude().valueInMeters());
   32:                 altitudeSeekBar.setOnChangedListner(this);
   33:
   34:                 angleSeekBar = (SeekBarWithText) view.findViewById(R.id.takeoffPit
ch);
   35:                 angleSeekBar.setValue(item.getMinPitch());
   36:                 angleSeekBar.setOnChangedListner(this);
   37:
   38:                 yawSeekBar = (SeekBarWithText) view.findViewById(R.id.waypointAngl
e);
   39:                 yawSeekBar.setValue(item.getYawAngle());
   40:                 yawSeekBar.setOnChangedListner(this);
   41:         }
   42:
   43:         @Override
   44:         public void onSeekBarChanged() {
   45:                 Takeoff item = (Takeoff) this.item;
   46:                 item.setAltitude(new Altitude(altitudeSeekBar.getValue()));
   47:                 item.setMinPitch(angleSeekBar.getValue());
   48:                 item.setYawAngle(yawSeekBar.getValue());
   49:         }
   50:
   51: }
```

```
    1: package com.droidplanner.fragments.mission;
    2:
    3: import android.view.View;
    4: import android.widget.CheckBox;
    5: import android.widget.CompoundButton;
    6: import android.widget.CompoundButton.OnCheckedChangeListener;
    7:
    8: import com.droidplanner.R;
    9: import com.droidplanner.drone.variables.mission.waypoints.Waypoint;
   10: import com.droidplanner.helpers.units.Altitude;
   11: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
   12: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   13:
   14: public class MissionWaypointFragment extends MissionDetailFragment implements
   15:                 OnTextSeekBarChangedListner, OnCheckedChangeListener {
   16:         private SeekBarWithText altitudeSeekBar;
   17:         private SeekBarWithText delaySeekBar;
   18:         private SeekBarWithText yawSeekBar;
   19:         private SeekBarWithText radiusSeekBar;
   20:         private SeekBarWithText orbitSeekBar;
   21:         private CheckBox orbitCCW;
   22:
   23:         @Override
   24:         protected int getResource() {
   25:                 return R.layout.fragment_detail_waypoint;
   26:         }
   27:
   28:         @Override
   29:         protected void setupViews(View view) {
   30:                 super.setupViews(view);
   31:                 typeSpinner.setSelection(commandAdapter.getPosition(MissionItemTyp
es.WAYPOINT));
   32:
   33:                 Waypoint item = (Waypoint) this.item;
   34:
   35:                 altitudeSeekBar = (SeekBarWithText) view
   36:                                 .findViewById(R.id.altitudeView);
   37:                 altitudeSeekBar.setValue(item.getAltitude().valueInMeters());
   38:                 altitudeSeekBar.setOnChangedListner(this);
   39:
   40:                 delaySeekBar = (SeekBarWithText) view.findViewById(R.id.waypointDe
lay);
   41:                 delaySeekBar.setValue(item.getDelay());
   42:                 delaySeekBar.setOnChangedListner(this);
   43:
   44:                 radiusSeekBar = (SeekBarWithText) view
   45:                                 .findViewById(R.id.waypointAcceptanceRadius);
   46:                 radiusSeekBar.setValue(item.getAcceptanceRadius());
   47:                 radiusSeekBar.setOnChangedListner(this);
   48:
   49:                 yawSeekBar = (SeekBarWithText) view.findViewById(R.id.waypointAngl
e);
   50:                 yawSeekBar.setValue(item.getYawAngle());
   51:                 yawSeekBar.setOnChangedListner(this);
   52:
   53:                 orbitSeekBar = (SeekBarWithText) view
   54:                                 .findViewById(R.id.waypointOrbitalRadius);
   55:                 orbitSeekBar.setOnChangedListner(this);
   56:                 orbitSeekBar.setAbsValue(item.getOrbitalRadius());
   57:
   58:                 orbitCCW = (CheckBox) view.findViewById(R.id.waypoint_CCW);
   59:                 orbitCCW.setChecked(item.isOrbitCCW());
   60:                 orbitCCW.setOnCheckedChangeListener(this);
   61:         }
   62:
   63:         @Override
   64:         public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
   65:                 onSeekBarChanged();
   66:         }
   67:
   68:         @Override
   69:         public void onSeekBarChanged() {
   70:                 Waypoint item = (Waypoint) this.item;
   71:                 item.setAltitude(new Altitude(altitudeSeekBar.getValue()));
   72:                 item.setDelay((float) delaySeekBar.getValue());
   73:                 item.setAcceptanceRadius((float) radiusSeekBar.getValue());
   74:                 item.setYawAngle((float) yawSeekBar.getValue());
   75:                 item.setOrbitalRadius((float) orbitSeekBar.getValue());
   76:                 item.setOrbitCCW(orbitCCW.isChecked());
   77:         }
   78:
   79: }
```

```
    1: package com.droidplanner.fragments.mission.survey;
    2:
    3: import android.app.Fragment;
    4: import android.content.Context;
    5: import android.os.Bundle;
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.View.OnClickListener;
    9: import android.view.ViewGroup;
   10: import android.widget.Spinner;
   11: import android.widget.Toast;
   12:
   13: import com.droidplanner.R;
   14: import com.droidplanner.drone.variables.mission.survey.SurveyData;
   15: import com.droidplanner.drone.variables.mission.survey.grid.Grid;
   16: import com.droidplanner.drone.variables.mission.survey.grid.GridBuilder;
   17: import com.droidplanner.file.IO.CameraInfo;
   18: import com.droidplanner.file.IO.CameraInfoReader;
   19: import com.droidplanner.file.help.CameraInfoLoader;
   20: import com.droidplanner.helpers.units.Altitude;
   21: import com.droidplanner.polygon.Polygon;
   22: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText.OnTextSeekBarChang
edListner;
   23: import com.droidplanner.widgets.spinners.SpinnerSelfSelect.OnSpinnerItemSelectedLi
stener;
   24: import com.google.android.gms.maps.model.LatLng;
   25:
   26: public class SurveyFragment extends Fragment implements
   27:                 OnTextSeekBarChangedListner, OnSpinnerItemSelectedListener, OnClic
kListener {
   28:         //public abstract void onPolygonGenerated(List<waypoint> list);
   29:
   30:         public interface OnNewGridListner{
   31:                 public void onNewGrid(Grid grid);
   32:                 public void onClearPolygon();
   33:         }
   34:
   35:         public enum pathModes {
   36:                 MISSION, POLYGON;
   37:         }
   38:
   39:         private Context context;
   40:         private Polygon polygon;
   41:
   42:         private SurveyData surveyData;
   43:         private CameraInfoLoader avaliableCameras;
   44:         private SurveyViews views;
   45:         private Grid grid;
   46:
   47:         private OnNewGridListner onSurveyListner;
   48:
   49:
   50:         @Override
   51:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   52:                         Bundle savedInstanceState) {
   53:                 context = getActivity().getApplicationContext();
   54:                 views = new SurveyViews(context);
   55:                 views.build(inflater, container,this);
   56:
   57:                 surveyData = new SurveyData();
   58:                 avaliableCameras = new CameraInfoLoader(context);
   59:                 views.updateCameraSpinner(avaliableCameras.getCameraInfoList());
   60:
   61:                 return views.getLayout();
   62:         }
   63:
   64:         public void setSurveyData(Polygon polygon, Altitude defaultAltitude){
   65:                 this.polygon = polygon;
   66:                 surveyData.setAltitude(defaultAltitude);
   67:                 update();
   68:         }
   69:
   70:         public void setOnSurveyListner(OnNewGridListner listner){
   71:                 this.onSurveyListner = listner;
   72:         }
   73:
   74:
   75:         @Override
   76:         public void onSeekBarChanged() {
   77:                 generateGrid();
   78:         }
   79:
   80:         public void generateGrid() {
   81:                 surveyData.update(views.angleView.getValue(), new Altitude(views.a
ltitudeView.getValue()),
   82:                                 views.overlapView.getValue(), views.sidelapView.ge
tValue());
   83:                 //TODO find a better origin point than (0,0)
   84:
   85:                 try {
   86:                         GridBuilder gridBuilder = new GridBuilder(polygon, surveyD
ata, new LatLng(0, 0));
   87:                         polygon.checkIfValid();
   88:                         grid = gridBuilder.generate();
   89:                         views.updateViews(surveyData,grid,polygon.getArea());
   90:                         grid.setAltitude(surveyData.getAltitude());
   91:                         onSurveyListner.onNewGrid(grid);
   92:                 } catch (Exception e) {
   93:                         Toast.makeText(context, e.getMessage(), Toast.LENGTH_SHORT
).show();
   94:                         views.blank();
   95:                 }
   96:         }
   97:
   98:         @Override
   99:         public void onSpinnerItemSelected(Spinner spinner, int position, String te
xt) {
  100:                 onCameraSelected(text);
  101:         }
  102:
  103:         private void onCameraSelected(String text) {
  104:                 CameraInfo cameraInfo;
  105:                 try {
  106:                         cameraInfo = avaliableCameras.openFile(text);
  107:                 } catch (Exception e) {
  108:                         Toast.makeText(context,
  109:                                         context.getString(R.string.error_when_open
ing_file),
  110:                                         Toast.LENGTH_SHORT).show();
  111:                         cameraInfo = CameraInfoReader.getNewMockCameraInfo();
  112:                 }
  113:                 surveyData.setCameraInfo(cameraInfo);
  114:                 update();
  115:         }
  116:
  117:         private void update() {
  118:                 views.updateSeekBarsValues(surveyData);
  119:                 generateGrid();
  120:         }
  121:
  122:         @Override
  123:         public void onClick(View view) {
  124:                 switch (view.getId()) {
  125:                 case R.id.checkBoxInnerWPs:
```

```
  126:                         surveyData.setInnerWpsState(views.innerWPsCheckbox.isCheck
ed());
  127:                         update();
  128:                         break;
  129:                 case R.id.clearPolyButton:
  130:                         onSurveyListner.onClearPolygon();
  131:                         break;
  132:                 case R.id.CheckBoxFootprints:
  133:                         update();
  134:                         break;
  135:                 }
  136:
  137:         }
  138:
  139:         public pathModes getPathToDraw() {
  140:                 if (views.modeButton.isChecked()) {
  141:                         return pathModes.POLYGON;
  142:                 }else{
  143:                         return pathModes.MISSION;
  144:                 }
  145:         }
  146:
  147:         public  SurveyData getSurveyData() {
  148:                 return surveyData;
  149:         }
  150:
  151:         public boolean isFootPrintOverlayEnabled() {
  152:                 return views.footprintCheckBox.isChecked();
  153:         }
  154:
  155: }
```

```
   1: package com.droidplanner.fragments.mission.survey;
   2:
   3: import android.content.Context;
   4: import android.view.LayoutInflater;
   5: import android.view.View;
   6: import android.view.ViewGroup;
   7: import android.widget.Button;
   8: import android.widget.CheckBox;
   9: import android.widget.SpinnerAdapter;
  10: import android.widget.TextView;
  11: import android.widget.ToggleButton;
  12:
  13: import com.droidplanner.R;
  14: import com.droidplanner.R.id;
  15: import com.droidplanner.R.string;
  16: import com.droidplanner.drone.variables.mission.survey.SurveyData;
  17: import com.droidplanner.drone.variables.mission.survey.grid.Grid;
  18: import com.droidplanner.helpers.units.Area;
  19: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
  20: import com.droidplanner.widgets.spinners.SpinnerSelfSelect;
  21:
  22: public class SurveyViews {
  23:         public SeekBarWithText overlapView;
  24:         public SeekBarWithText angleView;
  25:         public SeekBarWithText altitudeView;
  26:         public SeekBarWithText sidelapView;
  27:         public TextView distanceBetweenLinesTextView;
  28:         public TextView areaTextView;
  29:         public TextView distanceTextView;
  30:         public TextView footprintTextView;
  31:         public TextView groundResolutionTextView;
  32:         public SpinnerSelfSelect cameraSpinner;
  33:         public CheckBox innerWPsCheckbox;
  34:         public TextView numberOfPicturesView;
  35:         public TextView numberOfStripsView;
  36:         public TextView lengthView;
  37:         private Context context;
  38:         private View layout;
  39:         protected ToggleButton modeButton;
  40:         private Button clearPolyButton;
  41:         public CheckBox footprintCheckBox;
  42:
  43:         public SurveyViews(Context context) {
  44:                 this.context = context;
  45:         }
  46:
  47:         void updateViews(SurveyData surveyData, Grid grid, Area area) {
  48:                 footprintTextView.setText(context.getString(string.footprint) + ":
 "
  49:                                 + surveyData.getLateralFootPrint() + " x"
  50:                                 + surveyData.getLongitudinalFootPrint());
  51:                 groundResolutionTextView.setText(context
  52:                                 .getString(string.ground_resolution)
  53:                                 + surveyData.getGroundResolution() + "/px");
  54:                 distanceTextView.setText(context
  55:                                 .getString(string.distance_between_pictures)
  56:                                 + ": "
  57:                                 + surveyData.getLongitudinalPictureDistance());
  58:                 distanceBetweenLinesTextView.setText(context
  59:                                 .getString(string.distance_between_lines)
  60:                                 + ": "
  61:                                 + surveyData.getLateralPictureDistance());
  62:                 areaTextView.setText(context.getString(string.area) + ": " + area)
;
  63:                 lengthView.setText(context.getString(string.mission_length) + ": "
  64:                                 + grid.getLength());
  65:                 numberOfPicturesView.setText(context.getString(string.pictures) +
": "
  66:                                 + grid.getCameraCount());
  67:                 numberOfStripsView.setText(context.getString(string.number_of_stri
ps)
  68:                                 + ": " + grid.getNumberOfLines());
  69:         }
  70:
  71:         public void blank() {
  72:                 String unknowData = "???";
  73:                 footprintTextView.setText(unknowData);
  74:                 groundResolutionTextView.setText(unknowData);
  75:                 distanceTextView.setText(unknowData);
  76:                 distanceBetweenLinesTextView.setText(unknowData);
  77:                 areaTextView.setText(unknowData);
  78:                 lengthView.setText(unknowData);
  79:                 numberOfPicturesView.setText(unknowData);
  80:                 numberOfStripsView.setText(unknowData);
  81:
  82:         }
  83:
  84:         void updateSeekBarsValues(SurveyData surveyData) {
  85:                 angleView.setValue(surveyData.getAngle());
  86:                 altitudeView.setValue(surveyData.getAltitude().valueInMeters());
  87:                 sidelapView.setValue(surveyData.getSidelap());
  88:                 overlapView.setValue(surveyData.getOverlap());
  89:         }
  90:
  91:         public void build(LayoutInflater inflater, ViewGroup container,
  92:                         SurveyFragment surveyDialog) {
  93:                 layout = inflater.inflate(R.layout.fragment_detail_survey, null);
  94:                 cameraSpinner = (SpinnerSelfSelect) layout
  95:                                 .findViewById(id.cameraFileSpinner);
  96:                 modeButton = (ToggleButton) layout.findViewById(id.surveyModeButto
n);
  97:                 clearPolyButton = (Button) layout.findViewById(id.clearPolyButton)
;
  98:                 footprintCheckBox = (CheckBox) layout
  99:                                 .findViewById(id.CheckBoxFootprints);
 100:
 101:                 angleView = (SeekBarWithText) layout.findViewById(id.angleView);
 102:                 overlapView = (SeekBarWithText) layout.findViewById(id.overlapView
);
 103:                 sidelapView = (SeekBarWithText) layout.findViewById(id.sidelapView
);
 104:                 altitudeView = (SeekBarWithText) layout.findViewById(id.altitudeVi
ew);
 105:
 106:                 innerWPsCheckbox = (CheckBox) layout.findViewById(id.checkBoxInner
WPs);
 107:
 108:                 areaTextView = (TextView) layout.findViewById(id.areaTextView);
 109:                 distanceBetweenLinesTextView = (TextView) layout
 110:                                 .findViewById(id.distanceBetweenLinesTextView);
 111:                 footprintTextView = (TextView) layout
 112:                                 .findViewById(id.footprintTextView);
 113:                 groundResolutionTextView = (TextView) layout
 114:                                 .findViewById(id.groundResolutionTextView);
 115:                 distanceTextView = (TextView) layout.findViewById(id.distanceTextV
iew);
 116:                 numberOfPicturesView = (TextView) layout
 117:                                 .findViewById(id.numberOfPicturesTextView);
 118:                 numberOfStripsView = (TextView) layout
 119:                                 .findViewById(id.numberOfStripsTextView);
 120:                 lengthView = (TextView) layout.findViewById(id.lengthTextView);
 121:
 122:                 footprintCheckBox.setOnClickListener(surveyDialog);
 123:                 angleView.setOnChangedListner(surveyDialog);
```

```
124:                 altitudeView.setOnChangedListner(surveyDialog);
125:                 overlapView.setOnChangedListner(surveyDialog);
126:                 sidelapView.setOnChangedListner(surveyDialog);
127:                 innerWPsCheckbox.setOnClickListener(surveyDialog);
128:                 cameraSpinner.setOnSpinnerItemSelectedListener(surveyDialog);
129:                 clearPolyButton.setOnClickListener(surveyDialog);
130:         }
131:
132:         void updateCameraSpinner(SpinnerAdapter spinnerAdapter) {
133:                 cameraSpinner.setAdapter(spinnerAdapter);
134:                 cameraSpinner.setSelection(0);
135:         }
136:
137:         public View getLayout() {
138:                 return layout;
139:         }
140:
141: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import android.app.Activity;
    4: import android.app.Fragment;
    5: import android.os.Bundle;
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.View.OnClickListener;
    9: import android.view.ViewGroup;
   10: import android.widget.Button;
   11:
   12: import com.MAVLink.Messages.ApmModes;
   13: import com.droidplanner.DroidPlannerApp;
   14: import com.droidplanner.R;
   15: import com.droidplanner.drone.Drone;
   16: import com.droidplanner.drone.DroneInterfaces.OnStateListner;
   17:
   18: public class MissionControlFragment extends Fragment implements
   19:                 OnClickListener, OnStateListner {
   20:
   21:         public interface OnMissionControlInteraction {
   22:                 public void onJoystickSelected();
   23:
   24:                 public void onPlanningSelected();
   25:         }
   26:
   27:         private Drone drone;
   28:         private OnMissionControlInteraction listner;
   29:         private Button homeBtn;
   30:         private Button missionBtn;
   31:         private Button joystickBtn;
   32:         private Button landBtn;
   33:         private Button loiterBtn;
   34:         private Button takeoffBtn;
   35:         private Button followBtn;
   36:
   37:         @Override
   38:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   39:                         Bundle savedInstanceState) {
   40:                 View view = inflater.inflate(R.layout.fragment_mission_control,
   41:                                 container, false);
   42:                 setupViews(view);
   43:                 setupListner();
   44:                 drone = ((DroidPlannerApp) getActivity().getApplication()).drone;
   45:                 drone.state.addFlightStateListner(this);
   46:                 return view;
   47:         }
   48:
   49:         @Override
   50:         public void onAttach(Activity activity) {
   51:                 super.onAttach(activity);
   52:                 listner = (OnMissionControlInteraction) activity;
   53:         }
   54:
   55:         @Override
   56:         public void onDestroy() {
   57:                 super.onDestroy();
   58:                 drone.state.removeFlightStateListner(this);
   59:         }
   60:
   61:         private void setupViews(View parentView) {
   62:                 missionBtn = (Button) parentView.findViewById(R.id.mc_planningBtn)
;
   63:                 joystickBtn = (Button) parentView.findViewById(R.id.mc_joystickBtn
);
   64:                 homeBtn = (Button) parentView.findViewById(R.id.mc_homeBtn);
   65:                 landBtn = (Button) parentView.findViewById(R.id.mc_land);
   66:                 takeoffBtn = (Button) parentView.findViewById(R.id.mc_takeoff);
   67:                 loiterBtn = (Button) parentView.findViewById(R.id.mc_loiter);
   68:                 followBtn = (Button) parentView.findViewById(R.id.mc_follow);
   69:                 setToLandedState();
   70:         }
   71:
   72:         private void setupListner() {
   73:                 missionBtn.setOnClickListener(this);
   74:                 joystickBtn.setOnClickListener(this);
   75:                 homeBtn.setOnClickListener(this);
   76:                 landBtn.setOnClickListener(this);
   77:                 takeoffBtn.setOnClickListener(this);
   78:                 loiterBtn.setOnClickListener(this);
   79:                 followBtn.setOnClickListener(this);
   80:         }
   81:
   82:         @Override
   83:         public void onClick(View v) {
   84:                 switch (v.getId()) {
   85:                 case R.id.mc_planningBtn:
   86:                         listner.onPlanningSelected();
   87:                         break;
   88:                 case R.id.mc_joystickBtn:
   89:                         listner.onJoystickSelected();
   90:                         break;
   91:                 case R.id.mc_land:
   92:                         drone.state.changeFlightMode(ApmModes.ROTOR_LAND);
   93:                         break;
   94:                 case R.id.mc_takeoff:
   95:                         drone.state.changeFlightMode(ApmModes.ROTOR_TAKEOFF);
   96:                         break;
   97:                 case R.id.mc_homeBtn:
   98:                         drone.state.changeFlightMode(ApmModes.ROTOR_RTL);
   99:                         break;
  100:                 case R.id.mc_loiter:
  101:                         drone.state.changeFlightMode(ApmModes.ROTOR_LOITER);
  102:                         break;
  103:                 }
  104:         }
  105:
  106:         @Override
  107:         public void onFlightStateChanged() {
  108:                 if (drone.state.isFlying()) {
  109:                         setToFlyingState();
  110:                 }else{
  111:                         setToLandedState();
  112:                 }
  113:         }
  114:
  115:         private void setToLandedState() {
  116:                 takeoffBtn.setVisibility(View.VISIBLE);
  117:                 landBtn.setVisibility(View.GONE);
  118:                 homeBtn.setVisibility(View.GONE);
  119:                 loiterBtn.setVisibility(View.GONE);
  120:                 followBtn.setVisibility(View.GONE);
  121:         }
  122:
  123:         private void setToFlyingState() {
  124:                 landBtn.setVisibility(View.VISIBLE);
  125:                 homeBtn.setVisibility(View.VISIBLE);
  126:                 loiterBtn.setVisibility(View.VISIBLE);
  127:                 takeoffBtn.setVisibility(View.GONE);
  128:         }
  129:
  130:         @Override
  131:         public void onArmChanged() {
  132:                 // TODO Auto-generated method stub
```

```
133:
134:        }
135:
136:        @Override
137:        public void onFailsafeChanged() {
138:                // TODO Auto-generated method stub
139:
140:        }
141:
142: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import android.app.Activity;
    4: import android.app.Fragment;
    5: import android.os.Bundle;
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.ViewGroup;
    9: import android.widget.AdapterView;
   10: import android.widget.AdapterView.OnItemClickListener;
   11:
   12: import com.droidplanner.DroidPlannerApp;
   13: import com.droidplanner.DroidPlannerApp.OnWaypointChangedListner;
   14: import com.droidplanner.R;
   15: import com.droidplanner.drone.variables.mission.Mission;
   16: import com.droidplanner.drone.variables.mission.MissionItem;
   17: import com.droidplanner.fragments.helpers.OnMapInteractionListener;
   18: import com.droidplanner.widgets.adapterViews.MissionItemView;
   19: import com.mobeta.android.dslv.HorizontalListView;
   20:
   21: public class MissionFragment extends Fragment implements  OnWaypointChangedListner
, OnItemClickListener{
   22:         public HorizontalListView list;
   23:         private Mission mission;
   24:         private MissionItemView adapter;
   25:         private OnMapInteractionListener mListner;
   26:
   27:         @Override
   28:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   29:                         Bundle savedInstanceState) {
   30:                 View view = inflater.inflate(R.layout.fragment_mission, container,
   31:                                 false);
   32:                 list = (HorizontalListView) view.findViewById(R.id.listView1);
   33:                 //list.setDropListener(this);
   34:                 //list.setRemoveListener(this);
   35:                 //list.setDragScrollProfile(this);
   36:
   37:                 adapter = new MissionItemView(this.getActivity(), android.R.layout
.simple_list_item_1);
   38:                 list.setAdapter(adapter);
   39:
   40:
   41:                 mission = ((DroidPlannerApp) getActivity().getApplication()).drone
.mission;
   42:                 mission.addOnMissionUpdateListner(this);
   43:                 adapter = new MissionItemView(this.getActivity(), android.R.layout
.simple_list_item_1,mission.getItems());
   44:                 list.setAdapter(adapter);
   45:                 list.setOnItemClickListener(this);
   46:
   47:                 return view;
   48:         }
   49:
   50:         @Override
   51:         public void onAttach(Activity activity) {
   52:                 super.onAttach(activity);
   53:                 mListner = (OnMapInteractionListener) activity;
   54:         }
   55:
   56:         @Override
   57:         public void onDestroy() {
   58:                 super.onDestroy();
   59:                 mission.removeOnMissionUpdateListner(this);
   60:         }
   61:
   62:         public void update() {
   63:                 adapter.notifyDataSetChanged();
   64:         }
   65:
   66:         /*
   67:         @Override
   68:         public void onListItemClick(ListView l, View v, int position, long id) {
   69:                 Log.d("T", "touched "+position);
   70:                 DialogMissionFactory.getDialog(adapter.getItem(position), this.get
Activity(), mission);
   71:                 super.onListItemClick(l, v, position, id);
   72:         }*/
   73:
   74:         @Override
   75:         public void onMissionUpdate() {
   76:                 update();
   77:         }
   78:
   79:         @Override
   80:         public void onItemClick(AdapterView<?> parent, View view, int position, lo
ng id) {
   81:                 mListner.onMarkerClick(((MissionItem) parent.getItemAtPosition(pos
ition)));
   82:         }
   83:
   84: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import java.util.ArrayList;
    4: import java.util.Collections;
    5: import java.util.Comparator;
    6: import java.util.List;
    7:
    8: import android.app.Activity;
    9: import android.app.Fragment;
   10: import android.app.ProgressDialog;
   11: import android.content.Context;
   12: import android.graphics.Color;
   13: import android.os.Bundle;
   14: import android.view.LayoutInflater;
   15: import android.view.View;
   16: import android.view.View.OnClickListener;
   17: import android.view.ViewGroup;
   18: import android.widget.TableLayout;
   19: import android.widget.TextView;
   20: import android.widget.Toast;
   21:
   22: import com.MAVLink.Messages.enums.MAV_TYPE;
   23: import com.droidplanner.R;
   24: import com.droidplanner.activitys.helpers.SuperActivity;
   25: import com.droidplanner.dialogs.openfile.OpenFileDialog;
   26: import com.droidplanner.dialogs.openfile.OpenParameterDialog;
   27: import com.droidplanner.drone.Drone;
   28: import com.droidplanner.drone.DroneInterfaces.OnParameterManagerListner;
   29: import com.droidplanner.drone.variables.Parameters;
   30: import com.droidplanner.file.IO.ParameterWriter;
   31: import com.droidplanner.parameters.Parameter;
   32: import com.droidplanner.widgets.adapterViews.ParamRow;
   33:
   34: public class ParametersTableFragment extends Fragment implements
   35:                 OnClickListener, OnParameterManagerListner {
   36:
   37:         private TableLayout parameterTable;
   38:         private List<ParamRow> rowList = new ArrayList<ParamRow>();
   39:         private Drone drone;
   40:         private Context context;
   41:
   42:         private ProgressDialog pd;
   43:         private TextView refreshTextView;
   44:
   45:         @Override
   46:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   47:                         Bundle savedInstanceState) {
   48:                 View view = inflater.inflate(R.layout.fragment_parameters, contain
er,
   49:                                 false);
   50:                 parameterTable = (TableLayout) view.findViewById(R.id.parametersTa
ble);
   51:
   52:                 refreshTextView = (TextView) view
   53:                                 .findViewById(R.id.refreshTextView);
   54:                 refreshTextView.setOnClickListener(this);
   55:                 return view;
   56:         }
   57:
   58:         @Override
   59:         public void onAttach(Activity activity) {
   60:                 super.onAttach(activity);
   61:                 context = (Context) activity;
   62:                 drone = ((SuperActivity) activity).drone;
   63:                 drone.parameters.parameterListner = this;
   64:         }
   65:
   66:         public void refreshRowParameter(Parameter parameter, Parameters parameters
) {
   67:                 try {
   68:                         Parameter.checkParameterName(parameter.name);
   69:                         ParamRow row = findRowByName(parameter.name);
   70:                         if (row != null) {
   71:                                 row.setParam(parameter, parameters);
   72:                         } else {
   73:                                 addParameterRow(parameter, parameters);
   74:                         }
   75:                 } catch (Exception e) {
   76:                 }
   77:         }
   78:
   79:         private ParamRow findRowByName(String name) {
   80:                 for (ParamRow row : rowList) {
   81:                         if (row.getParamName().equals(name)) {
   82:                                 return row;
   83:                         }
   84:                 }
   85:                 return null;
   86:         }
   87:
   88:         private void addParameterRow(Parameter param, Parameters parameters) {
   89:                 ParamRow pRow = new ParamRow(this.getActivity());
   90:                 pRow.setParam(param, parameters);
   91:
   92:                 // alternate background colors for clarity
   93:                 if ((rowList.size() % 2) == 1)
   94:                         pRow.setBackgroundColor(Color.BLACK);
   95:
   96:                 rowList.add(pRow);
   97:                 parameterTable.addView(pRow);
   98:         }
   99:
  100:         private List<Parameter> getParameterListFromTable() {
  101:                 ArrayList<Parameter> parameters = new ArrayList<Parameter>();
  102:                 for (ParamRow row : rowList) {
  103:                         parameters.add(row.getParameterFromRow());
  104:                 }
  105:                 return parameters;
  106:         }
  107:
  108:         public List<ParamRow> getModifiedParametersRows() {
  109:                 ArrayList<ParamRow> modParameters = new ArrayList<ParamRow>();
  110:                 for (ParamRow row : rowList) {
  111:                         if (!row.isNewValueEqualToDroneParam()) {
  112:                                 modParameters.add(row);
  113:                         }
  114:                 }
  115:                 return modParameters;
  116:         }
  117:
  118:         public void saveParametersToFile() {
  119:                 List<Parameter> parameterList = getParameterListFromTable();
  120:                 if (parameterList.size() > 0) {
  121:                         ParameterWriter parameterWriter = new ParameterWriter(para
meterList);
  122:                         if (parameterWriter.saveParametersToFile()) {
  123:                                 Toast.makeText(this.getActivity(), "Parameters sav
ed",
  124:                                                 Toast.LENGTH_SHORT).show();
  125:                         }
  126:                 }
  127:         }
  128:
  129:         public void refresh(Parameters parameters) {
```

```
130:                 for (ParamRow row : rowList)
131:                         row.setParamMetadata(parameters);
132:         }
133:
134:         @Override
135:         public void onClick(View view) {
136:                 if (view.getId() == R.id.refreshTextView) {
137:                         refreshParameters();
138:                 }
139:
140:         }
141:
142:         private void refreshParameters() {
143:                 if (drone.MavClient.isConnected()) {
144:                         drone.parameters.getAllParameters();
145:                 } else {
146:                         Toast.makeText(context, "Please connect first", Toast.LENG
TH_SHORT)
147:                                         .show();
148:                 }
149:         }
150:
151:         private void writeModifiedParametersToDrone() {
152:                 List<ParamRow> modRows = getModifiedParametersRows();
153:                 for (ParamRow row : modRows) {
154:                         if (!row.isNewValueEqualToDroneParam()) {
155:                                 drone.parameters.sendParameter(row.getParameterFro
mRow());
156:                         }
157:                 }
158:                 Toast.makeText(context, "Write " + modRows.size() + " parameters",
159:                                 Toast.LENGTH_SHORT).show();
160:         }
161:
162:         private void openParametersFromFile() {
163:                 OpenFileDialog dialog = new OpenParameterDialog() {
164:                         @Override
165:                         public void parameterFileLoaded(List<Parameter> parameters
) {
166:                                 Collections.sort(parameters, new Comparator<Parame
ter>() {
167:                                         @Override
168:                                         public int compare(Parameter p1, Parameter
 p2) {
169:                                                 return p1.name.compareTo(p2.name);
170:                                         }
171:                                 });
172:                                 drone.parameters.loadMetadata(context, null);
173:                                 for (Parameter parameter : parameters)
174:                                         refreshRowParameter(parameter, drone.param
eters);
175:                         }
176:                 };
177:                 dialog.openDialog(context);
178:         }
179:
180:         @Override
181:         public void onBeginReceivingParameters() {
182:                 pd = new ProgressDialog(context);
183:                 pd.setTitle("Refreshing Parameters...");
184:                 pd.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
185:                 pd.setIndeterminate(true);
186:                 pd.setCancelable(false);
187:                 pd.setCanceledOnTouchOutside(true);
188:
189:                 pd.show();
190:         }
191:
192:         @Override
193:         public void onParameterReceived(Parameter parameter, int index, int count)
{
194:                 if (pd != null) {
195:                         if (pd.isIndeterminate()) {
196:                                 pd.setIndeterminate(false);
197:                                 pd.setMax(count);
198:                         }
199:                         pd.setProgress(index);
200:                 }
201:         }
202:
203:         @Override
204:         public void onEndReceivingParameters(List<Parameter> parameters) {
205:                 Collections.sort(parameters, new Comparator<Parameter>() {
206:                         @Override
207:                         public int compare(Parameter p1, Parameter p2) {
208:                                 return p1.name.compareTo(p2.name);
209:                         }
210:                 });
211:                 drone.parameters.loadMetadata(context, getMetadataType());
212:                 for (Parameter parameter : parameters)
213:                         refreshRowParameter(parameter, drone.parameters);
214:
215:                 // dismiss progress dialog
216:                 if (pd != null) {
217:                         pd.dismiss();
218:                         pd = null;
219:                 }
220:
221:                 //Remove the Refresh text view
222:                 refreshTextView.setVisibility(View.GONE);
223:
224:
225:         }
226:
227:         @Override
228:         public void onParamterMetaDataChanged() {
229:                 drone.parameters.loadMetadata(context, null);
230:                 refresh(drone.parameters);
231:         }
232:
233:         private String getMetadataType() {
234:                 if (drone.MavClient.isConnected()) {
235:                         // online: derive from connected vehicle type
236:                         switch (drone.type.getType()) {
237:                         case MAV_TYPE.MAV_TYPE_FIXED_WING: /* Fixed wing aircraft.
 | */
238:                                 return "ArduPlane";
239:
240:                         case MAV_TYPE.MAV_TYPE_GENERIC: /* Generic micro air vehic
le. | */
241:                         case MAV_TYPE.MAV_TYPE_QUADROTOR: /* Quadrotor | */
242:                         case MAV_TYPE.MAV_TYPE_COAXIAL: /* Coaxial helicopter | */
243:                         case MAV_TYPE.MAV_TYPE_HELICOPTER: /*
244:
        * Normal helicopter with tail
245:
        * rotor. |
246:
        */
247:                         case MAV_TYPE.MAV_TYPE_HEXAROTOR: /* Hexarotor | */
248:                         case MAV_TYPE.MAV_TYPE_OCTOROTOR: /* Octorotor | */
249:                         case MAV_TYPE.MAV_TYPE_TRICOPTER: /* Octorotor | */
250:                                 return "ArduCopter2";
251:
```

```
  252:                         case MAV_TYPE.MAV_TYPE_GROUND_ROVER: /* Ground rover | */
  253:                         case MAV_TYPE.MAV_TYPE_SURFACE_BOAT: /* Surface vessel, bo
at, ship | */
  254:                                 return "ArduRover";
  255:
  256:                                 // case MAV_TYPE.MAV_TYPE_ANTENNA_TRACKER: /* Grou
nd
  257:                                 // installation | */
  258:                                 // case MAV_TYPE.MAV_TYPE_GCS: /* Operator control
 unit / ground
  259:                                 // control station | */
  260:                                 // case MAV_TYPE.MAV_TYPE_AIRSHIP: /* Airship, con
trolled | */
  261:                                 // case MAV_TYPE.MAV_TYPE_FREE_BALLOON: /* Free ba
lloon,
  262:                                 // uncontrolled | */
  263:                                 // case MAV_TYPE.MAV_TYPE_ROCKET: /* Rocket | */
  264:                                 // case MAV_TYPE.MAV_TYPE_SUBMARINE: /* Submarine
| */
  265:                                 // case MAV_TYPE.MAV_TYPE_FLAPPING_WING: /* Flappi
ng wing | */
  266:                                 // case MAV_TYPE.MAV_TYPE_KITE: /* Flapping wing |
 */
  267:                         default:
  268:                                 // unsupported
  269:                                 return null;
  270:                         }
  271:                 } else {
  272:                         // offline: use configured parameter metadata type
  273:                         return null;
  274:                 }
  275:         }
  276: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.gesture.GestureOverlayView;
    7: import android.gesture.GestureOverlayView.OnGestureListener;
    8: import android.graphics.Point;
    9: import android.view.MotionEvent;
   10:
   11: import com.droidplanner.helpers.geoTools.Simplify;
   12:
   13: public class PathGesture implements OnGestureListener {
   14:
   15:         private static final int TOLERANCE = 15;
   16:         private static final int STROKE_WIDTH = 3;
   17:
   18:         public interface OnPathFinishedListner {
   19:
   20:                 void onPathFinished(List<Point> path);
   21:         }
   22:
   23:         public double toleranceInPixels;
   24:         public GestureOverlayView view;
   25:         public OnPathFinishedListner listner;
   26:
   27:         public PathGesture(GestureOverlayView view) {
   28:                 this.view = view;
   29:                 this.view.addOnGestureListener(this);
   30:                 this.view.setEnabled(false);
   31:                 this.view.setGestureStrokeWidth(scaleDpToPixels(STROKE_WIDTH));
   32:                 toleranceInPixels = scaleDpToPixels(TOLERANCE);
   33:         }
   34:
   35:         private int scaleDpToPixels(double value) {
   36:                 final float scale = view.getResources().getDisplayMetrics().densit
y;
   37:                 return (int) Math.round(value * scale);
   38:         }
   39:
   40:         public void enableGestureDetection() {
   41:                 view.setEnabled(true);
   42:         }
   43:
   44:         public void setOnPathFinishedListner(OnPathFinishedListner listner) {
   45:                 this.listner = listner;
   46:         }
   47:
   48:         @Override
   49:         public void onGestureEnded(GestureOverlayView arg0, MotionEvent arg1) {
   50:                 view.setEnabled(false);
   51:                 List<Point> path = decodeGesture();
   52:                 if (path.size() > 1) {
   53:                         path = Simplify.simplify(path, toleranceInPixels);
   54:                 }
   55:                 listner.onPathFinished(path);
   56:         }
   57:
   58:         private List<Point> decodeGesture() {
   59:                 List<Point> path = new ArrayList<Point>();
   60:                 extractPathFromGesture(path);
   61:                 return path;
   62:         }
   63:
   64:         private void extractPathFromGesture(List<Point> path) {
   65:                 float[] points = view.getGesture().getStrokes().get(0).points;
   66:                 for (int i = 0; i < points.length; i += 2) {
   67:                         path.add(new Point((int) points[i], (int) points[i + 1]));
   68:                 }
   69:         }
   70:
   71:         @Override
   72:         public void onGesture(GestureOverlayView arg0, MotionEvent arg1) {
   73:         }
   74:
   75:         @Override
   76:         public void onGestureCancelled(GestureOverlayView arg0, MotionEvent arg1)
{
   77:         }
   78:
   79:         @Override
   80:         public void onGestureStarted(GestureOverlayView arg0, MotionEvent arg1) {
   81:         }
   82: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3:
    4: import com.droidplanner.R;
    5:
    6: import android.app.Fragment;
    7: import android.os.Bundle;
    8: import android.view.LayoutInflater;
    9: import android.view.View;
   10: import android.view.ViewGroup;
   11:
   12: public class PlanningFragment extends Fragment {
   13:
   14:         @Override
   15:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   16:                         Bundle savedInstanceState) {
   17:                 return inflater.inflate(R.layout.fragment_planning, container, fal
se);
   18:         }
   19:
   20:
   21: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3:
    4: import java.util.List;
    5:
    6: import android.annotation.SuppressLint;
    7: import android.app.Activity;
    8: import android.graphics.Color;
    9: import android.graphics.Point;
   10: import android.os.Bundle;
   11: import android.view.LayoutInflater;
   12: import android.view.View;
   13: import android.view.ViewGroup;
   14:
   15: import com.droidplanner.DroidPlannerApp.OnWaypointChangedListner;
   16: import com.droidplanner.drone.variables.mission.waypoints.SpatialCoordItem;
   17: import com.droidplanner.fragments.PathGesture.OnPathFinishedListner;
   18: import com.droidplanner.fragments.helpers.CameraGroundOverlays;
   19: import com.droidplanner.fragments.helpers.DroneMap;
   20: import com.droidplanner.fragments.helpers.MapPath;
   21: import com.droidplanner.fragments.helpers.OnMapInteractionListener;
   22: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
   23: import com.droidplanner.polygon.Polygon;
   24: import com.droidplanner.polygon.PolygonPoint;
   25: import com.google.android.gms.maps.GoogleMap.OnMapClickListener;
   26: import com.google.android.gms.maps.GoogleMap.OnMapLongClickListener;
   27: import com.google.android.gms.maps.GoogleMap.OnMarkerClickListener;
   28: import com.google.android.gms.maps.GoogleMap.OnMarkerDragListener;
   29: import com.google.android.gms.maps.model.LatLng;
   30: import com.google.android.gms.maps.model.Marker;
   31:
   32: @SuppressLint("UseSparseArrays")
   33: public class PlanningMapFragment extends DroneMap implements
   34:                 OnMapLongClickListener, OnMarkerDragListener, OnMapClickListener,
   35:                 OnMarkerClickListener, OnPathFinishedListner, OnWaypointChangedLis
tner {
   36:
   37:
   38:         public OnMapInteractionListener mListener;
   39:         public MapPath polygonPath;
   40:
   41:         public CameraGroundOverlays cameraOverlays;
   42:         public Polygon polygon = new Polygon();
   43:
   44:         @Override
   45:         public View onCreateView(LayoutInflater inflater, ViewGroup viewGroup,
   46:                         Bundle bundle) {
   47:                 View view = super.onCreateView(inflater, viewGroup, bundle);
   48:
   49:                 mMap.setOnMarkerDragListener(this);
   50:                 mMap.setOnMarkerClickListener(this);
   51:                 mMap.setOnMapClickListener(this);
   52:                 mMap.setOnMapLongClickListener(this);
   53:                 polygonPath = new MapPath(mMap, Color.BLACK, 2);
   54:                 cameraOverlays = new CameraGroundOverlays(mMap);
   55:
   56:                 return view;
   57:         }
   58:
   59:         @Override
   60:         public void update() {
   61:                 super.update();
   62:                 markers.updateMarkers(polygon .getPolygonPoints(), true, context);

   63:                 polygonPath.update(polygon);
   64:         }
   65:
   66:         @Override
   67:         public void onMapLongClick(LatLng point) {
   68:                 mListener.onAddPoint(point);
   69:         }
   70:
   71:         @Override
   72:         public void onMarkerDrag(Marker marker) {
   73:                 MarkerSource source = markers.getSourceFromMarker(marker);
   74:                 checkForWaypointMarkerMoving(source, marker, true);
   75:         }
   76:
   77:         @Override
   78:         public void onMarkerDragStart(Marker marker) {
   79:                 MarkerSource source = markers.getSourceFromMarker(marker);
   80:                 checkForWaypointMarkerMoving(source, marker, false);
   81:         }
   82:
   83:         private void checkForWaypointMarkerMoving(MarkerSource source, Marker mark
er, boolean dragging) {
   84:                 if (SpatialCoordItem.class.isInstance(source)) {
   85:                         LatLng position = marker.getPosition();
   86:
   87:                         // update marker source
   88:                         SpatialCoordItem waypoint = (SpatialCoordItem) source;
   89:                         waypoint.setCoordinate(position);
   90:
   91:                         /*
   92:                         // update info window
   93:                         if(dragging)
   94:                                 waypoint.updateDistanceFromPrevPoint();
   95:                         else
   96:                                 waypoint.setPrevPoint(mission.getWaypoints());
   97:                         updateInfoWindow(waypoint, marker);
   98:                          */
   99:
  100:                         // update flight path
  101:                         missionPath.update(mission);
  102:                         mListener.onMovingWaypoint(waypoint, position);
  103:                 }
  104:         }
  105:
  106:         /*
  107:         private void updateInfoWindow(GenericWaypoint waypoint, Marker marker) {
  108:                 marker.setTitle(waypoint.getNumber() + " " + waypoint.getCmd().get
Name());
  109:
  110:                 // display distance from last waypoint if available
  111:                 double distanceFromPrevPathPoint = waypoint.getDistanceFromPrevPoi
nt();
  112:                 if(distanceFromPrevPathPoint != com.droidplanner.drone.variables.m
ission.waypoints.GenericWaypoint.UNKNOWN_DISTANCE)
  113:                         marker.setSnippet(String.format("%.0fm", distanceFromPrevP
athPoint));
  114:
  115:                 marker.showInfoWindow();
  116:         }
  117:         */
  118:
  119:         @Override
  120:         public void onMarkerDragEnd(Marker marker) {
  121:                 MarkerSource source = markers.getSourceFromMarker(marker);
  122:                 checkForWaypointMarker(source, marker);
  123:                 checkForPolygonMarker(source, marker);
  124:         }
  125:
  126:         private void checkForWaypointMarker(MarkerSource source, Marker marker) {
  127:                 if (SpatialCoordItem.class.isInstance(source)) {
```

```
  128:                         mListener.onMoveWaypoint((SpatialCoordItem) source, marker
.getPosition());
  129:                 }
  130:         }
  131:
  132:         private void checkForPolygonMarker(MarkerSource source, Marker marker) {
  133:                 if (PolygonPoint.class.isInstance(source)) {
  134:                         mListener.onMovePolygonPoint((PolygonPoint) source,
  135:                                         marker.getPosition());
  136:                 }
  137:         }
  138:
  139:         @Override
  140:         public void onMapClick(LatLng point) {
  141:                 mListener.onMapClick(point);
  142:         }
  143:
  144:         @Override
  145:         public void onAttach(Activity activity) {
  146:                 super.onAttach(activity);
  147:                 mListener = (OnMapInteractionListener) activity;
  148:         }
  149:
  150:         @Override
  151:         public boolean onMarkerClick(Marker marker) {
  152:                 MarkerSource source = markers.getSourceFromMarker(marker);
  153:                 if (source instanceof SpatialCoordItem) {
  154:                         return mListener.onMarkerClick((SpatialCoordItem) source);
  155:                 } else {
  156:                         return false;
  157:                 }
  158:         }
  159:
  160:         @Override
  161:         public void onPathFinished(List<Point> path) {
  162:                 // TODO Auto-generated method stub
  163:
  164:         }
  165:
  166: }
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import android.app.Fragment;
    4: import android.content.SharedPreferences;
    5: import android.os.Bundle;
    6: import android.preference.PreferenceManager;
    7: import android.view.InputDevice;
    8: import android.view.LayoutInflater;
    9: import android.view.MotionEvent;
   10: import android.view.View;
   11: import android.view.ViewGroup;
   12: import android.widget.TextView;
   13:
   14: import com.droidplanner.R;
   15: import com.droidplanner.widgets.joystick.JoystickMovedListener;
   16: import com.droidplanner.widgets.joystick.JoystickView;
   17:
   18: public class RCFragment extends Fragment {
   19:
   20:         private JoystickView joystickL, joystickR;
   21:         private TextView textViewThrottle, textViewRudder, textViewAileron,
   22:                         textViewElevator;
   23:         private boolean rcIsMode1 = false;
   24:
   25:         @Override
   26:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
   27:                         Bundle savedInstanceState) {
   28:                 View view = inflater.inflate(R.layout.fragment_rc, container, fals
e);
   29:
   30:                 textViewThrottle = (TextView) view
   31:                                 .findViewById(R.id.textViewRCThrottle);
   32:                 textViewThrottle.setText("(Thrt: 0%)");
   33:                 textViewRudder = (TextView) view.findViewById(R.id.textViewRCRudde
r);
   34:                 textViewRudder.setText("(Rudd: 0%)");
   35:                 textViewElevator = (TextView) view
   36:                                 .findViewById(R.id.textViewRCElevator);
   37:                 textViewElevator.setText("(Elev: 0%)");
   38:                 textViewAileron = (TextView) view.findViewById(R.id.textViewRCAile
ron);
   39:                 textViewAileron.setText("(Ail: 0%)");
   40:
   41:                 joystickL = (JoystickView) view.findViewById(R.id.joystickViewL);
   42:                 joystickR = (JoystickView) view.findViewById(R.id.joystickViewR);
   43:                 joystickL.setOnJostickMovedListener(lJoystick);
   44:                 joystickR.setOnJostickMovedListener(rJoystick);
   45:                 return view;
   46:         }
   47:
   48:         @Override
   49:         public void onResume() {
   50:                 SharedPreferences prefs = PreferenceManager
   51:                                 .getDefaultSharedPreferences(getActivity()
   52:                                                 .getApplicationContext());
   53:                 rcIsMode1 = prefs.getString("pref_rc_mode", "MODE2").equalsIgnoreC
ase(
   54:                                 "MODE1");
   55:                 if (rcIsMode1) {
   56:                         joystickL.setAxisAutoReturnToCenter(true, true);
   57:                         joystickR.setAxisAutoReturnToCenter(
   58:                                         prefs.getBoolean("pref_rc_throttle_returnt
ocenter", false),
   59:                                         true);
   60:                         joystickL.setYAxisInverted(prefs.getBoolean(
   61:                                         "pref_rc_elevator_reverse", false));
   62:                         joystickL.setXAxisInverted(prefs.getBoolean(
   63:                                         "pref_rc_rudder_reverse", false));
   64:                         joystickR.setYAxisInverted(prefs.getBoolean(
   65:                                         "pref_rc_throttle_reverse", false));
   66:                         joystickR.setXAxisInverted(prefs.getBoolean(
   67:                                         "pref_rc_aileron_reverse", false));
   68:                 } else { // else Mode2
   69:                         joystickL.setAxisAutoReturnToCenter(
   70:                                         prefs.getBoolean("pref_rc_throttle_returnt
ocenter", false),
   71:                                         true);
   72:                         joystickR.setAxisAutoReturnToCenter(true, true);
   73:                         joystickL.setYAxisInverted(prefs.getBoolean(
   74:                                         "pref_rc_throttle_reverse", false));
   75:                         joystickL.setXAxisInverted(prefs.getBoolean(
   76:                                         "pref_rc_rudder_reverse", false));
   77:                         joystickR.setYAxisInverted(prefs.getBoolean(
   78:                                         "pref_rc_elevator_reverse", false));
   79:                         joystickR.setXAxisInverted(prefs.getBoolean(
   80:                                         "pref_rc_aileron_reverse", false));
   81:                 }
   82:                 super.onResume();
   83:         }
   84:
   85:         @Override
   86:         public void onStop() {
   87:                 super.onDestroyView();
   88:         }
   89:
   90:         public boolean physicalJoyMoved(MotionEvent ev) {
   91:                 // Tested only for wikipad controller. Probably works with most ga
me
   92:                 // controllers.
   93:                 if ((ev.getSource() & InputDevice.SOURCE_CLASS_JOYSTICK) != 0) {
   94:                         lJoystick.OnMoved((double) ev.getAxisValue(MotionEvent.AXI
S_X),
   95:                                         (double) ev.getAxisValue(MotionEvent.AXIS_
Y));
   96:                         rJoystick.OnMoved((double) ev.getAxisValue(MotionEvent.AXI
S_Z),
   97:                                         (double) ev.getAxisValue(MotionEvent.AXIS_
RZ));
   98:                         return true;
   99:                 }
  100:                 return false;
  101:         }
  102:
  103:         JoystickMovedListener lJoystick = new JoystickMovedListener() {
  104:                 @Override
  105:                 public void OnReturnedToCenter() {
  106:                 }
  107:
  108:                 @Override
  109:                 public void OnReleased() {
  110:                 }
  111:
  112:                 @Override
  113:                 public void OnMoved(double pan, double tilt) {
  114:                         textViewRudder.setText(String.format("Rudd: %.0f%%", pan *
 100));
  115:                         if (rcIsMode1) {
  116:                                 textViewElevator.setText(String.format("Elev: %.0f
%%",
  117:                                                 tilt * 100));
  118:                         } else {
  119:                                 textViewThrottle.setText(String.format("Thrt: %.0f
%%",
  120:                                                 tilt * 100));
```

```
  121:                         }
  122:                 }
  123:         };
  124:         JoystickMovedListener rJoystick = new JoystickMovedListener() {
  125:                 @Override
  126:                 public void OnReturnedToCenter() {
  127:                 }
  128:
  129:                 @Override
  130:                 public void OnReleased() {
  131:                 }
  132:
  133:                 @Override
  134:                 public void OnMoved(double pan, double tilt) {
  135:                         textViewAileron.setText(String.format("Ail: %.0f%%", pan *
 100));
  136:                         if (rcIsMode1) {
  137:                                 textViewThrottle.setText(String.format("Thrt: %.0f
%%",
  138:                                                 tilt * 100));
  139:                         } else {
  140:                                 textViewElevator.setText(String.format("Elev: %.0f
%%",
  141:                                                 tilt * 100));
  142:                         }
  143:
  144:                 }
  145:         };
  146:
  147: }
```

```java
package com.droidplanner.fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.droidplanner.DroidPlannerApp;
import com.droidplanner.R;
import com.droidplanner.MAVLink.MavLinkStreamRates;
import com.droidplanner.drone.Drone;
import com.droidplanner.drone.DroneInterfaces.OnRcDataChangedListner;
import com.droidplanner.widgets.FillBar.FillBarWithText;
import com.droidplanner.widgets.RcStick.RcStick;

public class RcSetupFragment extends Fragment implements OnRcDataChangedListner {
        private static final int RC_MIN = 1000;
        private static final int RC_MAX = 2000;

        // Extreme RC update rate in this screen
        private static final int RC_MSG_RATE = 50;

        private Drone drone;

        private FillBarWithText bar5;
        private FillBarWithText bar6;
        private FillBarWithText bar7;
        private FillBarWithText bar8;

        private RcStick stickLeft;

        private RcStick stickRight;

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
                drone = ((DroidPlannerApp) getActivity().getApplication()).drone;
                View view = inflater.inflate(R.layout.fragment_rc_setup, container
,
                                false);
                setupLocalViews(view);

                drone.RC.setListner(this);
                return view;
        }

        private void setupLocalViews(View view) {
                stickLeft = (RcStick) view.findViewById(R.id.stickLeft);
                stickRight = (RcStick) view.findViewById(R.id.stickRight);
                bar5 = (FillBarWithText) view.findViewById(R.id.fillBar5);
                bar6 = (FillBarWithText) view.findViewById(R.id.fillBar6);
                bar7 = (FillBarWithText) view.findViewById(R.id.fillBar7);
                bar8 = (FillBarWithText) view.findViewById(R.id.fillBar8);

                bar5.setup("CH 5", RC_MAX, RC_MIN);
                bar6.setup("CH 6", RC_MAX, RC_MIN);
                bar7.setup("CH 7", RC_MAX, RC_MIN);
                bar8.setup("CH 8", RC_MAX, RC_MIN);
        }

        @Override
        public void onStart() {
                super.onStart();
                setupDataStreamingForRcSetup();
        }

        private void setupDataStreamingForRcSetup() {
                MavLinkStreamRates.setupStreamRates(drone.MavClient, 1, 0, 1, 1, 1
,
                                RC_MSG_RATE, 0, 0);
        }

        @Override
        public void onStop() {
                super.onStop();
                MavLinkStreamRates
                                .setupStreamRatesFromPref((DroidPlannerApp) getAct
ivity()
                                                .getApplication());
        }

        @Override
        public void onNewInputRcData() {
                int[] data = drone.RC.in;
                bar5.setValue(data[4]);
                bar6.setValue(data[5]);
                bar7.setValue(data[6]);
                bar8.setValue(data[7]);

                float x,y;
                x = (data[3] - RC_MIN) / ((float) (RC_MAX - RC_MIN))*2-1;
                y = (data[2] - RC_MIN) / ((float) (RC_MAX - RC_MIN))*2-1;
                stickLeft.setPosition(x, y);

                x = (data[0] - RC_MIN) / ((float) (RC_MAX - RC_MIN))*2-1;
                y = (data[1] - RC_MIN) / ((float) (RC_MAX - RC_MIN))*2-1;
                stickRight.setPosition(x, -y);
        }

        @Override
        public void onNewOutputRcData() {
                // TODO Auto-generated method stub
        }
}
```

```
    1: package com.droidplanner.fragments;
    2:
    3: import android.content.SharedPreferences;
    4: import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
    5: import android.content.pm.PackageManager.NameNotFoundException;
    6: import android.os.Bundle;
    7: import android.preference.EditTextPreference;
    8: import android.preference.PreferenceFragment;
    9: import android.preference.PreferenceManager;
   10:
   11: import com.droidplanner.DroidPlannerApp;
   12: import com.droidplanner.R;
   13: import com.droidplanner.file.DirectoryPath;
   14:
   15: public class SettingsFragment extends PreferenceFragment implements
   16:                 OnSharedPreferenceChangeListener {
   17:
   18:         @Override
   19:         public void onCreate(Bundle savedInstanceState) {
   20:                 super.onCreate(savedInstanceState);
   21:                 addPreferencesFromResource(R.xml.preferences);
   22:                 SharedPreferences sharedPref = PreferenceManager
   23:                                 .getDefaultSharedPreferences(getActivity());
   24:
   25:                 findPreference("pref_connection_type").setSummary(
   26:                                 sharedPref.getString("pref_connection_type", ""));
   27:                 findPreference("pref_baud_type").setSummary(
   28:                                 sharedPref.getString("pref_baud_type", ""));
   29:                 findPreference("pref_max_fligth_path_size").setSummary(
   30:                                 sharedPref.getString("pref_max_fligth_path_size",
""")
   31:                                                 + " (set to zero to disable).");
   32:                 findPreference("pref_server_ip").setSummary(
   33:                                 sharedPref.getString("pref_server_ip", ""));
   34:                 findPreference("pref_server_port").setSummary(
   35:                                 sharedPref.getString("pref_server_port", ""));
   36:                 findPreference("pref_udp_server_port").setSummary(
   37:                                 sharedPref.getString("pref_udp_server_port", ""));
   38:                 findPreference("pref_map_type").setSummary(
   39:                                 sharedPref.getString("pref_map_type", ""));
   40:                 findPreference("pref_param_metadata").setSummary(
   41:                                 sharedPref.getString("pref_param_metadata", ""));
   42:                 if (sharedPref.getString("pref_rc_mode", "MODE2").equalsIgnoreCase
(
   43:                                 "MODE1")) {
   44:                         findPreference("pref_rc_mode").setSummary(
   45:                                         "Mode1: Throttle on RIGHT stick");
   46:                 } else {
   47:                         findPreference("pref_rc_mode").setSummary(
   48:                                         "Mode2: Throttle on LEFT stick");
   49:                 }
   50:                 findPreference("pref_rc_quickmode_left").setSummary(
   51:                                 sharedPref.getString("pref_rc_quickmode_left", "")
);
   52:                 findPreference("pref_rc_quickmode_right").setSummary(
   53:                                 sharedPref.getString("pref_rc_quickmode_right", ""
));
   54:
   55:                 findPreference("pref_storage").setSummary(
   56:                                 DirectoryPath.getDroidPlannerPath());
   57:
   58:                 try {
   59:                         EditTextPreference versionPref = (EditTextPreference) find
Preference("pref_version");
   60:                         String version = getActivity().getPackageManager().getPack
ageInfo(
   61:                                         getActivity().getPackageName(), 0).version
Name;
   62:                         versionPref.setSummary(version);
   63:                 } catch (NameNotFoundException e) {
   64:                         e.printStackTrace();
   65:                 }
   66:         }
   67:
   68:         public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
   69:                         String key) {
   70:                 if (key.equals("pref_connection_type")) {
   71:                         findPreference(key)
   72:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
   73:                 }
   74:                 if (key.equals("pref_baud_type")) {
   75:                         findPreference(key)
   76:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
   77:                 }
   78:                 if (key.equals("pref_max_fligth_path_size")) {
   79:                         findPreference(key).setSummary(
   80:                                         sharedPreferences
   81:                                                         .getString("pref_max_fligt
h_path_size", "")
   82:                                                         + " (set to zero to disabl
e).");
   83:                 }
   84:                 if (key.equals("pref_server_ip")) {
   85:                         findPreference(key)
   86:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
   87:                 }
   88:                 if (key.equals("pref_server_port")) {
   89:                         findPreference(key)
   90:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
   91:                 }
   92:                 if (key.equals("pref_map_type")) {
   93:                         findPreference(key)
   94:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
   95:                         ((DroidPlannerApp) getActivity().getApplication()).drone
   96:                                         .notifyMapTypeChanged();
   97:                 }
   98:                 if (key.equals("pref_param_metadata")) {
   99:                         findPreference(key)
  100:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
  101:                         ((DroidPlannerApp) getActivity().getApplication()).drone.p
arameters
  102:                                         .notifyParameterMetadataChanged();
  103:                 }
  104:                 if (key.equals("pref_rc_mode")) {
  105:                         if (sharedPreferences.getString(key, "MODE2").equalsIgnore
Case(
  106:                                         "MODE1")) {
  107:                                 findPreference(key)
  108:                                                 .setSummary("Mode1: Throttle on RI
GHT stick");
  109:                         } else {
  110:                                 findPreference(key).setSummary("Mode2: Throttle on
 LEFT stick");
  111:                         }
  112:                 }
  113:                 if (key.equals("pref_rc_quickmode_left")) {
  114:                         findPreference(key)
  115:                                         .setSummary(sharedPreferences.getString(ke
```

```
y, ""));
  116:                 }
  117:                 if (key.equals("pref_rc_quickmode_right")) {
  118:                         findPreference(key)
  119:                                         .setSummary(sharedPreferences.getString(ke
y, ""));
  120:                 }
  121:         }
  122:
  123:         @Override
  124:         public void onResume() {
  125:                 super.onResume();
  126:                 getPreferenceScreen().getSharedPreferences()
  127:                                 .registerOnSharedPreferenceChangeListener(this);
  128:         }
  129:
  130:         @Override
  131:         public void onPause() {
  132:                 super.onPause();
  133:                 getPreferenceScreen().getSharedPreferences()
  134:                                 .unregisterOnSharedPreferenceChangeListener(this);
  135:         }
  136:
  137: }
```

```
 1: package com.droidplanner.fragments;
 2:
 3: import android.app.Fragment;
 4: import android.os.Bundle;
 5: import android.view.LayoutInflater;
 6: import android.view.View;
 7: import android.view.ViewGroup;
 8: import android.widget.TextView;
 9:
10: import com.droidplanner.DroidPlannerApp;
11: import com.droidplanner.R;
12: import com.droidplanner.drone.Drone;
13: import com.droidplanner.drone.DroneInterfaces.HudUpdatedListner;
14: import com.droidplanner.widgets.newHUD.newHUD;
15:
16: public class TelemetryFragment extends Fragment implements HudUpdatedListner {
17:
18:         private newHUD hud;
19:         private Drone drone;
20:         private TextView roll;
21:         private TextView yaw;
22:         private TextView pitch;
23:         private TextView groundSpeed;
24:         private TextView airSpeed;
25:         private TextView climbRate;
26:         private TextView altitude;
27:
28:         @Override
29:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
30:                         Bundle savedInstanceState) {
31:                 View view = inflater.inflate(R.layout.fragment_telemetry, containe
r,
32:                                 false);
33:                 hud = (newHUD) view.findViewById(R.id.hudView);
34:
35:                 roll = (TextView) view.findViewById(R.id.rollValueText);
36:                 yaw = (TextView) view.findViewById(R.id.yawValueText);
37:                 pitch = (TextView) view.findViewById(R.id.pitchValueText);
38:
39:
40:                 groundSpeed = (TextView) view.findViewById(R.id.groundSpeedValue);
41:                 airSpeed = (TextView) view.findViewById(R.id.airSpeedValue);
42:                 climbRate = (TextView) view.findViewById(R.id.climbRateValue);
43:                 altitude = (TextView) view.findViewById(R.id.altitudeValue);
44:
45:                 drone = ((DroidPlannerApp) getActivity().getApplication()).drone;
46:                 drone.setHudListner(this);
47:                 return view;
48:         }
49:
50:         @Override
51:         public void onOrientationUpdate() {
52:                 float r = (float) drone.orientation.getRoll();
53:                 float p = (float) drone.orientation.getPitch();
54:                 float y = (float) drone.orientation.getYaw();
55:
56:                 hud.setAttitude(r, p, y);
57:
58:                 roll.setText(String.format("%3.0fÂº", r));
59:                 pitch.setText(String.format("%3.0fÂº", p));
60:                 yaw.setText(String.format("%3.0fÂº", y));
61:
62:         }
63:
64:         @Override
65:         public void onSpeedAltitudeAndClimbRateUpdate() {
66:                 airSpeed.setText(String.format("%3.1fÂº", drone.speed.getAirSpeed(
)));
67:                 groundSpeed.setText(String.format("%3.1fÂº", drone.speed.getGround
Speed()));
68:                 climbRate.setText(String.format("%3.1f", drone.speed.getVerticalSp
eed()));
69:                 double alt = drone.altitude.getAltitude();
70:                 altitude.setText(String.format("%3.0f\n%3.0f\n%3.0f", alt+1,alt,al
t-1));
71:         }
72:
73: }
```

```
   1: package com.droidplanner.fragments;
   2:
   3: import android.app.Fragment;
   4: import android.graphics.Color;
   5: import android.os.Bundle;
   6: import android.view.LayoutInflater;
   7: import android.view.View;
   8: import android.view.ViewGroup;
   9:
  10: import com.droidplanner.DroidPlannerApp;
  11: import com.droidplanner.R;
  12: import com.droidplanner.MAVLink.MavLinkStreamRates;
  13: import com.droidplanner.drone.Drone;
  14: import com.droidplanner.drone.DroneInterfaces.OnTuningDataListner;
  15: import com.droidplanner.parameters.Parameter;
  16: import com.droidplanner.widgets.SeekBarWithText.SeekBarWithText;
  17: import com.droidplanner.widgets.graph.Chart;
  18: import com.droidplanner.widgets.graph.ChartSeries;
  19:
  20: public class TuningFragment extends Fragment implements OnTuningDataListner {
  21:
  22:         private static final int NAV_MSG_RATE = 50;
  23:         private static final int CHART_BUFFER_SIZE = 20*NAV_MSG_RATE; // About 20s
of data on the buffer
  24:
  25:
  26:         private Drone drone;
  27:
  28:         private Chart topChart;
  29:         private Chart bottomChart;
  30:
  31:         private SeekBarWithText rollPSeekBar;
  32:         private SeekBarWithText rollDSeekBar;
  33:         private SeekBarWithText yawPSeekBar;
  34:         private SeekBarWithText thrAclSeekBar;
  35:
  36:         private Parameter rollP;
  37:         private Parameter rollD;
  38:         private Parameter yawP;
  39:         private Parameter thrAcl;
  40:
  41:         private ChartSeries bottomDataReference;
  42:
  43:         private ChartSeries bottomDataValue;
  44:
  45:         private ChartSeries topDataReference;
  46:
  47:         private ChartSeries topDataValue;
  48:
  49:         @Override
  50:         public View onCreateView(LayoutInflater inflater, ViewGroup container,
  51:                         Bundle savedInstanceState) {
  52:                 View view = inflater.inflate(R.layout.fragment_tunning, container,
  53:                                 false);
  54:                 setupLocalViews(view);
  55:                 setupCharts();
  56:
  57:                 drone = ((DroidPlannerApp) getActivity().getApplication()).drone;
  58:                 drone.setTuningDataListner(this);
  59:
  60:                 return view;
  61:         }
  62:
  63:         @Override
  64:         public void onStart() {
  65:                 super.onStart();
  66:                 setupDataStreamingForTuning();
  67:         }
  68:
  69:         private void setupDataStreamingForTuning() {
  70:                 // Sets the nav messages at 50Hz and other messages at a low rate
1Hz
  71:                 MavLinkStreamRates.setupStreamRates(drone.MavClient, 1, 0, 1, 1, 1
, 0, 0, NAV_MSG_RATE);
  72:         }
  73:
  74:         @Override
  75:         public void onStop() {
  76:                 super.onStop();
  77:                 MavLinkStreamRates.setupStreamRatesFromPref((DroidPlannerApp) getA
ctivity().getApplication());
  78:         }
  79:
  80:         private void setupLocalViews(View view) {
  81:                 topChart = (Chart) view.findViewById(R.id.chartTop);
  82:                 bottomChart = (Chart) view.findViewById(R.id.chartBottom);
  83:
  84:                 rollPSeekBar = (SeekBarWithText) view
  85:                                 .findViewById(R.id.SeekBarRollPitchControl);
  86:                 rollDSeekBar = (SeekBarWithText) view
  87:                                 .findViewById(R.id.SeekBarRollPitchDampenning);
  88:                 yawPSeekBar = (SeekBarWithText) view
  89:                                 .findViewById(R.id.SeekBarYawControl);
  90:                 thrAclSeekBar = (SeekBarWithText) view
  91:                                 .findViewById(R.id.SeekBarThrottleAccel);
  92:         }
  93:
  94:         private void setupCharts() {
  95:                 topDataReference = new ChartSeries(800);
  96:                 topDataReference.setColor(Color.BLUE);
  97:                 topDataReference.enable();
  98:                 topChart.series.add(topDataReference);
  99:                 topDataValue = new ChartSeries(800);
 100:                 topDataValue.setColor(Color.WHITE);
 101:                 topDataValue.enable();
 102:                 topChart.series.add(topDataValue);
 103:
 104:                 bottomDataReference = new ChartSeries(CHART_BUFFER_SIZE);
 105:                 bottomDataReference.setColor(Color.BLUE);
 106:                 bottomDataReference.enable();
 107:                 bottomChart.series.add(bottomDataReference);
 108:                 bottomDataValue = new ChartSeries(CHART_BUFFER_SIZE);
 109:                 bottomDataValue.setColor(Color.WHITE);
 110:                 bottomDataValue.enable();
 111:                 bottomChart.series.add(bottomDataValue);
 112:
 113:         }
 114:
 115:         @Override
 116:         public void onNewOrientationData() {
 117:                  bottomDataValue.newData(drone.orientation.getPitch());
 118:                  topDataValue.newData(drone.orientation.getRoll());
 119:                  bottomDataReference.newData(drone.navigation.getNavPitch());

 120:                  topDataReference.newData(drone.navigation.getNavRoll());
 121:                  bottomChart.update();
 122:                  topChart.update();
 123:         }
 124:
 125:
 126:         @Override
 127:         public void onNewNavigationData() {
 128:         }
 129:
```

```
130: }
```

```
 1: package com.droidplanner.gcp;
 2:
 3: import android.content.Context;
 4:
 5: import com.droidplanner.fragments.markers.GcpMarker;
 6: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 7: import com.google.android.gms.maps.model.LatLng;
 8: import com.google.android.gms.maps.model.Marker;
 9: import com.google.android.gms.maps.model.MarkerOptions;
10:
11: public class Gcp implements MarkerSource {
12:         public LatLng coord;
13:         public boolean isMarked;
14:
15:         public Gcp(double lat, double lng) {
16:                 this.coord = new LatLng(lat, lng);
17:                 this.isMarked = false;
18:         }
19:
20:         public void toogleState() {
21:                 isMarked = !isMarked;
22:         }
23:
24:         @Override
25:         public MarkerOptions build(Context context) {
26:                 return GcpMarker.build(this);
27:         }
28:
29:         @Override
30:         public void update(Marker markerFromGcp, Context context) {
31:                 GcpMarker.update(markerFromGcp, this);
32:         }
33: }
```

```java
package com.droidplanner.gps;

import android.app.AlertDialog;
import android.app.Service;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.Settings;
import android.util.Log;

public class GPS extends Service implements LocationListener {

    private final Context mContext;

    // flag for GPS status
    boolean isGPSEnabled = false;

    // flag for network status
    boolean isNetworkEnabled = false;

    // flag for GPS status
    boolean canGetLocation = false;

    Location location; // location
    double latitude; // latitude
    double longitude; // longitude

    // The minimum distance to change Updates in meters
    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

    // The minimum time between updates in milliseconds
    private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute

    // Declaring a Location Manager
    protected LocationManager locationManager;

    public GPS(Context context) {
        this.mContext = context;
        getLocation();
    }

    public Location getLocation() {
        try {
            locationManager = (LocationManager) mContext
                    .getSystemService(LOCATION_SERVICE);

            // getting GPS status
            isGPSEnabled = locationManager
                    .isProviderEnabled(LocationManager.GPS_PROVIDER);

            // getting network status
            isNetworkEnabled = locationManager
                    .isProviderEnabled(LocationManager.NETWORK_PROVIDER);

            if (!isGPSEnabled && !isNetworkEnabled) {
                // no network provider is enabled
            } else {
                this.canGetLocation = true;
                // First get location from Network Provider
                if (isNetworkEnabled) {
                    locationManager.requestLocationUpdates(
                            LocationManager.NETWORK_PROVIDER,
                            MIN_TIME_BW_UPDATES,
                            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
                    Log.d("Network", "Network");
                    if (locationManager != null) {
                        location = locationManager
                                .getLastKnownLocation(LocationManager.NETWORK_PROV
IDER);
                        if (location != null) {
                            latitude = location.getLatitude();
                            longitude = location.getLongitude();
                        }
                    }
                }
                // if GPS Enabled get lat/long using GPS Services
                if (isGPSEnabled) {
                    if (location == null) {
                        locationManager.requestLocationUpdates(
                                LocationManager.GPS_PROVIDER,
                                MIN_TIME_BW_UPDATES,
                                MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
                        Log.d("GPS Enabled", "GPS Enabled");
                        if (locationManager != null) {
                            location = locationManager
                                    .getLastKnownLocation(LocationManager.GPS_PROV
IDER);
                            if (location != null) {
                                latitude = location.getLatitude();
                                longitude = location.getLongitude();
                            }
                        }
                    }
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return location;
    }

    /**
     * Stop using GPS listener
     * Calling this function will stop using GPS in your app
     * */
    public void stopUsingGPS(){
        if(locationManager != null){
            locationManager.removeUpdates(GPS.this);
        }
    }

    /**
     * Function to get latitude
     * */
    public double getLatitude(){
        if(location != null){
            latitude = location.getLatitude();
        }

        // return latitude
        return latitude;
    }

    /**
     * Function to get longitude
     * */
    public double getLongitude(){
```

```
  133:         if(location != null){
  134:             longitude = location.getLongitude();
  135:         }
  136:
  137:         // return longitude
  138:         return longitude;
  139:     }
  140:
  141:     /**
  142:      * Function to check GPS/wifi enabled
  143:      * @return boolean
  144:      * */
  145:     public boolean canGetLocation() {
  146:         return this.canGetLocation;
  147:     }
  148:
  149:     /**
  150:      * Function to show settings alert dialog
  151:      * On pressing Settings button will lauch Settings Options
  152:      * */
  153:     public void showSettingsAlert(){
  154:         AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);
  155:
  156:         // Setting Dialog Title
  157:         alertDialog.setTitle("GPS is settings");
  158:
  159:         // Setting Dialog Message
  160:         alertDialog.setMessage("GPS is not enabled. Do you want to go to settings
menu?");
  161:
  162:         // On pressing Settings button
  163:         alertDialog.setPositiveButton("Settings", new DialogInterface.OnClickListe
ner() {
  164:             public void onClick(DialogInterface dialog,int which) {
  165:                 Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTING
S);
  166:                 mContext.startActivity(intent);
  167:             }
  168:         });
  169:
  170:         // on pressing cancel button
  171:         alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListene
r() {
  172:             public void onClick(DialogInterface dialog, int which) {
  173:             dialog.cancel();
  174:             }
  175:         });
  176:
  177:         // Showing Alert Message
  178:         alertDialog.show();
  179:     }
  180:
  181:     @Override
  182:     public void onLocationChanged(Location location) {
  183:     }
  184:
  185:     @Override
  186:     public void onProviderDisabled(String provider) {
  187:     }
  188:
  189:     @Override
  190:     public void onProviderEnabled(String provider) {
  191:     }
  192:
  193:     @Override
  194:     public void onStatusChanged(String provider, int status, Bundle extras) {
  195:     }
  196:
  197:     @Override
  198:     public IBinder onBind(Intent arg0) {
  199:         return null;
  200:     }
  201:
  202: }
```

```
    1: package com.droidplanner.helpers;
    2:
    3: import android.util.Log;
    4: import android.view.InputDevice;
    5:
    6: public class ExternalJoystick {
    7:         public void printInputDevicesToLog() {
    8:                 int[] inputIds = InputDevice.getDeviceIds();
    9:                 Log.d("DEV", "Found " + inputIds.length);
   10:                 for (int i = 0; i < inputIds.length; i++) {
   11:                         InputDevice inputDevice = InputDevice.getDevice(inputIds[i
]);
   12:                         Log.d("DEV", "name:" + inputDevice.getName() + " Sources:"
   13:                                         + inputDevice.getSources());
   14:                 }
   15:         }
   16: }
```

```
    1: package com.droidplanner.helpers;
    2:
    3: import android.content.Context;
    4: import android.content.SharedPreferences;
    5: import android.location.Location;
    6: import android.location.LocationListener;
    7: import android.location.LocationManager;
    8: import android.os.Bundle;
    9: import android.preference.PreferenceManager;
   10: import android.widget.Toast;
   11:
   12: import com.droidplanner.drone.Drone;
   13: import com.google.android.gms.maps.model.LatLng;
   14:
   15: public class FollowMe implements LocationListener {
   16:         private static final long MIN_TIME_MS = 2000;
   17:         private static final float MIN_DISTANCE_M = 0;
   18:         private Context context;
   19:         private boolean followMeEnabled = false;
   20:         private LocationManager locationManager;
   21:         private Drone drone;
   22:
   23:         public FollowMe(Context context, Drone drone) {
   24:                 this.context = context;
   25:                 this.drone = drone;
   26:                 this.locationManager = (LocationManager) context
   27:                                 .getSystemService(Context.LOCATION_SERVICE);
   28:         }
   29:
   30:         public void toogleFollowMeState() {
   31:                 if (isEnabledInPreferences()) {
   32:                         if (isEnabled()) {
   33:                                 disableFollowMe();
   34:                         } else {
   35:                                 enableFollowMe();
   36:                         }
   37:                 } else {
   38:                         disableFollowMe();
   39:                 }
   40:         }
   41:
   42:         private void enableFollowMe() {
   43:                 Toast.makeText(context, "FollowMe Enabled", Toast.LENGTH_SHORT).sh
ow();
   44:
   45:                 // Register the listener with the Location Manager to receive loca
tion
   46:                 // updates
   47:                 locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDE
R,
   48:                                 MIN_TIME_MS, MIN_DISTANCE_M, this);
   49:
   50:                 followMeEnabled = true;
   51:         }
   52:
   53:         private void disableFollowMe() {
   54:                 Toast.makeText(context, "FollowMe Disabled", Toast.LENGTH_SHORT).s
how();
   55:                 locationManager.removeUpdates(this);
   56:                 followMeEnabled = false;
   57:         }
   58:
   59:         public boolean isEnabled() {
   60:                 return followMeEnabled;
   61:         }
   62:
   63:         @Override
   64:         public void onLocationChanged(Location location) {
   65:                 LatLng coord = new LatLng(location.getLatitude(), location.getLong
itude());
   66:                 // TODO find a better way to do the GUIDED altitude
   67:                 drone.guidedPoint.newGuidedPointWithCurrentAlt(coord);
   68:         }
   69:
   70:         @Override
   71:         public void onProviderDisabled(String provider) {
   72:         }
   73:
   74:         @Override
   75:         public void onProviderEnabled(String provider) {
   76:         }
   77:
   78:         @Override
   79:         public void onStatusChanged(String provider, int status, Bundle extras) {
   80:         }
   81:
   82:         private boolean isEnabledInPreferences() {
   83:                 SharedPreferences prefs = PreferenceManager
   84:                                 .getDefaultSharedPreferences(context);
   85:
   86:                 return prefs.getBoolean("pref_follow_me_mode_enabled", false);
   87:         }
   88: }
```

```
 1: package com.droidplanner.helpers.geoTools;
 2:
 3: import java.util.List;
 4:
 5: import com.droidplanner.helpers.units.Area;
 6: import com.droidplanner.polygon.Polygon;
 7: import com.google.android.gms.maps.model.LatLng;
 8:
 9: public class GeoTools {
10:         private static final double RADIUS_OF_EARTH = 6372797.560856d;
11:         public List<LatLng> waypoints;
12:
13:         public GeoTools() {
14:         }
15:
16:         /**
17:          * Returns the distance between two points
18:          *
19:          * @return distance between the points in degrees
20:          */
21:         public static Double getAproximatedDistance(LatLng p1, LatLng p2) {
22:                 return (Math.hypot((p1.latitude - p2.latitude),
23:                                 (p1.longitude - p2.longitude)));
24:         }
25:
26:         public static Double metersTolat(double meters) {
27:                 double radius_of_earth = 6378100.0;// # in meters
28:                 return Math.toDegrees(meters / radius_of_earth);
29:         }
30:
31:         public static Double latToMeters(double lat) {
32:                 double radius_of_earth = 6378100.0;// # in meters
33:                 return Math.toRadians(lat) * radius_of_earth;
34:         }
35:
36:         /**
37:          * Extrapolate latitude/longitude given a heading and distance thanks to
38:          * http://www.movable-type.co.uk/scripts/latlong.html
39:          *
40:          * @param origin
41:          *            Point of origin
42:          * @param bearing
43:          *            bearing to navigate
44:          * @param distance
45:          *            distance to be added
46:          * @return New point with the added distance
47:          */
48:         public static LatLng newCoordFromBearingAndDistance(LatLng origin,
49:                         double bearing, double distance) {
50:
51:                 double lat = origin.latitude;
52:                 double lon = origin.longitude;
53:                 double lat1 = Math.toRadians(lat);
54:                 double lon1 = Math.toRadians(lon);
55:                 double brng = Math.toRadians(bearing);
56:                 double dr = distance / RADIUS_OF_EARTH;
57:
58:                 double lat2 = Math.asin(Math.sin(lat1) * Math.cos(dr) + Math.cos(l
at1)
59:                                 * Math.sin(dr) * Math.cos(brng));
60:                 double lon2 = lon1
61:                                 + Math.atan2(Math.sin(brng) * Math.sin(dr) * Math.
cos(lat1),
62:                                                 Math.cos(dr) - Math.sin(lat1) * Ma
th.sin(lat2));
63:
64:                 return (new LatLng(Math.toDegrees(lat2), Math.toDegrees(lon2)));
65:         }
66:
67:         /**
68:          * Calculates the arc between two points
69:          * http://en.wikipedia.org/wiki/Haversine_formula
70:          *
71:          * @return the arc in degrees
72:          */
73:         static double getArcInRadians(LatLng from, LatLng to) {
74:
75:                 double latitudeArc = Math.toRadians(from.latitude - to.latitude);
76:                 double longitudeArc = Math.toRadians(from.longitude - to.longitude
);
77:
78:                 double latitudeH = Math.sin(latitudeArc * 0.5);
79:                 latitudeH *= latitudeH;
80:                 double lontitudeH = Math.sin(longitudeArc * 0.5);
81:                 lontitudeH *= lontitudeH;
82:
83:                 double tmp = Math.cos(Math.toRadians(from.latitude))
84:                                 * Math.cos(Math.toRadians(to.latitude));
85:                 return Math.toDegrees(2.0 * Math.asin(Math.sqrt(latitudeH + tmp
86:                                 * lontitudeH)));
87:         }
88:
89:         /**
90:          * Computes the distance between two coordinates
91:          *
92:          * @return distance in meters
93:          */
94:         public static double getDistance(LatLng from, LatLng to) {
95:                 return RADIUS_OF_EARTH * Math.toRadians(getArcInRadians(from, to))
;
96:         }
97:
98:         /**
99:          * Computes the heading between two coordinates
100:          *
101:          * @return heading in degrees
102:          */
103:         static double getHeadingFromCoordinates(LatLng fromLoc, LatLng toLoc) {
104:                 double fLat = Math.toRadians(fromLoc.latitude);
105:                 double fLng = Math.toRadians(fromLoc.longitude);
106:                 double tLat = Math.toRadians(toLoc.latitude);
107:                 double tLng = Math.toRadians(toLoc.longitude);
108:
109:                 double degree = Math.toDegrees(Math.atan2(
110:                                 Math.sin(tLng - fLng) * Math.cos(tLat),
111:                                 Math.cos(fLat) * Math.sin(tLat) - Math.sin(fLat)
112:                                                 * Math.cos(tLat) * Math.cos(tLng -
fLng)));
113:
114:                 if (degree >= 0) {
115:                         return degree;
116:                 } else {
117:                         return 360 + degree;
118:                 }
119:         }
120:
121:         /**
122:          * Experimental Function, needs testing! Calculate the area of the polygon
123:          *
124:          * @return area in mi¿½
125:          */
126:         // TODO test and fix this function
127:         public static Area getArea(Polygon poly) {
128:                 double sum = 0.0;
```

```
  129:                 int length = poly.getLatLngList().size();
  130:                 for (int i = 0; i < length - 1; i++) {
  131:                         sum = sum
  132:                                         + (latToMeters(poly.getLatLngList().get(i)
.longitude) * latToMeters(poly
  133:                                                         .getLatLngList().get(i + 1
).latitude))
  134:                                         - (latToMeters(poly.getLatLngList().get(i)
.latitude) * latToMeters(poly
  135:                                                         .getLatLngList().get(i + 1
).longitude));
  136:                 }
  137:                 sum = sum
  138:                                 + (latToMeters(poly.getLatLngList().get(length-1).
longitude) * latToMeters(poly
  139:                                 .getLatLngList().get(0).latitude))
  140:                                 - (latToMeters(poly.getLatLngList().get(length-1).
latitude) * latToMeters(poly
  141:                                                         .getLatLngList().get(0).longitude)
);
  142:                 return new Area(Math.abs(0.5 * sum));
  143:                 //return new Area(0);
  144:         }
  145:
  146:
  147: }
```

```
 1: package com.droidplanner.helpers.geoTools;
 2:
 3: import com.google.android.gms.maps.model.LatLng;
 4:
 5: public class LineLatLng {
 6:         public LatLng p1;
 7:         public LatLng p2;
 8:
 9:         public LineLatLng(LatLng p1, LatLng p2) {
10:                 this.p1 = p1;
11:                 this.p2 = p2;
12:         }
13:
14:         public LineLatLng(LineLatLng line) {
15:                 this(line.p1, line.p2);
16:         }
17:
18:         public LatLng getFarthestEndpointTo(LatLng point) {
19:                 if (getClosestEndpointTo(point).equals(p1)) {
20:                         return p2;
21:                 } else {
22:                         return p1;
23:                 }
24:         }
25:
26:         public LatLng getClosestEndpointTo(LatLng point) {
27:                 if (getDistanceToStart(point) < getDistanceToEnd(point)) {
28:                         return p1;
29:                 } else {
30:                         return p2;
31:                 }
32:         }
33:
34:         private Double getDistanceToEnd(LatLng point) {
35:                 return GeoTools.getAproximatedDistance(p2, point);
36:         }
37:
38:         private Double getDistanceToStart(LatLng point) {
39:                 return GeoTools.getAproximatedDistance(p1, point);
40:         }
41:
42: }
```

```
    1: package com.droidplanner.helpers.geoTools;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.google.android.gms.maps.model.LatLng;
    7:
    8: public class LineSampler {
    9:
   10:         private List<LatLng> points;
   11:         private List<LatLng> sampledPoints = new ArrayList<LatLng>();
   12:
   13:         public LineSampler(List<LatLng> points) {
   14:                 this.points = points;
   15:         }
   16:
   17:         public LineSampler(LatLng p1, LatLng p2) {
   18:                 points = new ArrayList<LatLng>();
   19:                 points.add(p1);
   20:                 points.add(p2);
   21:         }
   22:
   23:         public List<LatLng> sample(double sampleDistance) {
   24:                 for (int i = 1; i < points.size(); i++) {
   25:                         LatLng from = points.get(i - 1);
   26:                         LatLng to = points.get(i);
   27:                         sampledPoints.addAll(sampleLine(from, to, sampleDistance))
;
   28:                 }
   29:                 sampledPoints.add(getLast(points));
   30:                 return sampledPoints;
   31:         }
   32:
   33:         private List<LatLng> sampleLine(LatLng from, LatLng to,
   34:                         double samplingDistance) {
   35:                 List<LatLng> result = new ArrayList<LatLng>();
   36:                 double heading = GeoTools.getHeadingFromCoordinates(from, to);
   37:                 double totalLength = GeoTools.getDistance(from, to);
   38:                 double distance = 0;
   39:
   40:                 while (distance < totalLength) {
   41:                         result.add(GeoTools.newCoordFromBearingAndDistance(from, h
eading,
   42:                                         distance));
   43:                         distance += samplingDistance;
   44:                 }
   45:                 return result;
   46:         }
   47:
   48:         private LatLng getLast(List<LatLng> list) {
   49:                 return list.get(list.size() - 1);
   50:         }
   51:
   52: }
```

```
    1: package com.droidplanner.helpers.geoTools;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.droidplanner.polygon.PolyBounds;
    7: import com.google.android.gms.maps.model.LatLng;
    8:
    9: public class LineTools {
   10:
   11:         public static LineLatLng findExternalPoints(ArrayList<LatLng> crosses) {
   12:                 LatLng meanCoord = new PolyBounds(crosses).getMiddle();
   13:                 LatLng start = PointTools.findFarthestPoint(crosses, meanCoord);
   14:                 LatLng end = PointTools.findFarthestPoint(crosses, start);
   15:                 return new LineLatLng(start, end);
   16:         }
   17:
   18:         /**
   19:          * Finds the intersection of two lines http://stackoverflow.com/questions/
   20:          * 1119451/how-to-tell-if-a-line-intersects -a-polygon-in-c
   21:          *
   22:          * @throws Exception
   23:          */
   24:         public static LatLng FindLineIntersection(LineLatLng first,
   25:                         LineLatLng second) throws Exception {
   26:                 double denom = ((first.p2.longitude - first.p1.longitude) * (secon
d.p2.latitude - second.p1.latitude))
   27:                                 - ((first.p2.latitude - first.p1.latitude) * (seco
nd.p2.longitude - second.p1.longitude));
   28:                 if (denom == 0)
   29:                         throw new Exception("Parralel Lines");
   30:                 double numer = ((first.p1.latitude - second.p1.latitude) * (second
.p2.longitude - second.p1.longitude))
   31:                                 - ((first.p1.longitude - second.p1.longitude) * (s
econd.p2.latitude - second.p1.latitude));
   32:                 double r = numer / denom;
   33:                 double numer2 = ((first.p1.latitude - second.p1.latitude) * (first
.p2.longitude - first.p1.longitude))
   34:                                 - ((first.p1.longitude - second.p1.longitude) * (f
irst.p2.latitude - first.p1.latitude));
   35:                 double s = numer2 / denom;
   36:                 if ((r < 0 || r > 1) || (s < 0 || s > 1))
   37:                         throw new Exception("No Intersection");
   38:                 // Find intersection point
   39:                 double longitude = first.p1.longitude
   40:                                 + (r * (first.p2.longitude - first.p1.longitude));
   41:                 double latitude = first.p1.latitude
   42:                                 + (r * (first.p2.latitude - first.p1.latitude));
   43:                 return (new LatLng(latitude, longitude));
   44:         }
   45:
   46:         /**
   47:          * Finds the line that has the start or tip closest to a point.
   48:          *
   49:          * @param point
   50:          *            Point to the distance will be minimized
   51:          * @param list
   52:          *            A list of lines to search
   53:          * @return The closest Line
   54:          */
   55:         public static LineLatLng findClosestLineToPoint(LatLng point,
   56:                         List<LineLatLng> list) {
   57:                 LineLatLng answer = list.get(0);
   58:                 double shortest = Double.MAX_VALUE;
   59:
   60:                 for (LineLatLng line : list) {
   61:                         double ans1 = GeoTools.getAproximatedDistance(point, line.
p1);
   62:                         double ans2 = GeoTools.getAproximatedDistance(point, line.
p2);
   63:                         LatLng shorterpnt = ans1 < ans2 ? line.p1 : line.p2;
   64:
   65:                         if (shortest > GeoTools.getAproximatedDistance(point, shor
terpnt)) {
   66:                                 answer = line;
   67:                                 shortest = GeoTools.getAproximatedDistance(point,
shorterpnt);
   68:                         }
   69:                 }
   70:                 return answer;
   71:         }
   72:
   73: }
```

```java
package com.droidplanner.helpers.geoTools;

import java.util.ArrayList;
import java.util.List;

import com.google.android.gms.maps.model.LatLng;

public class PointTools {

        public static LatLng findFarthestPoint(ArrayList<LatLng> crosses,
                        LatLng middle) {
                double farthestDistance = Double.NEGATIVE_INFINITY;
                LatLng farthestPoint = null;
                for (LatLng cross : crosses) {
                        double distance = GeoTools.getAproximatedDistance(cross, m
iddle);
                        if (distance > farthestDistance) {
                                farthestPoint = cross;
                                farthestDistance = distance;
                        }
                }
                return farthestPoint;
        }

        /**
         * Finds the closest point in a list to another point
         *
         * @param point
         *            point that will be used as reference
         * @param list
         *            List of points to be searched
         * @return The closest point
         */
        @SuppressWarnings("unused")
        private static LatLng findClosestPoint(LatLng point, List<LatLng> list) {
                LatLng answer = null;
                double currentbest = Double.MAX_VALUE;

                for (LatLng pnt : list) {
                        double dist1 = GeoTools.getAproximatedDistance(point, pnt)
;

                        if (dist1 < currentbest) {
                                answer = pnt;
                                currentbest = dist1;
                        }
                }
                return answer;
        }

        /**
         * Finds the pair of adjacent points that minimize the distance to a
         * reference point
         *
         * @param point
         *            point that will be used as reference
         * @param waypoints2
         *            List of points to be searched
         * @return Position of the second point in the pair that minimizes the
         *         distance
         */
        static int findClosestPair(LatLng point, List<LatLng> waypoints2) {
                int answer = 0;
                double currentbest = Double.MAX_VALUE;
                double dist;
                LatLng p1, p2;

                for (int i = 0; i < waypoints2.size(); i++) {
                        if (i == waypoints2.size() - 1) {
                                p1 = waypoints2.get(i);
                                p2 = waypoints2.get(0);
                        } else {
                                p1 = waypoints2.get(i);
                                p2 = waypoints2.get(i + 1);
                        }

                        dist = PointTools.pointToLineDistance(p1, p2, point);
                        if (dist < currentbest) {
                                answer = i + 1;
                                currentbest = dist;
                        }
                }
                return answer;
        }

        /**
         * Provides the distance from a point P to the line segment that passes
         * through A-B. If the point is not on the side of the line, returns the
         * distance to the closest point
         *
         * @param L1
         *            First point of the line
         * @param L2
         *            Second point of the line
         * @param P
         *            Point to measure the distance
         */
        public static double pointToLineDistance(LatLng L1, LatLng L2, LatLng P) {
                double A = P.longitude - L1.longitude;
                double B = P.latitude - L1.latitude;
                double C = L2.longitude - L1.longitude;
                double D = L2.latitude - L1.latitude;

                double dot = A * C + B * D;
                double len_sq = C * C + D * D;
                double param = dot / len_sq;

                double xx, yy;

                if (param < 0) // point behind the segment
                {
                        xx = L1.longitude;
                        yy = L1.latitude;
                } else if (param > 1) // point after the segment
                {
                        xx = L2.longitude;
                        yy = L2.latitude;
                } else { // point on the side of the segment
                        xx = L1.longitude + param * C;
                        yy = L1.latitude + param * D;
                }

                return Math.hypot(xx - P.longitude, yy - P.latitude);
        }

}
```

```
    1: package com.droidplanner.helpers.geoTools;
    2:
    3: import java.util.List;
    4:
    5: import com.droidplanner.helpers.units.Length;
    6: import com.google.android.gms.maps.model.LatLng;
    7:
    8: public class PolylineTools {
    9:
   10:         /**
   11:          *      Total length of the polyline in meters
   12:          * @param points
   13:          * @return
   14:          */
   15:         public static Length getPolylineLength(List<LatLng> points) {
   16:                 double lenght = 0;
   17:                 for (int i = 1; i < points.size(); i++) {
   18:                         lenght+=GeoTools.getDistance(points.get(i),points.get(i-1)
);
   19:                 }
   20:                 return new Length(lenght);
   21:         }
   22:
   23: }
```

```
    1: package com.droidplanner.helpers.geoTools;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.graphics.Point;
    7:
    8: /**
    9:  * Based on the Ramerâ\200\223Douglasâ\200\223Peucker algorithm algorithm
   10:  * http://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm
   11:  */
   12: public class Simplify {
   13:         public static List<Point> simplify(List<Point> list, double tolerance) {    );
   14:                 int index = 0;
   15:                 double dmax = 0;
   16:                 double squareTolerance = tolerance*tolerance;
   17:                 int lastIndex = list.size() - 1;
   18:
   19:                 // Find the point with the maximum distance
   20:                 for (int i = 1; i < list.size() - 1; i++) {
   21:                         double d = pointToLineDistance(list.get(0), list.get(lastI
ndex),
   22:                                         list.get(i));
   23:                         if (d > dmax) {
   24:                                 index = i;
   25:                                 dmax = d;
   26:                         }
   27:                 }
   28:
   29:                 // If max distance is greater than epsilon, recursively simplify
   30:                 List<Point> ResultList = new ArrayList<Point>();
   31:                 if (dmax > squareTolerance) {
   32:                         // Recursive call
   33:                         List<Point> recResults1 = simplify(list.subList(0, index +
 1),
   34:                                         tolerance);
   35:                         List<Point> recResults2 = simplify(
   36:                                         list.subList(index, lastIndex + 1), tolera
nce);
   37:
   38:                         // Build the result list
   39:                         recResults1.remove(recResults1.size() - 1);
   40:                         ResultList.addAll(recResults1);
   41:                         ResultList.addAll(recResults2);
   42:                 } else {
   43:                         ResultList.add(list.get(0));
   44:                         ResultList.add(list.get(lastIndex));
   45:                 }
   46:
   47:                 // Return the result
   48:                 return ResultList;
   49:         }
   50:
   51:         /**
   52:          * Perpendicular Distance of point to line
   53:          *
   54:          * @param L1
   55:          *            First point of the line
   56:          * @param L2
   57:          *            Second point of the line
   58:          * @param P
   59:          *            Point to measure the distance
   60:          * @return The square distance
   61:          */
   62:         public static double pointToLineDistance(Point L1, Point L2, Point P) {
   63:                 double x0, y0, x1, y1, x2, y2, dx, dy, t;
   64:
   65:                 x1 = L1.x;
   66:                 y1 = L1.y;
   67:                 x2 = L2.x;
   68:                 y2 = L2.y;
   69:                 x0 = P.x;
   70:                 y0 = P.y;
   71:
   72:                 dx = x2 - x1;
   73:                 dy = y2 - y1;
   74:
   75:                 if (dx != 0.0d || dy != 0.0d) {
   76:                         t = ((x0 - x1) * dx + (y0 - y1) * dy) / (dx * dx + dy * dy
   77:
   78:                         if (t > 1.0d) {
   79:                                 x1 = x2;
   80:                                 y1 = y2;
   81:                         } else if (t > 0.0d) {
   82:                                 x1 += dx * t;
   83:                                 y1 += dy * t;
   84:                         }
   85:                 }
   86:
   87:                 dx = x0 - x1;
   88:                 dy = y0 - y1;
   89:
   90:                 return dx * dx + dy * dy;
   91:         }
   92: }
```

```
 1: package com.droidplanner.helpers;
 2:
 3: import java.io.ByteArrayOutputStream;
 4: import java.io.FileInputStream;
 5: import java.io.FileNotFoundException;
 6: import java.io.IOException;
 7: import java.util.Locale;
 8:
 9: import android.util.Log;
10:
11: import com.droidplanner.file.DirectoryPath;
12: import com.droidplanner.file.FileStream;
13: import com.google.android.gms.maps.model.Tile;
14: import com.google.android.gms.maps.model.TileProvider;
15:
16: /**
17:  * Title provider for a MapView from the local storage. Based on:
18:  * http://stackoverflow.com/questions/14784841/tileprovider-using-local-tiles
19:  *
20:  */
21: public class LocalMapTileProvider implements TileProvider {
22:         private static final int TILE_WIDTH = 256;
23:         private static final int TILE_HEIGHT = 256;
24:         private static final int BUFFER_SIZE = 16 * 1024;
25:
26:         public LocalMapTileProvider() {
27:                 tryToAddANoMediaFile();
28:         }
29:
30:         private void tryToAddANoMediaFile() {
31:                 try {
32:                         FileStream.createNoMediaFile();
33:                 } catch (Exception e) {
34:                         e.printStackTrace();
35:                 }
36:         }
37:
38:         @Override
39:         public Tile getTile(int x, int y, int zoom) {
40:                 byte[] image = readTileImage(x, y, zoom);
41:                 if (image == null) {
42:                         return NO_TILE;
43:                 } else {
44:                         return new Tile(TILE_WIDTH, TILE_HEIGHT, image);
45:                 }
46:
47:         }
48:
49:         private byte[] readTileImage(int x, int y, int zoom) {
50:                 FileInputStream in = null;
51:                 ByteArrayOutputStream buffer = null;
52:
53:                 try {
54:                         String patch = DirectoryPath.getMapsPath()
55:                                         + getTileFilename(x, y, zoom);
56:                         in = new FileInputStream(patch);
57:                         buffer = new ByteArrayOutputStream();
58:
59:                         int nRead;
60:                         byte[] data = new byte[BUFFER_SIZE];
61:
62:                         while ((nRead = in.read(data, 0, BUFFER_SIZE)) != -1) {
63:                                 buffer.write(data, 0, nRead);
64:                         }
65:                         buffer.flush();
66:
67:                         return buffer.toByteArray();
68:                 } catch (FileNotFoundException e) {
69:                         return null;
70:                 } catch (IOException e) {
71:                         return null;
72:                 } catch (OutOfMemoryError e) {
73:                         e.printStackTrace();
74:                         return null;
75:                 } finally {
76:                         if (in != null)
77:                                 try {
78:                                         in.close();
79:                                 } catch (Exception ignored) {
80:                                 }
81:                         if (buffer != null)
82:                                 try {
83:                                         buffer.close();
84:                                 } catch (Exception ignored) {
85:                                 }
86:                 }
87:         }
88:
89:         private String getTileFilename(int x, int y, int zoom) {
90:                 return String.format(Locale.US, "%d/%d/%d.jpg", zoom, y, x);
91:         }
92:
93: }
```

```
    1: package com.droidplanner.helpers;
    2:
    3: import java.util.Arrays;
    4: import java.util.concurrent.Executors;
    5: import java.util.concurrent.ScheduledExecutorService;
    6: import java.util.concurrent.TimeUnit;
    7:
    8: import android.content.Context;
    9: import android.content.SharedPreferences;
   10: import android.preference.PreferenceManager;
   11:
   12: import com.droidplanner.MAVLink.MavLinkRC;
   13: import com.droidplanner.drone.Drone;
   14:
   15: public class RcOutput {
   16:         private static final int DISABLE_OVERRIDE = 0;
   17:         private static final int RC_TRIM = 1500;
   18:         private static final int RC_RANGE = 550;
   19:         private Context parrentContext;
   20:         private ScheduledExecutorService scheduleTaskExecutor;
   21:         private Drone drone;
   22:         public int[] rcOutputs = new int[8];
   23:
   24:         public static final int AILERON = 0;
   25:         public static final int ELEVATOR = 1;
   26:         public static final int TROTTLE = 2;
   27:         public static final int RUDDER = 3;
   28:
   29:         public static final int RC5 = 4;
   30:         public static final int RC6 = 5;
   31:         public static final int RC7 = 6;
   32:         public static final int RC8 = 7;
   33:
   34:         public RcOutput(Drone drone, Context context) {
   35:                 this.drone = drone;
   36:                 parrentContext = context;
   37:         }
   38:
   39:         public void disableRcOverride() {
   40:                 if (isRcOverrided()) {
   41:                         scheduleTaskExecutor.shutdownNow();
   42:                         scheduleTaskExecutor = null;
   43:                 }
   44:                 Arrays.fill(rcOutputs, DISABLE_OVERRIDE);       // Start with all
channels disabled, external callers can enable them as desired
   45:                 MavLinkRC.sendRcOverrideMsg(drone, rcOutputs); // Just to be sure
send 3
   46:
                    // disable
   47:                 MavLinkRC.sendRcOverrideMsg(drone, rcOutputs);
   48:                 MavLinkRC.sendRcOverrideMsg(drone, rcOutputs);
   49:         }
   50:
   51:         public void enableRcOverride() {
   52:                 if (!isRcOverrided()) {
   53:                         Arrays.fill(rcOutputs, DISABLE_OVERRIDE);
   54:                         MavLinkRC.sendRcOverrideMsg(drone, rcOutputs); // Just to
be sure send 3
   55:                         MavLinkRC.sendRcOverrideMsg(drone, rcOutputs);
   56:                         MavLinkRC.sendRcOverrideMsg(drone, rcOutputs);
Arrays.fill(rcOutputs, DISABLE_OVERRIDE);
   57:                         scheduleTaskExecutor = Executors.newScheduledThreadPool(5)
;
   58:                         scheduleTaskExecutor.scheduleWithFixedDelay(new Runnable()
 {
   59:                                 @Override
   60:                                 public void run() {
   61:                                         MavLinkRC.sendRcOverrideMsg(drone, rcOutpu
ts);
   62:                                 }
   63:                         }, 0, getRcOverrideDelayMs(), TimeUnit.MILLISECONDS);
   64:                 }
   65:         }
   66:
   67:         private int getRcOverrideDelayMs() {
   68:                 SharedPreferences prefs = PreferenceManager
   69:                                 .getDefaultSharedPreferences(parrentContext);
   70:                 int rate = Integer.parseInt(prefs.getString(
   71:                                 "pref_mavlink_stream_rate_RC_override", "0"));
   72:                 if ((rate > 1) & (rate < 500)) {
   73:                         return 1000 / rate;
   74:                 } else {
   75:                         return 20;
   76:                 }
   77:         }
   78:
   79:         public boolean isRcOverrided() {
   80:                 return (scheduleTaskExecutor != null);
   81:         }
   82:
   83:         public void setRcChannel(int ch, double value) {
   84:                 if (value > +1)
   85:                         value = +1;
   86:                 if (value < -1)
   87:                         value = -1;
   88:                 rcOutputs[ch] = (int) (value * RC_RANGE + RC_TRIM);
   89:         }
   90:
   91: }
```

```
    1: package com.droidplanner.helpers;
    2:
    3: import android.content.Context;
    4: import android.location.Location;
    5: import android.location.LocationListener;
    6: import android.location.LocationManager;
    7: import android.os.Bundle;
    8: import android.widget.Toast;
    9:
   10: import com.droidplanner.drone.Drone;
   11: import com.google.android.gms.maps.model.LatLng;
   12:
   13: public class RecordMe implements LocationListener {
   14:         private static final long MIN_TIME_MS = 2000;
   15:         private static final float MIN_DISTANCE_M = 0;
   16:
   17:         private Context context;
   18:         private Drone drone;
   19:         private LocationManager locationManager;
   20:         private boolean recordMeEnabled = false;
   21:
   22:         public RecordMe(Context context, Drone drone) {
   23:                 this.context = context;
   24:                 this.drone = drone;
   25:                 this.locationManager = (LocationManager) context
   26:                                 .getSystemService(Context.LOCATION_SERVICE);
   27:         }
   28:
   29:         public void toogleRecordMeState() {
   30:                 if (isEnabled()) {
   31:                         finishRecordMe();
   32:                 } else {
   33:                         startRecordMe();
   34:                 }
   35:         }
   36:
   37:         private void startRecordMe() {
   38:                 Toast.makeText(context, "Record Enabled", Toast.LENGTH_SHORT).show
();
   39:                 locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDE
R,
   40:                                 MIN_TIME_MS, MIN_DISTANCE_M, this);
   41:                 recordMeEnabled = true;
   42:         }
   43:
   44:         private void finishRecordMe() {
   45:                 Toast.makeText(context, "Record Disabled", Toast.LENGTH_SHORT).sho
w();
   46:                 locationManager.removeUpdates(this);
   47:                 recordMeEnabled = false;
   48:         }
   49:
   50:         public boolean isEnabled() {
   51:                 return recordMeEnabled;
   52:         }
   53:
   54:         // @Override
   55:         public void onLocationChanged(Location location) {
   56:                 // TODO find a better way to do the altitude
   57:
   58:                 LatLng coord = new LatLng(location.getLatitude(), location.getLong
itude());
   59:                 drone.mission.addWaypoint(coord, drone.mission.getDefaultAlt());
   60:                 drone.mission.onMissionUpdate();
   61:         }
   62:
   63:         @Override
   64:         public void onProviderDisabled(String provider) {
   65:         }
   66:
   67:         @Override
   68:         public void onProviderEnabled(String provider) {
   69:         }
   70:
   71:         @Override
   72:         public void onStatusChanged(String provider, int status, Bundle extras) {
   73:         }
   74: }
```

```
    1: package com.droidplanner.helpers;
    2:
    3: import java.util.Locale;
    4:
    5: import android.content.Context;
    6: import android.content.SharedPreferences;
    7: import android.preference.PreferenceManager;
    8: import android.speech.tts.TextToSpeech;
    9: import android.speech.tts.TextToSpeech.OnInitListener;
   10:
   11: import com.MAVLink.Messages.ApmModes;
   12:
   13: public class TTS implements OnInitListener {
   14:         private static final double BATTERY_DISCHARGE_NOTIFICATION_EVERY_PERCENT =
 10;
   15:
   16:         TextToSpeech tts;
   17:         private SharedPreferences prefs;
   18:         private int lastBatteryDischargeNotification;
   19:
   20:         public TTS(Context context) {
   21:                 tts = new TextToSpeech(context, this);
   22:                 this.prefs = PreferenceManager.getDefaultSharedPreferences(context
);
   23:         }
   24:
   25:         @Override
   26:         public void onInit(int status) {
   27:                 tts.setLanguage(Locale.US);
   28:         }
   29:
   30:         public void speak(String string) {
   31:                 if (tts != null) {
   32:                         if (shouldEnableTTS()) {
   33:                                 tts.speak(string, TextToSpeech.QUEUE_FLUSH, null);
   34:                         }
   35:                 }
   36:         }
   37:
   38:         private boolean shouldEnableTTS() {
   39:                 return prefs.getBoolean("pref_enable_tts", false);
   40:         }
   41:
   42:         public void speakGpsMode(int fix) {
   43:                 switch (fix) {
   44:                 case 2:
   45:                         speak("GPS 2D Lock");
   46:                         break;
   47:                 case 3:
   48:                         speak("GPS 3D Lock");
   49:                         break;
   50:                 default:
   51:                         speak("Lost GPS Lock");
   52:                         break;
   53:                 }
   54:         }
   55:
   56:         public void speakArmedState(boolean armed) {
   57:                 if (armed) {
   58:                         speak("Armed");
   59:                 } else {
   60:                         speak("Disarmed");
   61:                 }
   62:         }
   63:
   64:         public void speakMode(ApmModes mode) {
   65:                 String modeString = "Mode ";
   66:                 switch (mode) {
   67:                 case FIXED_WING_FLY_BY_WIRE_A:
   68:                         modeString += "Fly by wire A";
   69:                         break;
   70:                 case FIXED_WING_FLY_BY_WIRE_B:
   71:                         modeString += "Fly by wire B";
   72:                         break;
   73:                 case ROTOR_ACRO:
   74:                         modeString += "Acrobatic";
   75:                         break;
   76:                 case ROTOR_ALT_HOLD:
   77:                         modeString += "Altitude hold";
   78:                         break;
   79:                 case ROTOR_POSITION:
   80:                         modeString += "Position hold";
   81:                         break;
   82:                 case FIXED_WING_RTL:
   83:                 case ROTOR_RTL:
   84:                         modeString += "Return to home";
   85:                         break;
   86:                 default:
   87:                         modeString += mode.getName();
   88:                         break;
   89:                 }
   90:                 speak(modeString);
   91:         }
   92:
   93:         public void batteryDischargeNotification(double battRemain) {
   94:                 if (lastBatteryDischargeNotification != (int) ((battRemain - 1) /
BATTERY_DISCHARGE_NOTIFICATION_EVERY_PERCENT)) {
   95:                         lastBatteryDischargeNotification = (int) ((battRemain - 1)
 / BATTERY_DISCHARGE_NOTIFICATION_EVERY_PERCENT);
   96:                         speak("Battery at" + (int) battRemain + "%");
   97:                 }
   98:         }
   99: }
```

```
1: package com.droidplanner.helpers.units;
2:
3: public class Altitude extends Length {
4:
5:         public Altitude(double heightInMeters) {
6:                 super(heightInMeters);
7:         }
8: }
```

```
    1: package com.droidplanner.helpers.units;
    2:
    3: import java.util.Locale;
    4:
    5: public class Area {
    6:         private final String SQUARE = "\u00B2";
    7:         private double areaInSqMeters;
    8:
    9:         public Area(double areaInSqMeters) {
   10:                 this.areaInSqMeters = areaInSqMeters;
   11:         }
   12:
   13:         public double valueInSqMeters() {
   14:                 return areaInSqMeters;
   15:         }
   16:
   17:         public void set(double areaInSqMeters) {
   18:                 this.areaInSqMeters = areaInSqMeters;
   19:         }
   20:
   21:         @Override
   22:         public String toString() {
   23:                 if (areaInSqMeters > 100000) {
   24:                         return String.format(Locale.US,"%2.1f km"+SQUARE,areaInSqM
eters/1000000);
   25:                 }else if (areaInSqMeters>1) {
   26:                         return String.format(Locale.US,"%2.1f m"+SQUARE,areaInSqMe
ters);
   27:                 }else if (areaInSqMeters>0.00001) {
   28:                         return String.format(Locale.US,"%2.2f cm"+SQUARE,areaInSqM
eters*10000);
   29:                 }else{
   30:                         return areaInSqMeters + " m"+SQUARE;
   31:                 }
   32:
   33:         }
   34:
   35: }
```

```
    1: package com.droidplanner.helpers.units;
    2:
    3: import java.util.Locale;
    4:
    5: public class Length {
    6:         private double lengthInMeters;
    7:
    8:         public Length(double lengthInMeters) {
    9:                 set(lengthInMeters);
   10:         }
   11:
   12:         public double valueInMeters() {
   13:                 return lengthInMeters;
   14:         }
   15:
   16:         public void set(double lengthInMeters) {
   17:                 this.lengthInMeters = lengthInMeters;
   18:         }
   19:
   20:         @Override
   21:         public String toString() {
   22:                 if (lengthInMeters > 1000) {
   23:                         return String.format(Locale.US,"%2.1f km",lengthInMeters/1
000);
   24:                 }else if (lengthInMeters>1) {
   25:                         return String.format(Locale.US,"%2.1f m",lengthInMeters);
   26:                 }else if (lengthInMeters>0.001) {
   27:                         return String.format(Locale.US,"%2.1f m",lengthInMeters*10
00);
   28:                 }else{
   29:                         return lengthInMeters + " m";
   30:                 }
   31:         }
   32:
   33: }
```

```
    1: package com.droidplanner.MAVLink;
    2:
    3: import com.MAVLink.Messages.ardupilotmega.msg_command_long;
    4: import com.MAVLink.Messages.enums.MAV_CMD;
    5: import com.MAVLink.Messages.enums.MAV_COMPONENT;
    6: import com.droidplanner.drone.Drone;
    7:
    8: public class MavLinkArm {
    9:
   10:         public static void sendArmMessage(Drone drone, boolean arm) {
   11:                 msg_command_long msg = new msg_command_long();
   12:                 msg.target_system = 1;
   13:                 msg.target_component = (byte) MAV_COMPONENT.MAV_COMP_ID_SYSTEM_CON
TROL;
   14:
   15:                 msg.command = MAV_CMD.MAV_CMD_COMPONENT_ARM_DISARM;
   16:                 msg.param1 = arm?1:0;
   17:                 msg.param2 = 0;
   18:                 msg.param3 = 0;
   19:                 msg.param4 = 0;
   20:                 msg.param5 = 0;
   21:                 msg.param6 = 0;
   22:                 msg.param7 = 0;
   23:                 msg.confirmation = 0;
   24:                 drone.MavClient.sendMavPacket(msg.pack());
   25:         }
   26:
   27: }
```

```
 1: package com.droidplanner.MAVLink;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_command_ack;
 4: import com.MAVLink.Messages.ardupilotmega.msg_command_long;
 5: import com.MAVLink.Messages.enums.MAV_CMD;
 6: import com.MAVLink.Messages.enums.MAV_CMD_ACK;
 7: import com.droidplanner.drone.Drone;
 8:
 9: public class MavLinkCalibration {
10:
11:         public static void sendCalibrationAckMessage(int count, Drone drone) {
12:                 msg_command_ack msg = new msg_command_ack();
13:                 msg.command = (short) count;
14:                 msg.result = MAV_CMD_ACK.MAV_CMD_ACK_OK;
15:                 drone.MavClient.sendMavPacket(msg.pack());
16:         }
17:
18:         public static void sendStartCalibrationMessage(Drone drone) {
19:                 msg_command_long msg = new msg_command_long();
20:                 msg.target_system = 1;
21:                 msg.target_component = 1;
22:
23:                 msg.command = MAV_CMD.MAV_CMD_PREFLIGHT_CALIBRATION;
24:                 msg.param1 = 0;
25:                 msg.param2 = 0;
26:                 msg.param3 = 0;
27:                 msg.param4 = 0;
28:                 msg.param5 = 1;
29:                 msg.param6 = 0;
30:                 msg.param7 = 0;
31:                 msg.confirmation = 0;
32:                 drone.MavClient.sendMavPacket(msg.pack());
33:         }
34:
35: }
```

```
 1: package com.droidplanner.MAVLink;
 2:
 3: import com.MAVLink.Messages.ApmModes;
 4: import com.MAVLink.Messages.ardupilotmega.msg_mission_item;
 5: import com.MAVLink.Messages.ardupilotmega.msg_set_mode;
 6: import com.MAVLink.Messages.enums.MAV_CMD;
 7: import com.MAVLink.Messages.enums.MAV_FRAME;
 8: import com.droidplanner.drone.Drone;
 9:
10: public class MavLinkModes {
11:         public static void setGuidedMode(Drone drone, double latitude,
12:                         double longitude, double d) {
13:                 msg_mission_item msg = new msg_mission_item();
14:                 msg.seq = 0;
15:                 msg.current = 2; // TODO use guided mode enum
16:                 msg.frame = MAV_FRAME.MAV_FRAME_GLOBAL;
17:                 msg.command = MAV_CMD.MAV_CMD_NAV_WAYPOINT; //
18:                 msg.param1 = 0; // TODO use correct parameter
19:                 msg.param2 = 0; // TODO use correct parameter
20:                 msg.param3 = 0; // TODO use correct parameter
21:                 msg.param4 = 0; // TODO use correct parameter
22:                 msg.x = (float) latitude;
23:                 msg.y = (float) longitude;
24:                 msg.z = (float) d;
25:                 msg.autocontinue = 1; // TODO use correct parameter
26:                 msg.target_system = 1;
27:                 msg.target_component = 1;
28:                 drone.MavClient.sendMavPacket(msg.pack());
29:         }
30:
31:         public static void changeFlightMode(Drone drone, ApmModes mode) {
32:                 msg_set_mode msg = new msg_set_mode();
33:                 msg.target_system = 1;
34:                 msg.base_mode = 1; // TODO use meaningful constant
35:                 msg.custom_mode = mode.getNumber();
36:                 drone.MavClient.sendMavPacket(msg.pack());
37:         }
38: }
```

```java
package com.droidplanner.MAVLink;

import com.MAVLink.Messages.ApmModes;
import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.ardupilotmega.msg_attitude;
import com.MAVLink.Messages.ardupilotmega.msg_global_position_int;
import com.MAVLink.Messages.ardupilotmega.msg_gps_raw_int;
import com.MAVLink.Messages.ardupilotmega.msg_heartbeat;
import com.MAVLink.Messages.ardupilotmega.msg_mission_current;
import com.MAVLink.Messages.ardupilotmega.msg_nav_controller_output;
import com.MAVLink.Messages.ardupilotmega.msg_radio;
import com.MAVLink.Messages.ardupilotmega.msg_rc_channels_raw;
import com.MAVLink.Messages.ardupilotmega.msg_servo_output_raw;
import com.MAVLink.Messages.ardupilotmega.msg_sys_status;
import com.MAVLink.Messages.ardupilotmega.msg_vfr_hud;
import com.MAVLink.Messages.enums.MAV_MODE_FLAG;
import com.MAVLink.Messages.enums.MAV_STATE;
import com.droidplanner.drone.Drone;
import com.google.android.gms.maps.model.LatLng;

public class MavLinkMsgHandler {

        private Drone drone;

        public MavLinkMsgHandler(Drone drone) {
                this.drone = drone;
        }

        public void receiveData(MAVLinkMessage msg) {
                drone.waypointMananger.processMessage(msg);
                drone.parameters.processMessage(msg);
                drone.calibrationSetup.processMessage(msg);

                switch (msg.msgid) {
                case msg_attitude.MAVLINK_MSG_ID_ATTITUDE:
                        msg_attitude m_att = (msg_attitude) msg;
                        drone.orientation.setRollPitchYaw(m_att.roll * 180.0 / Mat
h.PI,
                                        m_att.pitch * 180.0 / Math.PI, m_att.yaw *
 180.0 / Math.PI);
                        break;
                case msg_vfr_hud.MAVLINK_MSG_ID_VFR_HUD:
                        msg_vfr_hud m_hud = (msg_vfr_hud) msg;
                        drone.setAltitudeGroundAndAirSpeeds(m_hud.alt, m_hud.groun
dspeed,
                                        m_hud.airspeed, m_hud.climb);
                        break;
                case msg_mission_current.MAVLINK_MSG_ID_MISSION_CURRENT:
                        drone.missionStats.setWpno(((msg_mission_current) msg).seq
);
                        break;
                case msg_nav_controller_output.MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPU
T:
                        msg_nav_controller_output m_nav = (msg_nav_controller_outp
ut) msg;
                        drone.setDisttowpAndSpeedAltErrors(m_nav.wp_dist, m_nav.al
t_error,
                                        m_nav.aspd_error);
                        drone.navigation.setNavPitchRollYaw(m_nav.nav_pitch,
                                        m_nav.nav_roll, m_nav.nav_bearing);
                        break;
                case msg_heartbeat.MAVLINK_MSG_ID_HEARTBEAT:
                        msg_heartbeat msg_heart = (msg_heartbeat) msg;
                        drone.type.setType(msg_heart.type);
                        processState(msg_heart);
                        ApmModes newMode;
                        newMode = ApmModes.getMode(msg_heart.custom_mode,
                                        drone.type.getType());
                        drone.state.setMode(newMode);
                        break;
                case msg_global_position_int.MAVLINK_MSG_ID_GLOBAL_POSITION_INT:
                        drone.GPS.setPosition(new LatLng(
                                        ((msg_global_position_int) msg).lat / 1E7,
                                        ((msg_global_position_int) msg).lon / 1E7)
);
                        break;
                case msg_sys_status.MAVLINK_MSG_ID_SYS_STATUS:
                        msg_sys_status m_sys = (msg_sys_status) msg;
                        drone.battery.setBatteryState(m_sys.voltage_battery / 1000
.0,
                                        m_sys.battery_remaining, m_sys.current_bat
tery / 100.0);
                        break;
                case msg_radio.MAVLINK_MSG_ID_RADIO:
                        // TODO implement link quality
                        break;
                case msg_gps_raw_int.MAVLINK_MSG_ID_GPS_RAW_INT:
                        drone.GPS.setGpsState(((msg_gps_raw_int) msg).fix_type,
                                        ((msg_gps_raw_int) msg).satellites_visible
,
                                        ((msg_gps_raw_int) msg).eph);
                        break;
                case msg_rc_channels_raw.MAVLINK_MSG_ID_RC_CHANNELS_RAW:
                        drone.RC.setRcInputValues((msg_rc_channels_raw) msg);
                        break;
                case msg_servo_output_raw.MAVLINK_MSG_ID_SERVO_OUTPUT_RAW:
                        drone.RC.setRcOutputValues((msg_servo_output_raw) msg);
                        break;
                }
        }

        public void processState(msg_heartbeat msg_heart) {
                checkArmState(msg_heart);
                checkFailsafe(msg_heart);
                checkIfIsFlying(msg_heart);
        }

        private void checkFailsafe(msg_heartbeat msg_heart) {
                boolean failsafe2 = msg_heart.system_status == (byte) MAV_STATE.MA
V_STATE_CRITICAL;
                drone.state.setFailsafe(failsafe2);
        }

        private void checkArmState(msg_heartbeat msg_heart) {
                drone.state
                                .setArmed((msg_heart.base_mode & (byte) MAV_MODE_F
LAG.MAV_MODE_FLAG_SAFETY_ARMED) == (byte) MAV_MODE_FLAG.MAV_MODE_FLAG_SAFETY_ARMED);
        }

        private void checkIfIsFlying(msg_heartbeat msg_heart) {
                switch (msg_heart.system_status) {
                case MAV_STATE.MAV_STATE_ACTIVE:
                case MAV_STATE.MAV_STATE_CRITICAL:
                        drone.state.setIsFlying(true);
                        break;
                case MAV_STATE.MAV_STATE_STANDBY:
                case MAV_STATE.MAV_STATE_CALIBRATING:
                        drone.state.setIsFlying(false);
                        break;
                }
        }
}
```

```
 1: package com.droidplanner.MAVLink;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_param_request_list;
 4: import com.MAVLink.Messages.ardupilotmega.msg_param_request_read;
 5: import com.MAVLink.Messages.ardupilotmega.msg_param_set;
 6: import com.droidplanner.drone.Drone;
 7: import com.droidplanner.parameters.Parameter;
 8:
 9: public class MavLinkParameters {
10:         public static void requestParametersList(Drone drone) {
11:                 msg_param_request_list msg = new msg_param_request_list();
12:                 msg.target_system = 1;
13:                 msg.target_component = 1;
14:                 drone.MavClient.sendMavPacket(msg.pack());
15:         }
16:
17:         public static void sendParameter(Drone drone, Parameter parameter) {
18:                 msg_param_set msg = new msg_param_set();
19:                 msg.target_system = 1;
20:                 msg.target_component = 1;
21:                 msg.setParam_Id(parameter.name);
22:                 msg.param_type = (byte) parameter.type;
23:                 msg.param_value = (float) parameter.value;
24:                 drone.MavClient.sendMavPacket(msg.pack());
25:         }
26:
27:         public static void readParameter(Drone drone, String name) {
28:                 msg_param_request_read msg = new msg_param_request_read();
29:                 msg.target_system = 1;
30:                 msg.target_component = 1;
31:                 msg.setParam_Id(name);
32:                 drone.MavClient.sendMavPacket(msg.pack());
33:         }
34: }
```

```
 1: package com.droidplanner.MAVLink;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_rc_channels_override;
 4: import com.droidplanner.drone.Drone;
 5:
 6: public class MavLinkRC {
 7:         public static void sendRcOverrideMsg(Drone drone, int[] rcOutputs) {
 8:                 msg_rc_channels_override msg = new msg_rc_channels_override();
 9:                 msg.chan1_raw = (short) rcOutputs[0];
10:                 msg.chan2_raw = (short) rcOutputs[1];
11:                 msg.chan3_raw = (short) rcOutputs[2];
12:                 msg.chan4_raw = (short) rcOutputs[3];
13:                 msg.chan5_raw = (short) rcOutputs[4];
14:                 msg.chan6_raw = (short) rcOutputs[5];
15:                 msg.chan7_raw = (short) rcOutputs[6];
16:                 msg.chan8_raw = (short) rcOutputs[7];
17:                 msg.target_system = 1;
18:                 msg.target_component = 1;
19:                 drone.MavClient.sendMavPacket(msg.pack());
20:         }
21: }
```

```
    1: package com.droidplanner.MAVLink;
    2:
    3: import android.content.SharedPreferences;
    4: import android.preference.PreferenceManager;
    5:
    6: import com.MAVLink.Messages.ardupilotmega.msg_request_data_stream;
    7: import com.MAVLink.Messages.enums.MAV_DATA_STREAM;
    8: import com.droidplanner.DroidPlannerApp;
    9: import com.droidplanner.service.MAVLinkClient;
   10:
   11: public class MavLinkStreamRates {
   12:         public static void setupStreamRatesFromPref(DroidPlannerApp droidPlannerAp
p) {
   13:                 SharedPreferences prefs = PreferenceManager
   14:                                 .getDefaultSharedPreferences(droidPlannerApp);
   15:
   16:                 int extendedStatus = Integer.parseInt(prefs.getString(
   17:                                 "pref_mavlink_stream_rate_ext_stat", "0"));
   18:                 int extra1 = Integer.parseInt(prefs.getString(
   19:                                 "pref_mavlink_stream_rate_extra1", "0"));
   20:                 int extra2 = Integer.parseInt(prefs.getString(
   21:                                 "pref_mavlink_stream_rate_extra2", "0"));
   22:                 int extra3 = Integer.parseInt(prefs.getString(
   23:                                 "pref_mavlink_stream_rate_extra3", "0"));
   24:                 int position = Integer.parseInt(prefs.getString(
   25:                                 "pref_mavlink_stream_rate_position", "0"));
   26:                 int rcChannels = Integer.parseInt(prefs.getString(
   27:                                 "pref_mavlink_stream_rate_rc_channels", "0"));
   28:                 int rawSensors = Integer.parseInt(prefs.getString(
   29:                                 "pref_mavlink_stream_rate_raw_sensors", "0"));
   30:                 int rawController = Integer.parseInt(prefs.getString(
   31:                                 "pref_mavlink_stream_rate_raw_controller", "0"));
   32:
   33:                 setupStreamRates(droidPlannerApp.drone.MavClient, extendedStatus,
   34:                                 extra1, extra2, extra3, position, rcChannels, rawS
ensors,
   35:                                 rawController);
   36:         }
   37:
   38:         public static void setupStreamRates(MAVLinkClient MAVClient,
   39:                         int extendedStatus, int extra1, int extra2, int extra3,
   40:                         int position, int rcChannels, int rawSensors, int rawContr
oler) {
   41:                 requestMavlinkDataStream(MAVClient,
   42:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_EXTENDED_STATUS, e
xtendedStatus);
   43:                 requestMavlinkDataStream(MAVClient,
   44:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA1, extra1);
   45:                 requestMavlinkDataStream(MAVClient,
   46:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA2, extra2);
   47:                 requestMavlinkDataStream(MAVClient,
   48:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA3, extra3);
   49:                 requestMavlinkDataStream(MAVClient,
   50:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_POSITION, position
);
   51:                 requestMavlinkDataStream(MAVClient,
   52:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_RAW_SENSORS, rawSe
nsors);
   53:                 requestMavlinkDataStream(MAVClient,
   54:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_RAW_CONTROLLER, ra
wControler);
   55:                 requestMavlinkDataStream(MAVClient,
   56:                                 MAV_DATA_STREAM.MAV_DATA_STREAM_RC_CHANNELS, rcCha
nnels);
   57:         }
   58:
   59:         private static void requestMavlinkDataStream(MAVLinkClient mAVClient,
   60:                         int stream_id, int rate) {
   61:                 msg_request_data_stream msg = new msg_request_data_stream();
   62:                 msg.target_system = 1;
   63:                 msg.target_component = 1;
   64:
   65:                 msg.req_message_rate = (short) rate;
   66:                 msg.req_stream_id = (byte) stream_id;
   67:
   68:                 if (rate > 0) {
   69:                         msg.start_stop = 1;
   70:                 } else {
   71:                         msg.start_stop = 0;
   72:                 }
   73:                 mAVClient.sendMavPacket(msg.pack());
   74:         }
   75: }
```

```
 1: package com.droidplanner.MAVLink;
 2:
 3: import com.MAVLink.Messages.ardupilotmega.msg_mission_ack;
 4: import com.MAVLink.Messages.ardupilotmega.msg_mission_count;
 5: import com.MAVLink.Messages.ardupilotmega.msg_mission_request;
 6: import com.MAVLink.Messages.ardupilotmega.msg_mission_request_list;
 7: import com.MAVLink.Messages.ardupilotmega.msg_mission_set_current;
 8: import com.MAVLink.Messages.enums.MAV_MISSION_RESULT;
 9: import com.droidplanner.drone.Drone;
10:
11: public class MavLinkWaypoint {
12:
13:         public static void sendAck(Drone drone) {
14:                 msg_mission_ack msg = new msg_mission_ack();
15:                 msg.target_system = 1;
16:                 msg.target_component = 1;
17:                 msg.type = MAV_MISSION_RESULT.MAV_MISSION_ACCEPTED;
18:                 drone.MavClient.sendMavPacket(msg.pack());
19:
20:         }
21:
22:         public static void requestWayPoint(Drone drone, int index) {
23:                 msg_mission_request msg = new msg_mission_request();
24:                 msg.target_system = 1;
25:                 msg.target_component = 1;
26:                 msg.seq = (short) index;
27:                 drone.MavClient.sendMavPacket(msg.pack());
28:         }
29:
30:         public static void requestWaypointsList(Drone drone) {
31:                 msg_mission_request_list msg = new msg_mission_request_list();
32:                 msg.target_system = 1;
33:                 msg.target_component = 1;
34:                 drone.MavClient.sendMavPacket(msg.pack());
35:         }
36:
37:         public static void sendWaypointCount(Drone drone, int count) {
38:                 msg_mission_count msg = new msg_mission_count();
39:                 msg.target_system = 1;
40:                 msg.target_component = 1;
41:                 msg.count = (short) count;
42:                 drone.MavClient.sendMavPacket(msg.pack());
43:         }
44:
45:         public static void sendSetCurrentWaypoint(Drone drone, short i) {
46:                 msg_mission_set_current msg = new msg_mission_set_current();
47:                 msg.target_system = 1;
48:                 msg.target_component = 1;
49:                 msg.seq = i;
50:                 drone.MavClient.sendMavPacket(msg.pack());
51:         }
52:
53: }
```

```
    1: package com.droidplanner.parameters;
    2:
    3: import java.text.DecimalFormat;
    4:
    5: import com.MAVLink.Messages.ardupilotmega.msg_param_value;
    6:
    7: public class Parameter {
    8:
    9:     public String name;
   10:         public double value;
   11:         public int type;
   12:
   13:     private final static DecimalFormat format = (DecimalFormat) DecimalFormat.getI
nstance();
   14:     static { format.applyPattern("0.###"); }
   15:
   16:
   17:         public Parameter(String name, double value, int type) {
   18:                 this.name = name;
   19:                 this.value = value;
   20:                 this.type = type;
   21:         }
   22:
   23:         public Parameter(msg_param_value m_value) {
   24:                 this(m_value.getParam_Id(), m_value.param_value, m_value.param_typ
e);
   25:         }
   26:
   27:         public Parameter(String name, Double value) {
   28:                 this(name, value, 0); // TODO Setting type to Zero may cause an er
ror
   29:         }
   30:
   31:         public Parameter(String name) {
   32:                 this(name, 0, 0); // TODO Setting type to Zero may cause an error
   33:         }
   34:
   35:         public String getValue() {
   36:                 return format.format(value);
   37:         }
   38:
   39:         public static void checkParameterName(String name) throws Exception {
   40:                 if (name.equals("SYSID_SW_MREV")) {
   41:                         throw new Exception("ExludedName");
   42:                 } else if (name.contains("WP_TOTAL")) {
   43:                         throw new Exception("ExludedName");
   44:                 } else if (name.contains("CMD_TOTAL")) {
   45:                         throw new Exception("ExludedName");
   46:                 } else if (name.contains("FENCE_TOTAL")) {
   47:                         throw new Exception("ExludedName");
   48:                 } else if (name.contains("SYS_NUM_RESETS")) {
   49:                         throw new Exception("ExludedName");
   50:                 } else if (name.contains("ARSPD_OFFSET")) {
   51:                         throw new Exception("ExludedName");
   52:                 } else if (name.contains("GND_ABS_PRESS")) {
   53:                         throw new Exception("ExludedName");
   54:                 } else if (name.contains("GND_TEMP")) {
   55:                         throw new Exception("ExludedName");
   56:                 } else if (name.contains("CMD_INDEX")) {
   57:                         throw new Exception("ExludedName");
   58:                 } else if (name.contains("LOG_LASTFILE")) {
   59:                         throw new Exception("ExludedName");
   60:                 } else if (name.contains("FORMAT_VERSION")) {
   61:                         throw new Exception("ExludedName");
   62:                 } else {
   63:                         return;
   64:                 }
   65:         }
   66:
   67:     public static DecimalFormat getFormat() {
   68:         return format;
   69:     }
   70: }
```

```java
package com.droidplanner.parameters;

import java.text.DecimalFormat;
import java.text.ParseException;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;

public class ParameterMetadata {
    public static final int RANGE_LOW = 0;
    public static final int RANGE_HIGH = 1;

    private String name;
        private String displayName;
        private String description;

        private String units;
    private String range;
    private String values;


    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDisplayName() {
        return displayName;
    }

    public void setDisplayName(String displayName) {
        this.displayName = displayName;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getUnits() {
        return units;
    }

    public void setUnits(String units) {
        this.units = units;
    }

    public String getRange() {
        return range;
    }

    public void setRange(String range) {
        this.range = range;
    }

    public String getValues() {
        return values;
    }

    public void setValues(String values) {
        this.values = values;
    }

    public boolean hasInfo() {
        return (description != null && !description.isEmpty()) || (values != null
&& !values.isEmpty());
    }

    public double[] parseRange() throws ParseException {
        final DecimalFormat format = Parameter.getFormat();

        final String[] parts = this.range.split(" ");
        if(parts.length != 2) {
            throw new IllegalArgumentException();
        }

        final double[] outRange = new double[2];
        outRange[RANGE_LOW] = format.parse(parts[RANGE_LOW]).doubleValue();
        outRange[RANGE_HIGH] = format.parse(parts[RANGE_HIGH]).doubleValue();

        return outRange;
    }

    public Map<Double, String> parseValues() throws ParseException {
        final DecimalFormat format = Parameter.getFormat();

        final Map<Double, String> outValues = new LinkedHashMap<Double, String>();
        if(values != null) {
            final String[] tparts = this.values.split(",");
            for (String tpart : tparts) {
                final String[] parts = tpart.split(":");
                if(parts.length != 2)
                    throw new IllegalArgumentException();
                outValues.put(format.parse(parts[0].trim()).doubleValue(), parts[1
].trim());
            }
        }
        return outValues;
    }
}
```

```
    1: package com.droidplanner.polygon;
    2:
    3: import java.util.List;
    4:
    5: import com.droidplanner.helpers.geoTools.GeoTools;
    6: import com.google.android.gms.maps.model.LatLng;
    7: import com.google.android.gms.maps.model.LatLngBounds;
    8:
    9: /**
   10:  *
   11:  * Object for holding boundary for a polygon
   12:  *
   13:  */
   14: public class PolyBounds {
   15:         public LatLng sw;
   16:         public LatLng ne;
   17:
   18:         public PolyBounds(List<LatLng> points) {
   19:                 LatLngBounds.Builder builder = new LatLngBounds.Builder();
   20:                 for (LatLng point : points) {
   21:                         builder.include(point);
   22:                 }
   23:                 LatLngBounds bounds = builder.build();
   24:                 sw = bounds.southwest;
   25:                 ne = bounds.northeast;
   26:         }
   27:
   28:         public double getDiag() {
   29:                 return GeoTools.latToMeters(GeoTools.getAproximatedDistance(ne, sw
));
   30:         }
   31:
   32:         public LatLng getMiddle() {
   33:                 return (new LatLng((ne.latitude + sw.latitude) / 2,
   34:                                 (ne.longitude + sw.longitude) / 2));
   35:
   36:         }
   37: }
```

```
    1: package com.droidplanner.polygon;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.droidplanner.fragments.helpers.MapPath.PathSource;
    7: import com.droidplanner.helpers.geoTools.GeoTools;
    8: import com.droidplanner.helpers.geoTools.LineLatLng;
    9: import com.droidplanner.helpers.units.Area;
   10: import com.google.android.gms.maps.model.LatLng;
   11:
   12: public class Polygon implements PathSource {
   13:
   14:         private List<PolygonPoint> points = new ArrayList<PolygonPoint>();
   15:
   16:         public void addPoints(List<LatLng> pointList) {
   17:                 for (LatLng point : pointList) {
   18:                         addPoint(point);
   19:                 }
   20:         }
   21:
   22:         public void addPoint(LatLng coord) {
   23:                 points.add(new PolygonPoint(coord));
   24:         }
   25:
   26:         public void clearPolygon() {
   27:                 points.clear();
   28:         }
   29:
   30:         public List<LatLng> getLatLngList() {
   31:                 List<LatLng> list = new ArrayList<LatLng>();
   32:                 for (PolygonPoint point : points) {
   33:                         list.add(point.coord);
   34:                 }
   35:                 return list;
   36:         }
   37:
   38:         public List<LineLatLng> getLines() {
   39:                 List<LineLatLng> list = new ArrayList<LineLatLng>();
   40:                 for (int i = 0; i < points.size(); i++) {
   41:                         int endIndex = (i==0)? points.size()-1: i-1;
   42:                         list.add(new LineLatLng(points.get(i).coord,points.get(end
Index).coord));
   43:                 }
   44:                 return list;
   45:         }
   46:
   47:         public List<PolygonPoint> getPolygonPoints() {
   48:                 return points;
   49:         }
   50:
   51:         public void movePoint(LatLng coord, int number) {
   52:                 points.get(number).coord = coord;
   53:
   54:         }
   55:
   56:         public Area getArea() {
   57:                 return GeoTools.getArea(this);
   58:         }
   59:
   60:         @Override
   61:         public List<LatLng> getPathPoints() {
   62:                 List<LatLng> path = getLatLngList();
   63:                 if (getLatLngList().size() > 2) {
   64:                         path.add(path.get(0));
   65:                 }
   66:                 return path;
   67:         }
   68:
   69:         public void checkIfValid() throws Exception {
   70:                 switch (points.size()) {
   71:                 case 0:
   72:                         throw new Exception("Draw a polygon");
   73:                 case 1:
   74:                         throw new Exception("Draw at least 2 more polygon points")
;
   75:                 case 2:
   76:                         throw new Exception("Draw at least 1 more polygon points")
;
   77:                 default:
   78:                         return;
   79:                 }
   80:
   81:         }
   82:
   83: }
```

```
 1: package com.droidplanner.polygon;
 2:
 3: import android.content.Context;
 4:
 5: import com.droidplanner.fragments.markers.MarkerManager.MarkerSource;
 6: import com.droidplanner.fragments.markers.PolygonMarker;
 7: import com.google.android.gms.maps.model.LatLng;
 8: import com.google.android.gms.maps.model.Marker;
 9: import com.google.android.gms.maps.model.MarkerOptions;
10:
11: public class PolygonPoint implements MarkerSource {
12:
13:         public LatLng coord;
14:
15:         public PolygonPoint(Double lat, Double lng) {
16:                 coord = new LatLng(lat, lng);
17:         }
18:
19:         public PolygonPoint(LatLng coord) {
20:                 this.coord = coord;
21:         }
22:
23:         @Override
24:         public MarkerOptions build(Context context) {
25:                 return PolygonMarker.build(this);
26:         }
27:
28:         @Override
29:         public void update(Marker marker, Context context) {
30:                 PolygonMarker.update(marker, this);
31:
32:         }
33: }
```

```java
package com.droidplanner.service;

import android.annotation.SuppressLint;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPacket;

// provide a common class for some ease of use functionality
public class MAVLinkClient {
        public static final int MSG_RECEIVED_DATA = 0;
        public static final int MSG_SELF_DESTRY_SERVICE = 1;

        Context parent;
        private OnMavlinkClientListner listner;
        Messenger mService = null;
        final Messenger mMessenger = new Messenger(new IncomingHandler());
        private boolean mIsBound;

        public interface OnMavlinkClientListner {
                public void notifyConnected();

                public void notifyDisconnected();

                public void notifyReceivedData(MAVLinkMessage m);

                void notifyArmed();

                void notifyDisarmed();
        }

        public MAVLinkClient(Context context, OnMavlinkClientListner listner) {
                parent = context;
                this.listner = listner;
        }

        public void init() {
                parent.bindService(new Intent(parent, MAVLinkService.class),
                                mConnection, Context.BIND_AUTO_CREATE);
                mIsBound = true;
        }

        public void close() {
                if (isConnected()) {
                        // If we have received the service, and hence registered with
                        // it, then now is the time to unregister.
                        if (mService != null) {
                                try {
                                        Message msg = Message.obtain(null,
                                                        MAVLinkService.MSG_UNREGISTER_CLIENT);
                                        msg.replyTo = mMessenger;
                                        mService.send(msg);

                                } catch (RemoteException e) {
                                        e.printStackTrace();
                                } catch (IllegalArgumentException e) {
                                        e.printStackTrace();
                                }
                                // Unbinding the service.
                                parent.unbindService(mConnection);
                                onDisconnectService();
                        }
                }
        }

        /**
         * Handler of incoming messages from service.
         */
        @SuppressLint("HandlerLeak")
        // TODO fix this error message
        class IncomingHandler extends Handler {
                @Override
                public void handleMessage(Message msg) {
                        switch (msg.what) {
                        // Received data from... somewhere
                        case MSG_RECEIVED_DATA:
                                Bundle b = msg.getData();
                                MAVLinkMessage m = (MAVLinkMessage) b.getSerializable("msg");

                                listner.notifyReceivedData(m);
                                break;
                        case MSG_SELF_DESTRY_SERVICE:
                                close();
                                break;
                        default:
                                super.handleMessage(msg);
                        }
                }
        }

        /** Defines callbacks for service binding, passed to bindService() */
        private ServiceConnection mConnection = new ServiceConnection() {

                @Override
                public void onServiceConnected(ComponentName className, IBinder service) {
                        mService = new Messenger(service);
                        try {
                                Message msg = Message.obtain(null,
                                                MAVLinkService.MSG_REGISTER_CLIENT);

                                msg.replyTo = mMessenger;
                                mService.send(msg);
                                onConnectedService();
                        } catch (RemoteException e) {
                        }
                }

                @Override
                public void onServiceDisconnected(ComponentName arg0) {
                        onDisconnectService();
                }
        };

        public void sendMavPacket(MAVLinkPacket pack) {
                Message msg = Message.obtain(null, MAVLinkService.MSG_SEND_DATA);
                Bundle data = new Bundle();
                data.putSerializable("msg", pack);
                msg.setData(data);
                try {
                        mService.send(msg);
                } catch (RemoteException e) {
                        e.printStackTrace();
```

```
130:                 } catch (NullPointerException e) {
131:                         e.printStackTrace();
132:                 }
133:
134:         }
135:
136:         private void onConnectedService() {
137:                 listner.notifyConnected();
138:         }
139:
140:         private void onDisconnectService() {
141:                 mIsBound = false;
142:                 listner.notifyDisconnected();
143:         }
144:
145:         public void queryConnectionState() {
146:                 if (mIsBound) {
147:                         listner.notifyConnected();
148:                 } else {
149:                         listner.notifyDisconnected();
150:                 }
151:
152:         }
153:
154:         public boolean isConnected() {
155:                 return mIsBound;
156:         }
157:
158:         public void toggleConnectionState() {
159:                 if (isConnected()) {
160:                         close();
161:                 } else {
162:                         init();
163:                 }
164:         }
165: }
```

```
 1: package com.droidplanner.service;
 2:
 3: import android.annotation.SuppressLint;
 4: import android.app.NotificationManager;
 5: import android.app.PendingIntent;
 6: import android.app.Service;
 7: import android.content.Context;
 8: import android.content.Intent;
 9: import android.os.Bundle;
10: import android.os.Handler;
11: import android.os.IBinder;
12: import android.os.Message;
13: import android.os.Messenger;
14: import android.os.PowerManager;
15: import android.os.PowerManager.WakeLock;
16: import android.os.RemoteException;
17: import android.preference.PreferenceManager;
18: import android.support.v4.app.NotificationCompat;
19:
20: import com.MAVLink.Messages.MAVLinkMessage;
21: import com.MAVLink.Messages.MAVLinkPacket;
22: import com.droidplanner.R;
23: import com.droidplanner.activitys.FlightActivity;
24: import com.droidplanner.connection.BluetoothConnection;
25: import com.droidplanner.connection.MAVLinkConnection;
26: import com.droidplanner.connection.MAVLinkConnection.MavLinkConnectionListner;
27: import com.droidplanner.connection.TcpConnection;
28: import com.droidplanner.connection.UdpConnection;
29: import com.droidplanner.connection.UsbConnection;
30:
31: /**
32:  * http://developer.android.com/guide/components/bound-services.html#Messenger
33:  *
34:  */
35: public class MAVLinkService extends Service implements MavLinkConnectionListner {
36:         public static final int MSG_REGISTER_CLIENT = 1;
37:         public static final int MSG_UNREGISTER_CLIENT = 2;
38:         public static final int MSG_SEND_DATA = 3;
39:
40:         private WakeLock wakeLock;
41:         private MAVLinkConnection mavConnection;
42:         Messenger msgCenter = null;
43:         final Messenger mMessenger = new Messenger(new IncomingHandler());
44:         private boolean couldNotOpenConnection = false;
45:
46:         /**
47:          * Handler of incoming messages from clients.
48:          */
49:         @SuppressLint("HandlerLeak")
50:         // TODO fix this error message
51:         class IncomingHandler extends Handler {
52:
53:                 @Override
54:                 public void handleMessage(Message msg) {
55:                         switch (msg.what) {
56:                         case MSG_REGISTER_CLIENT:
57:                                 msgCenter = msg.replyTo;
58:                                 if (couldNotOpenConnection) {
59:                                         selfDestryService();
60:                                 }
61:                                 break;
62:                         case MSG_UNREGISTER_CLIENT:
63:                                 msgCenter = null;
64:                                 break;
65:                         case MSG_SEND_DATA:
66:                                 Bundle b = msg.getData();
67:                                 MAVLinkPacket packet = (MAVLinkPacket) b.getSerial
izable("msg");
68:                                 if (mavConnection != null) {
69:                                         mavConnection.sendMavPacket(packet);
70:                                 }
71:                         default:
72:                                 super.handleMessage(msg);
73:                         }
74:                 }
75:         }
76:
77:         /**
78:          * Target we publish for clients to send messages to IncomingHandler.
79:          */
80:         @Override
81:         public IBinder onBind(Intent intent) {
82:                 return mMessenger.getBinder();
83:         }
84:
85:         private void notifyNewMessage(MAVLinkMessage m) {
86:                 try {
87:                         if (msgCenter != null) {
88:                                 Message msg = Message.obtain(null,
89:                                                 MAVLinkClient.MSG_RECEIVED_DATA);
90:                                 Bundle data = new Bundle();
91:                                 data.putSerializable("msg", m);
92:                                 msg.setData(data);
93:                                 msgCenter.send(msg);
94:                         }
95:                 } catch (RemoteException e) {
96:                         e.printStackTrace();
97:                 }
98:         }
99:
100:        @Override
101:        public void onReceiveMessage(MAVLinkMessage msg) {
102:                notifyNewMessage(msg);
103:        }
104:
105:        @Override
106:        public void onDisconnect() {
107:                couldNotOpenConnection = true;
108:                selfDestryService();
109:        }
110:
111:        private void selfDestryService() {
112:                try {
113:                        if (msgCenter != null) {
114:                                Message msg = Message.obtain(null,
115:                                                MAVLinkClient.MSG_SELF_DESTRY_SERV
ICE);
116:
117:                                msgCenter.send(msg);
118:                        }
119:                } catch (RemoteException e) {
120:                        e.printStackTrace();
121:                }
122:        }
123:
124:        @Override
125:        public void onCreate() {
126:                super.onCreate();
127:                connectMAVconnection();
128:                showNotification();
129:                aquireWakelock();
130:                updateNotification(getResources().getString(R.string.conected));
131:        }
132:
133:        @Override
```

```
  133:         public void onDestroy() {
  134:                 disconnectMAVConnection();
  135:                 dismissNotification();
  136:                 releaseWakelock();
  137:                 super.onDestroy();
  138:         }
  139:
  140:         /**
  141:          * Toggle the current state of the MAVlink connection. Starting and closin
g
  142:          * the as needed. May throw a onConnect or onDisconnect callback
  143:          */
  144:         private void connectMAVconnection() {
  145:                 String connectionType = PreferenceManager.getDefaultSharedPreferen
ces(
  146:                                 getApplicationContext()).getString("pref_connectio
n_type", "");
  147:                 if (connectionType.equals("USB")) {
  148:                         mavConnection = new UsbConnection(this);
  149:                 } else if (connectionType.equals("TCP")) {
  150:                         mavConnection = new TcpConnection(this);
  151:                 } else if (connectionType.equals("UDP")) {
  152:                         mavConnection = new UdpConnection(this);
  153:                 } else if (connectionType.equals("BLUETOOTH")) {
  154:                         mavConnection = new BluetoothConnection(this);
  155:                 } else {
  156:                         return;
  157:                 }
  158:                 mavConnection.start();
  159:         }
  160:
  161:         private void disconnectMAVConnection() {
  162:                 if (mavConnection != null) {
  163:                         mavConnection.disconnect();
  164:                         mavConnection = null;
  165:                 }
  166:         }
  167:
  168:         /**
  169:          * Show a notification while this service is running.
  170:          */
  171:         static final int StatusBarNotification = 1;
  172:
  173:         private void showNotification() {
  174:                 updateNotification(getResources().getString(R.string.disconnected)
);
  175:         }
  176:
  177:         private void updateNotification(String text) {
  178:                 NotificationCompat.Builder mBuilder = new NotificationCompat.Build
er(
  179:                                 this).setSmallIcon(R.drawable.ic_launcher)
  180:                                 .setContentTitle(getResources().getString(R.string
.app_title))
  181:                                 .setContentText(text);
  182:                 PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
  183:                                 new Intent(this, FlightActivity.class), 0);
  184:                 mBuilder.setContentIntent(contentIntent);
  185:
  186:                 NotificationManager mNotificationManager = (NotificationManager) g
etSystemService(Context.NOTIFICATION_SERVICE);
  187:                 mNotificationManager.notify(StatusBarNotification, mBuilder.build(
));
  188:         }
  189:
  190:         private void dismissNotification() {
  191:                 NotificationManager mNotificationManager = (NotificationManager) g
etSystemService(Context.NOTIFICATION_SERVICE);
  192:                 mNotificationManager.cancelAll();
  193:
  194:         }
  195:
  196:         @SuppressWarnings("deprecation")
  197:         protected void aquireWakelock() {
  198:                 if (wakeLock == null) {
  199:                         PowerManager pm = (PowerManager) getSystemService(Context.
POWER_SERVICE);
  200:                         if (PreferenceManager.getDefaultSharedPreferences(
  201:                                         getApplicationContext()).getBoolean(
  202:                                         "pref_keep_screen_bright", false)) {
  203:                                 wakeLock = pm.newWakeLock(PowerManager.SCREEN_BRIG
HT_WAKE_LOCK
  204:                                                 | PowerManager.ON_AFTER_RELEASE, "
CPU");
  205:                         } else {
  206:                                 wakeLock = pm.newWakeLock(PowerManager.SCREEN_DIM_
WAKE_LOCK,
  207:                                                 "CPU");
  208:                         }
  209:
  210:                         wakeLock.acquire();
  211:                 }
  212:         }
  213:
  214:         protected void releaseWakelock() {
  215:                 if (wakeLock != null) {
  216:                         wakeLock.release();
  217:                         wakeLock = null;
  218:                 }
  219:         }
  220: }
```

```
    1: package com.droidplanner.widgets.adapterViews;
    2:
    3: import java.util.List;
    4:
    5: import android.content.Context;
    6: import android.view.LayoutInflater;
    7: import android.view.View;
    8: import android.view.ViewGroup;
    9: import android.widget.ArrayAdapter;
   10: import android.widget.TextView;
   11:
   12: import com.droidplanner.R;
   13:
   14: public class MissionItemView extends ArrayAdapter<com.droidplanner.drone.variables
.mission.MissionItem> {
   15:
   16:         private Context context;
   17:         private List<com.droidplanner.drone.variables.mission.MissionItem> waypoin
ts;
   18:
   19:
   20:         private TextView nameView;
   21:         private TextView altitudeView;
   22:         private TextView typeView;
   23:         private TextView descView;
   24:         private TextView distanceView;
   25:
   26:
   27:
   28:         public MissionItemView(Context context, int resource, List<com.droidplanne
r.drone.variables.mission.MissionItem> list) {
   29:                 super(context, resource, list);
   30:                 this.waypoints = list;
   31:                 this.context = context;
   32:         }
   33:
   34:         public MissionItemView(Context context, int resource) {
   35:                 super(context, resource);
   36:                 this.context = context;
   37:         }
   38:
   39:         @Override
   40:         public View getView(int position, View convertView, ViewGroup parent) {
   41:                 return createRowViews(parent, position);
   42:         }
   43:
   44:         private View createRowViews(ViewGroup root, int position) {
   45:                 com.droidplanner.drone.variables.mission.MissionItem waypoint = wa
ypoints.get(position);
   46:                 View view = createLayoutFromResource();
   47:                 findViewObjects(view);
   48:                 setupViewsText(waypoint);
   49:                 return view;
   50:         }
   51:
   52:         private View createLayoutFromResource() {
   53:                 LayoutInflater inflater = (LayoutInflater) context
   54:                                 .getSystemService(Context.LAYOUT_INFLATER_SERVICE)
;
   55:                 View view = inflater.inflate(R.layout.adapterview_mission_item, nu
ll);
   56:                 return view;
   57:         }
   58:
   59:
   60:         private void findViewObjects(View view) {
   61:                 nameView = (TextView) view.findViewById(R.id.rowNameView);
   62:                 /*altitudeView = (TextView) view.findViewById(R.id.rowAltitudeView
);
   63:                 typeView = (TextView) view.findViewById(R.id.rowTypeView);
   64:                 descView = (TextView) view.findViewById(R.id.rowDescView);
   65:                 distanceView = (TextView) view.findViewById(R.id.rowDistanceView);
   66: */
   67:         }
   68:         private void setupViewsText(com.droidplanner.drone.variables.mission.Missi
onItem waypoint) {
   69:                 /*
   70:                 if (waypoint.getCmd().showOnMap()) {
   71:                         altitudeView.setText(String.format(Locale.ENGLISH, "%3.0fm
", waypoint.getHeight()));
   72:                 } else {
   73:                         altitudeView.setText("-");
   74:                 }
   75: */
   76:                 //TODO fix the numbering
   77:                 //nameView.setText(String.format("%3d", waypoint.getNumber()));
   78:
   79:
   80:         /*      typeView.setText(waypoint.getCmd().getName());
   81:                 descView.setText(setupDescription(waypoint));
   82:
   83:                 double distanceFromPrevPoint = waypoint.getDistanceFromPrevPoint()
;
   84:                 if(distanceFromPrevPoint != waypoint.UNKNOWN_DISTANCE) {
   85:                         distanceView.setText(String.format(Locale.ENGLISH, "%4.0fm
", distanceFromPrevPoint));
   86:                 }
   87:                 else {
   88:                         distanceView.setText("-");
   89:                 }
   90:                 */
   91:         }
   92: /*
   93:         private String setupDescription(waypoint waypoint) {
   94:                 String descStr = null;
   95:                 String tmpStr = null;
   96:                 float tmpVal;
   97:                 descStr = "";
   98:
   99:                 switch(waypoint.getCmd().getType())
  100:                 {
  101:                 case MAV_CMD.MAV_CMD_NAV_WAYPOINT:
  102:                         if(waypoint.missionItem.param1<=0){
  103:                                 descStr = String.format(Locale.ENGLISH, context.ge
tString(R.string.waypointDesc_Waypoint_1),
  104:                                                 waypoint.missionItem.param4);
  105:                         }
  106:                         else{
  107:                                 descStr = String.format(Locale.ENGLISH, context.ge
tString(R.string.waypointDesc_Waypoint_2),
  108:                                                 waypoint.missionItem.param1,waypoi
nt.missionItem.param4);
  109:                         }
  110:                         break;
  111:
  112:                 case MAV_CMD.MAV_CMD_NAV_LOITER_UNLIM:
  113:                         tmpVal = waypoint.missionItem.param3<0?-1*waypoint.mission
Item.param3:waypoint.missionItem.param3;
  114:                         tmpStr = waypoint.missionItem.param3<0?context.getString(R
.string.waypointDesc_CCW):context.getString(R.string.waypointDesc_CW);
  115:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_Loiter),
  116:                                         tmpVal,tmpStr,waypoint.missionItem.param4)
;
```

```
117:                         break;
118:
119:                 case MAV_CMD.MAV_CMD_NAV_LOITER_TURNS:
120:                         tmpVal = waypoint.missionItem.param3<0?-1*waypoint.mission
Item.param3:waypoint.missionItem.param3;
121:                         tmpStr = waypoint.missionItem.param3<0?context.getString(R
.string.waypointDesc_CCW):context.getString(R.string.waypointDesc_CW);
122:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_LoiterN),
123:                                         waypoint.missionItem.param1,tmpVal,tmpStr,
waypoint.missionItem.param4);
124:                         break;
125:
126:                 case MAV_CMD.MAV_CMD_NAV_LOITER_TIME:
127:                         tmpVal = waypoint.missionItem.param3<0?-1*waypoint.mission
Item.param3:waypoint.missionItem.param3;
128:                         tmpStr = waypoint.missionItem.param3<0?context.getString(R
.string.waypointDesc_CCW):context.getString(R.string.waypointDesc_CW);
129:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_LoiterT),
130:                                         waypoint.missionItem.param1,tmpVal,tmpStr,
waypoint.missionItem.param4);
131:                         break;
132:
133:                 case MAV_CMD.MAV_CMD_NAV_TAKEOFF:
134:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_Takeoff), waypoint.missionItem.param1);
135:                         break;
136:
137:                 case MAV_CMD.MAV_CMD_CONDITION_CHANGE_ALT:
138:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_CondAlt),
139:                                         waypoint.missionItem.z,waypoint.missionIte
m.param1);
140:                         break;
141:
142:                 case MAV_CMD.MAV_CMD_CONDITION_DISTANCE:
143:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_CondDist),
144:                                         waypoint.missionItem.param1);
145:                         break;
146:
147:                 case MAV_CMD.MAV_CMD_CONDITION_YAW:
148:                         tmpStr = waypoint.missionItem.param4>0?context.getString(R
.string.waypoint_yawrelative):
149:                                 context.getString(R.string.waypoint_yawabsolute);
150:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_CondYaw),
151:                                         waypoint.missionItem.param1,
152:                                         waypoint.missionItem.param2,
153:                                         waypoint.missionItem.param3>0?context.getS
tring(R.string.waypointDesc_CCW):context.getString(R.string.waypointDesc_CW),
154:                                         tmpStr);
155:                         break;
156:
157:                 case MAV_CMD.MAV_CMD_DO_SET_HOME:
158:                         descStr += context.getString(R.string.waypointDesc_SetHome
);
159:                         descStr += " ";
160:                         if(waypoint.missionItem.param1>0){
161:                                 descStr += context.getString(R.string.waypointDesc
_coordmav);
162:                         }
163:                         else {
164:                                 switch(waypoint.homeType){
165:                                 case 0:
166:                                         descStr += context.getString(R.string.wayp
ointDesc_coordwp);
167:                                         break;
168:                                 case 1:
169:                                         descStr += context.getString(R.string.wayp
ointDesc_coordgcs);
170:                                         break;
171:                                 case 2:
172:                                         descStr += context.getString(R.string.wayp
ointDesc_coordmanual);
173:                                         break;
174:                                 default:
175:                                         break;
176:                                 }
177:                                 descStr +=" ";
178:                                 descStr += String.format(Locale.ENGLISH, context.g
etString(R.string.waypointDesc_GPS),
179:                                                 waypoint.getCoord().latitude,waypo
int.getCoord().longitude);
180:                                 break;
181:                         }
182:
183:                         break;
184:
185:                 case MAV_CMD.MAV_CMD_DO_CHANGE_SPEED:
186:                         tmpStr = waypoint.missionItem.param1>0?"Ground Speed":"Air
 Speed";
187:                         descStr += String.format(Locale.ENGLISH, context.getString
(R.string.waypointDesc_SetSpeed),
188:                                         waypoint.missionItem.param2,waypoint.missi
onItem.param3, tmpStr);
189:                         break;
190:
191:                 case MAV_CMD.MAV_CMD_DO_SET_RELAY:
192:                         descStr += context.getString(R.string.waypointDesc_SetRela
y);
193:                         descStr += " ";
194:                         break;
195:
196:                 case MAV_CMD.MAV_CMD_DO_REPEAT_RELAY:
197:                         descStr += context.getString(R.string.waypointDesc_SetRepe
at);
198:                         descStr += " ";
199:                         break;
200:
201:                 case MAV_CMD.MAV_CMD_DO_JUMP:
202:                         descStr += descStr += String.format(Locale.ENGLISH, contex
t.getString(R.string.waypointDesc_SetJump),
203:                                         (int)waypoint.missionItem.param1+1,waypoin
ts.get((int)waypoint.missionItem.param1).getCmd().getName());
204:                         descStr += " ";
205:                         break;
206:                 }
207:
208:                 return descStr;
209:         }
210:          */
211: }
```

```java
package com.droidplanner.widgets.adapterViews;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.Color;
import android.graphics.Typeface;
import android.text.Editable;
import android.text.InputType;
import android.text.TextWatcher;
import android.util.AttributeSet;
import android.view.Gravity;
import android.view.View;
import android.widget.EditText;
import android.widget.TableRow;
import android.widget.TextView;

import com.droidplanner.R;
import com.droidplanner.dialogs.parameters.DialogParameterInfo;
import com.droidplanner.dialogs.parameters.DialogParameterValues;
import com.droidplanner.drone.variables.Parameters;
import com.droidplanner.parameters.Parameter;
import com.droidplanner.parameters.ParameterMetadata;

import java.text.ParseException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class ParamRow extends TableRow implements
        TextWatcher, View.OnClickListener, View.OnFocusChangeListener {
        private TextView nameView;
    private TextView displayNameView;
        private EditText valueView;
        private Parameter param;
    private ParameterMetadata metadata;

    private enum Validation { NA, INVALID, VALID }


    public ParamRow(Context context) {
                super(context);
                createRowViews(context);
        }

        public ParamRow(Context context, AttributeSet attrs) {
                super(context, attrs);
                createRowViews(context);
        }

        public void setParam(Parameter param, Parameters parameters) {
                this.param = param;
                nameView.setText(param.name);
                valueView.setText(param.getValue());

        setParamMetadata(parameters);
    }

    public void setParamMetadata(Parameters parameters) {
        metadata = parameters.getMetadata(param.name);

        String displayNameViewText = "";
        if(metadata != null) {
            // display-name (units)
            displayNameViewText = metadata.getDisplayName();
            if(metadata.getUnits() != null)
                displayNameViewText += " (" + metadata.getUnits() + ")";
        }
        displayNameView.setText(displayNameViewText);
    }

    private void createRowViews(Context context) {
        // name
        nameView = new TextView(context);
        nameView.setWidth(300);
        nameView.setOnClickListener(this);
        addView(nameView);

        // display
        displayNameView = new TextView(context);
        displayNameView.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT
, LayoutParams.MATCH_PARENT, (float) 1.0));
        displayNameView.setGravity(Gravity.CENTER_VERTICAL);
        displayNameView.setOnClickListener(this);
        addView(displayNameView);

        // value
                valueView = new EditText(context);
                valueView.setInputType(
                InputType.TYPE_CLASS_NUMBER |
                InputType.TYPE_NUMBER_FLAG_DECIMAL |
                InputType.TYPE_NUMBER_FLAG_SIGNED);
                valueView.setWidth(220);
                valueView.setGravity(Gravity.RIGHT);
        valueView.addTextChangedListener(this);
        valueView.setOnFocusChangeListener(this);
        addView(valueView);
    }

        public Parameter getParameterFromRow() {
                return (new Parameter(param.name, getParamValue(), param.type));
        }

        public double getParamValue() {
        try {
            return Parameter.getFormat().parse(valueView.getText().toString()).dou
bleValue();
        } catch (ParseException ex) {
            throw new NumberFormatException(ex.getMessage());
        }
    }

        public String getParamName() {
                return param.name;
        }

        @Override
        public void afterTextChanged(Editable s) {
        final String newValue = valueView.getText().toString();

        final int color;
        if(isValueEqualToDroneParam(newValue)) {
            color = Color.WHITE;
            valueView.setTypeface(null, Typeface.NORMAL);
        } else {
            final Validation validation = validateValue(newValue);
            if (validation == Validation.VALID) {
                color = Color.GREEN;
            } else if (validation == Validation.INVALID) {
                color = Color.RED;
            } else {
                color = Color.YELLOW;
            }
            valueView.setTypeface(null, Typeface.BOLD);
```

```
        }
        valueView.setTextColor(color);
    }

        public boolean isNewValueEqualToDroneParam() {
                return isValueEqualToDroneParam(valueView.getText().toString());
        }

    private boolean isValueEqualToDroneParam(String value) {
        return param.getValue().equals(value);
    }

    /*
     * Return TRUE if valid or unable to validate
     */
    private Validation validateValue(String value) {
        if(metadata == null) {
            return Validation.NA;

        } else if(metadata.getRange() != null) {
            return validateInRange(value);

        } else if(metadata.getValues() != null) {
            return validateInValues(value);

        } else {
            return Validation.NA;
        }
    }

    private Validation validateInRange(String value) {
        try {
            final double dval = Parameter.getFormat().parse(value).doubleValue();
            final double[] range = metadata.parseRange();
            return (dval >= range[ParameterMetadata.RANGE_LOW] && dval <= range[Pa
rameterMetadata.RANGE_HIGH]) ?
                    Validation.VALID : Validation.INVALID;
        } catch (ParseException ex) {
            return Validation.NA;
        }
    }

    private Validation validateInValues(String value) {
        try {
            final double dval = Parameter.getFormat().parse(value).doubleValue();
            final Map<Double, String> values = metadata.parseValues();
            if (values.keySet().contains(dval)) {
                return Validation.VALID;
            }
            else {
                return Validation.INVALID;
            }
        } catch (ParseException ex) {
            return Validation.NA;
        }
    }

    @Override
        public void beforeTextChanged(CharSequence s, int start, int count,
                        int after) {
        }

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count
) {
        }

    @Override
    public void onFocusChange(View view, boolean hasFocus) {
        if(!hasFocus) {
            // refresh value on leaving view - show results of rounding etc.
            valueView.setText(Parameter.getFormat().format(getParamValue()));
        }
    }

    @Override
    public void onClick(View view) {
        if(metadata == null || !metadata.hasInfo())
            return;

        final AlertDialog.Builder builder = DialogParameterInfo.build(metadata, ge
tContext());

        // add edit button if metadata supplies known values
        if(metadata.getValues() != null)
            addEditValuesButton(builder);

        builder.show();
    }

    private AlertDialog.Builder addEditValuesButton(AlertDialog.Builder builder) {
        return builder.setPositiveButton(R.string.parameter_row_edit, new DialogIn
terface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                DialogParameterValues.build(param.name, metadata, valueView.getTex
t().toString(), new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int which
) {
                        try {
                            final List<Double> values = new ArrayList<Double>(meta
data.parseValues().keySet());
                            valueView.setText(Parameter.getFormat().format(values.
get(which)));
                            dialogInterface.dismiss();
                        } catch (ParseException ex) {
                            // nop
                        }
                    }
                }, getContext()).show();
            }
        });
    }
}
```

```
 1: package com.droidplanner.widgets.FillBar;
 2:
 3: import android.content.Context;
 4: import android.graphics.Canvas;
 5: import android.graphics.Color;
 6: import android.graphics.Paint;
 7: import android.graphics.Paint.Style;
 8: import android.graphics.Path;
 9: import android.util.AttributeSet;
10: import android.view.View;
11:
12: public class FillBar extends View {
13:
14:         private Paint paintOutline;
15:         private Paint paintFill;
16:         private Path outlinePath = new Path();
17:         private Path fillPath = new Path();
18:         private int height;
19:         private int width;
20:         private float percentage = 0.5f;
21:
22:
23:         public FillBar(Context context, AttributeSet attrs) {
24:                 super(context, attrs);
25:                 initialize();
26:         }
27:
28:         private void initialize() {
29:
30:                 paintOutline = new Paint();
31:                 paintOutline.setAntiAlias(false);
32:                 paintOutline.setStyle(Style.STROKE);
33:                 paintOutline.setStrokeWidth(3);
34:                 paintOutline.setColor(Color.parseColor("#E0E0E0"));
35:
36:                 paintFill = new Paint(paintOutline);
37:                 paintFill.setStyle(Style.FILL);
38:         }
39:
40:         @Override
41:         protected void onSizeChanged(int w, int h, int oldw, int oldh) {
42:                 super.onSizeChanged(w, h, oldw, oldh);
43:                 width = w - 1;
44:                 height = h - 1;
45:         }
46:
47:         @Override
48:         protected void onDraw(Canvas canvas) {
49:                 super.onDraw(canvas);
50:
51:                 // Yaw Arrow
52:                 outlinePath.reset();
53:                 outlinePath.moveTo(0, 0);
54:                 outlinePath.lineTo(0, height);
55:                 outlinePath.lineTo(width, height);
56:                 outlinePath.lineTo(width, 0);
57:                 outlinePath.lineTo(0, 0);
58:                 canvas.drawPath(outlinePath, paintOutline);
59:
60:                 float fillHeight = height * (1 - percentage);
61:                 fillPath.reset();
62:                 fillPath.moveTo(0, fillHeight);
63:                 fillPath.lineTo(0, height);
64:                 fillPath.lineTo(width, height);
65:                 fillPath.lineTo(width, fillHeight);
66:                 fillPath.lineTo(0, fillHeight);
67:                 canvas.drawPath(fillPath, paintFill);
68:         }
69:
70:         public float getPercentage() {
71:                 return percentage;
72:         }
73:
74:         public void setPercentage(float percentage) {
75:                 this.percentage = percentage;
76:                 invalidate();
77:         }
78: }
```

```
 1: package com.droidplanner.widgets.FillBar;
 2:
 3: import android.content.Context;
 4: import android.util.AttributeSet;
 5: import android.widget.LinearLayout;
 6: import android.widget.TextView;
 7:
 8: import com.droidplanner.R;
 9:
10: public class FillBarWithText extends LinearLayout {
11:
12:         private int max;
13:         private int min;
14:         private TextView title;
15:         private TextView value;
16:         private FillBar bar;
17:
18:         public FillBarWithText(Context context, AttributeSet attrs) {
19:                 super(context, attrs);
20:
21:                 setOrientation(VERTICAL);
22:
23:                 inflate(context, R.layout.subview_fillbar_with_text, this);
24:
25:                 title = (TextView) findViewById(R.id.textViewBarTitle);
26:                 value = (TextView) findViewById(R.id.TextViewBarValue);
27:                 bar = (FillBar) findViewById(R.id.fillBarSubview);
28:         }
29:
30:         public void setup(String title, int max, int min) {
31:                 this.max = max;
32:                 this.min = min;
33:                 this.title.setText(title);
34:         }
35:
36:         public void setValue(int value) {
37:                 this.value.setText(Integer.toString(value));
38:                 this.bar.setPercentage((value - min) / ((float)(max - min)));
39:         }
40:
41: }
```

```java
package com.droidplanner.widgets.graph;

import java.util.ArrayList;
import java.util.List;

import android.content.Context;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import com.droidplanner.widgets.graph.ChartScale.OnScaleListner;
import com.droidplanner.widgets.helpers.RenderThread;
import com.droidplanner.widgets.helpers.RenderThread.canvasPainter;

/*
 * Widget for a Chart Originally copied from http://code.google.com/p/copter-gcs/
 */
public class Chart extends SurfaceView implements SurfaceHolder.Callback,
                canvasPainter, OnScaleListner {
        private RenderThread renderer;
        protected int width;
        protected int height;

        public ChartColorsStack colors = new ChartColorsStack();
        protected ChartScale scale;
        private ChartGrid grid = new ChartGrid();
        public List<ChartSeries> series = new ArrayList<ChartSeries>();
        private ChartDataRender dataRender = new ChartDataRender();

        public Chart(Context context, AttributeSet attributeSet) {
                super(context, attributeSet);
                getHolder().addCallback(this);

                scale = new ChartScale(context, this);
        }

        @Override
        public void onDraw(Canvas canvas) {
                grid.drawGrid(this, canvas);
                for (ChartSeries serie : series) {
                        dataRender.drawSeries(this, canvas, serie);
                }
        }

        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int width,
                        int height) {
                this.width = width;
                this.height = height;
        }

        @Override
        public void surfaceCreated(SurfaceHolder holder) {
                renderer = new RenderThread(getHolder(), this);
                if (!renderer.isRunning()) {
                        renderer.setRunning(true);
                        renderer.start();
                }
        }

        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
                boolean retry = true;
                renderer.setRunning(false);
                while (retry) {
                        try {
                                renderer.join();
                                renderer = null;
                                retry = false;
                        } catch (InterruptedException e) {
                                // we will try it again and again...
                        }
                }
        }

        @Override
        public boolean onTouchEvent(MotionEvent ev) {
                // Let the ScaleGestureDetector inspect all events.
                super.onTouchEvent(ev);
                scale.scaleDetector.onTouchEvent(ev);
                return true;
        }

        public void update() {
                if (renderer != null)
                        renderer.setDirty();
        }

        @Override
        public void onScaleListner() {
                update();
        }
}
```

```
 1: package com.droidplanner.widgets.graph;
 2:
 3: import android.content.Context;
 4: import android.graphics.Color;
 5: import android.util.AttributeSet;
 6: import android.view.Gravity;
 7: import android.widget.CheckBox;
 8: import android.widget.CompoundButton;
 9: import android.widget.CompoundButton.OnCheckedChangeListener;
10:
11: public class ChartCheckBox extends CheckBox implements OnCheckedChangeListener {
12:         public ChartCheckBox(Context context, AttributeSet attrs) {
13:                 super(context, attrs);
14:         }
15:
16:         private Chart chart;
17:
18:         public ChartCheckBox(Context context, String label, Chart chart) {
19:                 super(context);
20:                 ChartSeries serie = new ChartSeries(800);
21:                 this.chart = chart;
22:                 this.chart.series.add(serie);
23:                 setText(label);
24:                 setChecked(serie.isActive());
25:                 setGravity(Gravity.LEFT);
26:                 setTag(serie);
27:                 setOnCheckedChangeListener(this);
28:         }
29:
30:         @Override
31:         public void onCheckedChanged(CompoundButton checkBox, boolean isChecked) {
32:                 ChartSeries serie = (ChartSeries) getTag();
33:                 if (isChecked) {
34:                         serie.enable();
35:                         Integer color = chart.colors.retriveColor();
36:                         serie.setColor(color);
37:                         setTextColor(color);
38:                 } else {
39:                         serie.disable();
40:                         chart.colors.depositColor(serie.getColor());
41:                         setTextColor(Color.WHITE);
42:                 }
43:                 chart.update();
44:         }
45: }
```

```
    1: package com.droidplanner.widgets.graph;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import android.widget.LinearLayout;
    7:
    8: public class ChartCheckBoxList {
    9:         private List<ChartCheckBox> checkBoxList = new ArrayList<ChartCheckBox>();
   10:
   11:         public void populateView(LinearLayout view, String[] labels, Chart chart)
{
   12:                 for (String label : labels) {
   13:                         ChartCheckBox checkBox = new ChartCheckBox(view.getContext
(),
   14:                                         label, chart);
   15:                         checkBoxList.add(checkBox);
   16:                         view.addView(checkBox);
   17:                 }
   18:         }
   19:
   20:         public void updateCheckBox(String label, double value) {
   21:                 for (ChartCheckBox box : checkBoxList) {
   22:                         if (box.getText().equals(label)) {
   23:                                 ((ChartSeries) box.getTag()).newData(value);
   24:                         }
   25:                 }
   26:         }
   27:
   28: }
```

```
 1: package com.droidplanner.widgets.graph;
 2:
 3: import java.util.ArrayList;
 4: import java.util.List;
 5:
 6: import android.graphics.Color;
 7:
 8: public class ChartColorsStack {
 9:         List<Integer> avaliableColors = new ArrayList<Integer>();
10:
11:         public ChartColorsStack() {
12:                 avaliableColors.add(Color.RED);
13:                 avaliableColors.add(Color.BLUE);
14:                 avaliableColors.add(Color.GREEN);
15:                 avaliableColors.add(Color.YELLOW);
16:                 avaliableColors.add(Color.MAGENTA);
17:                 avaliableColors.add(Color.CYAN);
18:         }
19:
20:         public Integer retriveColor() {
21:                 if (avaliableColors.size() > 0) {
22:                         Integer color = avaliableColors.get(0);
23:                         avaliableColors.remove(0);
24:                         return color;
25:                 } else {
26:                         return Color.WHITE;
27:                 }
28:         }
29:
30:         public void depositColor(Integer color) {
31:                 avaliableColors.add(0, color);
32:         }
33: }
```

```
    1: package com.droidplanner.widgets.graph;
    2:
    3: import android.graphics.Canvas;
    4:
    5: public class ChartDataRender {
    6:         private int numPtsToDraw = 100;
    7:
    8:         protected void drawSeries(Chart chart, Canvas canvas, ChartSeries serie) {
    9:                 // scale the data to +- 500
   10:                 // target 0-height
   11:                 // so D in the range +-500
   12:                 // (D + 500) / 1000 * height
   13:
   14:                 float delta = (float) chart.width / (float) numPtsToDraw;
   15:
   16:                 if (serie.isActive()) {
   17:
   18:                         int start = (serie.newestData - numPtsToDraw + serie.data.
length)
   19:                                         % serie.data.length;
   20:                         int pos = 0;
   21:                         for (int i = start; i < start + numPtsToDraw; i++) {
   22:
   23:                                 double y_i = -serie.data[i % serie.data.length];
   24:                                 y_i = (y_i + chart.scale.getRange())
   25:                                                 / (2 * chart.scale.getRange()) * c
hart.height;
   26:
   27:                                 double y_i1 = -serie.data[(i + 1) % serie.data.len
gth];
   28:                                 y_i1 = (y_i1 + chart.scale.getRange())
   29:                                                 / (2 * chart.scale.getRange()) * c
hart.height;
   30:
   31:                                 canvas.drawLine((float) pos * delta, (float) y_i,
   32:                                                 (float) (pos + 1) * delta, (float)
 y_i1,
   33:                                                 serie.getPaint());
   34:                                 pos++;
   35:                         }
   36:                 }
   37:         }
   38:
   39:         protected void setDrawRate(Chart chart, int p) {
   40:                 if (p > 0)
   41:                         numPtsToDraw = chart.width / p;
   42:         }
   43: }
```

```
    1: package com.droidplanner.widgets.graph;
    2:
    3: import android.graphics.Canvas;
    4: import android.graphics.Color;
    5: import android.graphics.Paint;
    6:
    7: public class ChartGrid {
    8:         public Paint grid_paint = new Paint();
    9:         public Paint grid_paint_center_line = new Paint();
   10:
   11:         public ChartGrid() {
   12:                 grid_paint.setColor(Color.rgb(100, 100, 100));
   13:                 grid_paint_center_line.setColor(Color.rgb(100, 100, 100));
   14:                 grid_paint_center_line.setStrokeWidth(5f);
   15:         }
   16:
   17:         void drawGrid(Chart chart, Canvas canvas) {
   18:                 // clear screen
   19:                 canvas.drawColor(Color.rgb(20, 20, 20));
   20:
   21:                 for (int vertical = 1; vertical < 10; vertical++) {
   22:                         canvas.drawLine(vertical * (chart.width / 10) + 1, 1, vert
ical
   23:                                         * (chart.width / 10) + 1, chart.height + 1
, grid_paint);
   24:
   25:                 }
   26:
   27:                 for (int horizontal = 1; horizontal < 10; horizontal++) {
   28:                         Paint paint;
   29:                         if (horizontal == 5) {
   30:                                 paint = grid_paint_center_line;
   31:                         } else {
   32:                                 paint = grid_paint;
   33:                         }
   34:                         canvas.drawLine(1, horizontal * (chart.height / 10) + 1,
   35:                                         chart.width + 1, horizontal * (chart.heigh
t / 10) + 1,
   36:                                         paint);
   37:
   38:                 }
   39:         }
   40: }
```

```
 1: package com.droidplanner.widgets.graph;
 2:
 3: import android.content.Context;
 4: import android.view.ScaleGestureDetector;
 5:
 6: public class ChartScale {
 7:         public interface OnScaleListner {
 8:                 public void onScaleListner();
 9:         }
10:
11:         private OnScaleListner listner;
12:
13:         // range values to display
14:         private double range = 45;
15:         // minimal range
16:         private double min = 10;
17:         // maximal range
18:         private double max = 45;
19:
20:         protected ScaleGestureDetector scaleDetector;
21:
22:         public ChartScale(Context context, OnScaleListner listner) {
23:                 scaleDetector = new ScaleGestureDetector(context,
24:                                 new ChartScaleListener());
25:                 this.listner = listner;
26:         }
27:
28:         public double getRange() {
29:                 return range;
30:         }
31:
32:         class ChartScaleListener extends
33:                         ScaleGestureDetector.SimpleOnScaleGestureListener {
34:
35:                 @Override
36:                 public boolean onScale(ScaleGestureDetector detector) {
37:                         range /= detector.getScaleFactor();
38:
39:                         range = Math.max(min, Math.min(range, max));
40:                         listner.onScaleListner();
41:                         return true;
42:                 }
43:         }
44: }
```

```
 1: package com.droidplanner.widgets.graph;
 2:
 3: import android.graphics.Paint;
 4:
 5: public class ChartSeries {
 6:
 7:         private boolean enabled = false;
 8:         public double[] data;
 9:         public int newestData = 0;
10:         private Paint paint = new Paint();
11:
12:         public ChartSeries(int bufferSize) {
13:                 this.data = new double[bufferSize];
14:         }
15:
16:         public void newData(double d) {
17:                 if (data.length > 0) {
18:                         newestData = (newestData + 1) % data.length;
19:                         data[newestData] = d;
20:                 }
21:         }
22:
23:         public Paint getPaint() {
24:                 return paint;
25:         }
26:
27:         public void setColor(int color) {
28:                 paint.setColor(color);
29:         }
30:
31:         public int getColor() {
32:                 return paint.getColor();
33:         }
34:
35:         public void enable() {
36:                 enabled = true;
37:         }
38:
39:         public void disable() {
40:                 enabled = false;
41:         }
42:
43:         public boolean isActive() {
44:                 return enabled;
45:         }
46:
47: }
```

```
    1: package com.droidplanner.widgets.helpers;
    2:
    3: import android.annotation.SuppressLint;
    4: import android.graphics.Canvas;
    5: import android.view.SurfaceHolder;
    6:
    7: public class RenderThread extends Thread {
    8:         public interface canvasPainter {
    9:                 public void onDraw(Canvas c);
   10:         }
   11:
   12:         private SurfaceHolder _surfaceHolder;
   13:         private canvasPainter painter;
   14:         private volatile boolean running = false;
   15:         private Object dirty = new Object();
   16:
   17:         public RenderThread(SurfaceHolder surfaceHolder, canvasPainter painter) {
   18:                 _surfaceHolder = surfaceHolder;
   19:                 this.painter = painter;
   20:         }
   21:
   22:         public boolean isRunning() {
   23:                 return running;
   24:
   25:         }
   26:
   27:         public void setRunning(boolean run) {
   28:                 running = run;
   29:                 setDirty();
   30:         }
   31:
   32:         /** We may need to redraw */
   33:         public void setDirty() {
   34:                 synchronized (dirty) {
   35:                         dirty.notify();
   36:                 }
   37:         }
   38:
   39:         @SuppressLint("WrongCall")
   40:         // TODO fix error
   41:         @Override
   42:         public void run() {
   43:                 Canvas c;
   44:                 while (running) {
   45:                         synchronized (dirty) {
   46:                                 c = null;
   47:                                 try {
   48:                                         c = _surfaceHolder.lockCanvas(null);
   49:                                         synchronized (_surfaceHolder) {
   50:                                                 if (c != null) {
   51:                                                         painter.onDraw(c);
   52:                                                 }
   53:                                         }
   54:                                 } finally {
   55:                                         // do this in a finally so that if an exce
ption is
   56:                                         // thrown
   57:                                         // during the above, we don't leave the Su
rface in an
   58:                                         // inconsistent state
   59:                                         if (c != null) {
   60:                                                 _surfaceHolder.unlockCanvasAndPost
(c);
   61:                                         }
   62:                                 }
   63:
   64:                                 // We do this wait at the _end_ to ensure we alway
s draw at
   65:                                 // least one frame of
   66:                                 // HUD data
   67:                                 try {
   68:                                         // Log.d("HUD", "Waiting for change");
   69:                                         dirty.wait(); // TODO - not quite ready
   70:                                         // Log.d("HUD", "Handling change");
   71:                                 } catch (InterruptedException e) {
   72:                                         // We will try again and again
   73:                                 }
   74:                         }
   75:                 }
   76:         }
   77: }
```

```java
/**
 * Copied from https://code.google.com/p/mobile-anarchy-widgets/
 */
package com.droidplanner.widgets.joystick;

public interface JoystickClickedListener {
        public void OnClicked();

        public void OnReleased();
}
```

```java
/**
 * Copied from https://code.google.com/p/mobile-anarchy-widgets/
 */
package com.droidplanner.widgets.joystick;

public interface JoystickMovedListener {
        public void OnMoved(double pan, double tilt);

        public void OnReleased();

        public void OnReturnedToCenter();
}
```

```java
/**
 * Copied from https://code.google.com/p/mobile-anarchy-widgets/
 */
package com.droidplanner.widgets.joystick;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.util.Log;
import android.view.HapticFeedbackConstants;
import android.view.MotionEvent;
import android.view.View;

public class JoystickView extends View {
        public static final int INVALID_POINTER_ID = -1;
        public String TAG = "JoystickView";

        private static final double HAPTIC_FEEDBACK_ZONE = 0.05;
        private int handleRadius = 20;
        private int movementRadius = handleRadius * 4;

        private boolean yAxisInverted = false;
        private boolean xAxisInverted = false;
        private boolean yAxisAutoReturnToCenter = true;
        private boolean xAxisAutoReturnToCenter = true;
        private boolean autoReturnToCenter = true;

        private Paint bgHandlePaint;
        private Paint handlePaint;

        private JoystickMovedListener moveListener;

        // Last touch point in view coordinates
        private int pointerId = INVALID_POINTER_ID;
        private float touchX, touchY;

        // Cartesian coordinates of last touch point - joystick center is (0,0)
        private double cartX, cartY;

        // User coordinates of last touch point
        private double userX, userY;
        private double userXold, userYold;

        private float firstTouchX,firstTouchY;
        private double releaseX = 0;
        private double releaseY = 0;

        private boolean handleVisible = false;

        public JoystickView(Context context) {
                super(context);
                initJoystickView();
        }

        public JoystickView(Context context, AttributeSet attrs) {
                super(context, attrs);
                initJoystickView();
        }

        public JoystickView(Context context, AttributeSet attrs, int defStyle) {
                super(context, attrs, defStyle);
                initJoystickView();
        }

        private void initJoystickView() {
                setFocusable(true);
                setHapticFeedbackEnabled(true);

                handlePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
                handlePaint.setColor(Color.BLACK);
                handlePaint.setStrokeWidth(1);
                handlePaint.setStyle(Paint.Style.FILL_AND_STROKE);

                bgHandlePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
                bgHandlePaint.setColor(Color.BLUE);
                bgHandlePaint.setStrokeWidth(1);
                bgHandlePaint.setStyle(Paint.Style.FILL_AND_STROKE);
        }

        public void setAutoReturnToCenter(boolean autoReturnToCenter) {
                this.autoReturnToCenter = autoReturnToCenter;
        }

        public boolean isAutoReturnToCenter() {
                return autoReturnToCenter;
        }

        public boolean isXAxisInverted() {
                return xAxisInverted;
        }

        public boolean isYAxisInverted() {
                return yAxisInverted;
        }

        public void setXAxisInverted(boolean xAxisInverted) {
                this.xAxisInverted = xAxisInverted;
        }

        public void setYAxisInverted(boolean yAxisInverted) {
                this.yAxisInverted = yAxisInverted;
        }

        public void setOnJostickMovedListener(JoystickMovedListener listener) {
                this.moveListener = listener;
        }

        @Override
        protected void onDraw(Canvas canvas) {
                canvas.save();

                // Draw the handle
                if (handleVisible) {
                        canvas.drawCircle(firstTouchX, firstTouchY, movementRadius
,
                                        bgHandlePaint);
                        canvas.drawCircle(firstTouchX, firstTouchY, handleRadius,
                                        handlePaint);
                }
                canvas.restore();
        }


        @Override
        public boolean onTouchEvent(MotionEvent ev) {
                int pointerIndex;
                int pointerId;
                final int action = ev.getAction();
                switch (action & MotionEvent.ACTION_MASK) {
                case MotionEvent.ACTION_MOVE:
                        return processMove(ev);
                case MotionEvent.ACTION_CANCEL:
```

```
134:                 case MotionEvent.ACTION_UP:
135:                         if (isPointerValid()) {
136:                                 return processRelease();
137:                         }
138:                         break;
139:                 case MotionEvent.ACTION_POINTER_UP:
140:                         pointerIndex = (action & MotionEvent.ACTION_POINTER_INDEX_
MASK) >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
141:                         pointerId = ev.getPointerId(pointerIndex);
142:                         if (pointerId == this.pointerId) {
143:                                 return processRelease();
144:                         }
145:                         break;
146:                 case MotionEvent.ACTION_DOWN:
147:                         if (!isPointerValid()) {
148:                                 this.pointerId = ev.getPointerId(0);
149:                                 processFirstTouch(ev);
150:                                 return true;
151:                         }
152:                         break;
153:                 case MotionEvent.ACTION_POINTER_DOWN:
154:                         pointerIndex = (action & MotionEvent.ACTION_POINTER_INDEX_
MASK) >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
155:                         pointerId = ev.getPointerId(pointerIndex);
156:                         if (pointerId == INVALID_POINTER_ID) {
157:                                 this.pointerId = pointerId;
158:                                 processFirstTouch(ev);
159:                                 return true;
160:                         }
161:                         break;
162:                 }
163:                 return false;
164:         }
165:
166:         private boolean processRelease() {
167:                 this.pointerId = INVALID_POINTER_ID;
168:                 handleVisible = false;
169:                 invalidate();
170:                 if (moveListener != null) {
171:                         releaseX = xAxisAutoReturnToCenter ? 0 : userX;
172:                         releaseY = yAxisAutoReturnToCenter ? 0 : userY;
173:                         moveListener.OnMoved(releaseX, releaseY);
174:                 }
175:                 return true;
176:         }
177:
178:         private void processFirstTouch(MotionEvent ev) {
179:                 firstTouchX = ev.getX();
180:                 firstTouchY = ev.getY();
181:                 touchX = 0;
182:                 touchY = 0;
183:                 handleVisible = true;
184:                 invalidate();
185:         }
186:
187:         private boolean isPointerValid() {
188:                 return pointerId != INVALID_POINTER_ID;
189:         }
190:
191:         private boolean processMove(MotionEvent ev) {
192:                 if (isPointerValid()) {
193:                         final int pointerIndex = ev.findPointerIndex(pointerId);
194:
195:                         // Translate touch position to center of view
196:                         float x = ev.getX(pointerIndex);
197:                         float y = ev.getY(pointerIndex);
198:
199:                         touchX = x - firstTouchX;
200:                         touchY = y - firstTouchY;
201:
202:                         reportOnMoved();
203:                         return true;
204:                 }
205:                 return false;
206:         }
207:
208:         private void reportOnMoved() {
209:                 calcUserCoordinates();
210:                 constrainBox();
211:
212:                 hapticFeedback();
213:
214:                 if (moveListener != null) {
215:                         moveListener.OnMoved(userX, userY);
216:                 }
217:         }
218:
219:         private void hapticFeedback() {
220:                 if (hasEnteredHapticFeedbackZone(userX, userXold)) {
221:                         performHapticFeedback(HapticFeedbackConstants.VIRTUAL_KEY)
;
222:                         Log.d(TAG, "XonCenter");
223:                 }
224:                 if (hasEnteredHapticFeedbackZone(userY, userYold)) {
225:                         performHapticFeedback(HapticFeedbackConstants.VIRTUAL_KEY)
;
226:                         Log.d(TAG, "YonCenter");
227:                 }
228:
229:                 userXold = userX;
230:                 userYold = userY;
231:         }
232:
233:         private boolean hasEnteredHapticFeedbackZone(double value, double oldValue
) {
234:                 return isInHapticFeedbackZone(value)&(!isInHapticFeedbackZone(oldV
alue));
235:         }
236:
237:         private boolean isInHapticFeedbackZone(double value) {
238:                 return Math.abs(value) < HAPTIC_FEEDBACK_ZONE;
239:         }
240:
241:         private void calcUserCoordinates() {
242:                 // First convert to cartesian coordinates
243:                 cartX = (touchX / movementRadius);
244:                 cartY = (touchY / movementRadius);
245:
246:                 // Invert axis if requested
247:                 if (!xAxisInverted)
248:                         cartX *= -1;
249:                 if (!yAxisInverted)
250:                         cartY *= -1;
251:
252:                 userX = cartX + (xAxisAutoReturnToCenter ? 0 : releaseX);
253:                 userY = cartY + (yAxisAutoReturnToCenter ? 0 : releaseY);
254:
255:         }
256:
257:         // Constrain touch within a box
258:         private void constrainBox() {
259:                 userX = Math.max(Math.min(userX, 1), -1);
260:                 userY = Math.max(Math.min(userY, 1), -1);
261:         }
```

```
262:
263:         public void setAxisAutoReturnToCenter(boolean yAxisAutoReturnToCenter,
264:                         boolean xAxisAutoReturnToCenter) {
265:                 this.yAxisAutoReturnToCenter = yAxisAutoReturnToCenter;
266:                 this.xAxisAutoReturnToCenter = xAxisAutoReturnToCenter;
267:         }
268: }
```

```java
package com.droidplanner.widgets.newHUD;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.Path;
import android.graphics.RectF;
import android.graphics.Shader.TileMode;
import android.util.AttributeSet;
import android.view.View;

public class newHUD extends View {

        private static final float INTERNAL_RADIUS = 0.95f;
        private static final float YAW_ARROW_SIZE = 1.2f;
        private static final float YAW_ARROW_ANGLE = 4.5f;
        private static final float PITCH_TICK_LINE_LENGTH = 0.4f;
        private static final int PITCH_RANGE = 45;
        private static final int PITCH_TICK_SPACING = 15;
        private static final int PITCH_TICK_PADDING = 2;
        private static final float PLANE_BODY_SIZE = 0.2f;
        private static final float PLANE_WING_WIDTH = 0.05f;
        private static final float PLANE_SIZE = 0.8f;

        private float halfWidth;
        private float halfHeight;
        private float radiusExternal;
        private float radiusInternal;
        private RectF internalBounds;

        private Paint yawPaint;
        private Paint skyPaint;
        private Paint groundPaint;
        private Paint planePaint;

        private Path yawPath = new Path();
        private Path groundPath = new Path();
        private Path planePath = new Path();

        private float yaw, roll, pitch;
        private Paint tickPaint;

        public newHUD(Context context, AttributeSet attrs) {
                super(context, attrs);
                initialize();
                setAttitude(-30, 20, -45);
        }

        private void initialize() {

                Paint fillPaint = new Paint();
                fillPaint.setAntiAlias(true);
                fillPaint.setStyle(Style.FILL);

                yawPaint = new Paint(fillPaint);
                yawPaint.setColor(Color.WHITE);

                skyPaint = new Paint(fillPaint);

                groundPaint = new Paint(fillPaint);
                groundPaint.setColor(Color.parseColor("#723700"));

                planePaint = new Paint(fillPaint);
                planePaint.setColor(Color.WHITE);

                tickPaint = new Paint(fillPaint);
                tickPaint.setColor(Color.WHITE);
                tickPaint.setStrokeWidth(2);
        }

        @Override
        protected void onSizeChanged(int w, int h, int oldw, int oldh) {
                super.onSizeChanged(w, h, oldw, oldh);
                halfHeight = h / 2f;
                halfWidth = w / 2f;
                radiusExternal = Math.min(halfHeight, halfWidth) / YAW_ARROW_SIZE;
                radiusInternal = radiusExternal * INTERNAL_RADIUS;
                internalBounds = new RectF(-radiusInternal, -radiusInternal,
                                radiusInternal, radiusInternal);
                buildPlanePath();
                skyPaint.setShader(new LinearGradient(0, -radiusInternal, 0,
                                radiusInternal, Color.parseColor("#004444"), Color
                                                .parseColor("#00FFFF"), TileMode.C
LAMP));

        }

        private void buildPlanePath() {
                planePath.reset();
                planePath.moveTo(0, radiusInternal * PLANE_SIZE * PLANE_WING_WIDTH
);
                planePath.lineTo(radiusInternal * PLANE_SIZE, 0);
                planePath.lineTo(0, -radiusInternal * PLANE_SIZE * PLANE_WING_WIDT
H);
                planePath.lineTo(-radiusInternal * PLANE_SIZE, 0);
                planePath.lineTo(0, radiusInternal * PLANE_SIZE * PLANE_WING_WIDTH
);
                planePath.moveTo(radiusInternal * PLANE_SIZE * PLANE_WING_WIDTH, 0
);
                planePath.lineTo(0, -radiusInternal * PLANE_SIZE / 2);
                planePath.lineTo(-radiusInternal * PLANE_SIZE * PLANE_WING_WIDTH,
0);

        }

        @Override
        protected void onDraw(Canvas canvas) {
                super.onDraw(canvas);
                canvas.translate(halfWidth, halfHeight);
                drawYaw(canvas);
                drawSkyAndGround(canvas);
                drawPitchTicks(canvas);
                drawPlane(canvas);
        }

        private void drawYaw(Canvas canvas) {
                // Fill the background
                canvas.drawCircle(0, 0, radiusExternal, yawPaint);

                // Yaw Arrow
                float mathYaw = (float) Math.toRadians(180 - yaw);
                yawPath.reset();
                yawPath.moveTo(0, 0);
                radialLineTo(yawPath, mathYaw + YAW_ARROW_ANGLE, radiusExternal);
                radialLineTo(yawPath, mathYaw, radiusExternal * YAW_ARROW_SIZE);
                radialLineTo(yawPath, mathYaw - YAW_ARROW_ANGLE, radiusExternal);
                canvas.drawPath(yawPath, yawPaint);
        }

        private void radialLineTo(Path path, float angle, float radius) {
                path.lineTo((float) Math.sin(angle) * radius, (float) Math.cos(ang
```

```
le)
  129:                                * radius);
  130:         }
  131:
  132:         private void drawSkyAndGround(Canvas canvas) {
  133:                 // Fill with the sky
  134:                 canvas.drawCircle(0, 0, radiusInternal, skyPaint);
  135:
  136:                 // Overlay the ground
  137:                 groundPath.reset();
  138:                 float pitchProjection = (float) Math.toDegrees(Math.acos(pitch
  139:                                 / PITCH_RANGE));
  140:                 groundPath.addArc(internalBounds, 90 - pitchProjection - roll,
  141:                                 pitchProjection * 2);
  142:                 canvas.drawPath(groundPath, groundPaint);
  143:
  144:         }
  145:
  146:         private void drawPitchTicks(Canvas canvas) {
  147:                 float lineX = (float) (Math.cos(Math.toRadians(-roll)) * radiusInt
ernal)
  148:                                 * PITCH_TICK_LINE_LENGTH;
  149:                 float lineY = (float) (Math.sin(Math.toRadians(-roll)) * radiusInt
ernal)
  150:                                 * PITCH_TICK_LINE_LENGTH;
  151:                 float dx = (float) (Math.cos(Math.toRadians(-roll - 90))
  152:                                 * radiusInternal / PITCH_RANGE);
  153:                 float dy = (float) (Math.sin(Math.toRadians(-roll - 90))
  154:                                 * radiusInternal / PITCH_RANGE);
  155:                 int i = (int) ((-PITCH_RANGE + pitch + PITCH_TICK_PADDING) / PITCH
_TICK_SPACING);
  156:                 int loopEnd = (int) ((PITCH_RANGE + pitch - PITCH_TICK_PADDING) /
PITCH_TICK_SPACING);
  157:                 for (; i <= loopEnd; i++) {
  158:                         float degree = -pitch + PITCH_TICK_SPACING * i;
  159:                         canvas.drawLine(lineX + dx * degree, lineY + dy * degree,
-lineX
  160:                                         + dx * degree, -lineY + dy * degree, tickP
aint);
  161:                 }
  162:         }
  163:
  164:         private void drawPlane(Canvas canvas) {
  165:                 canvas.drawPath(planePath, planePaint);
  166:                 canvas.drawCircle(0, 0, radiusInternal * PLANE_SIZE * PLANE_BODY_S
IZE,
  167:                                 planePaint);
  168:         }
  169:
  170:         public void setAttitude(float roll, float pitch, float yaw) {
  171:                 this.roll = roll;
  172:                 this.pitch = pitch;
  173:                 this.yaw = yaw;
  174:                 invalidate();
  175:         }
  176: }
```

```
    1: package com.droidplanner.widgets.RcStick;
    2:
    3: import android.content.Context;
    4: import android.graphics.Canvas;
    5: import android.graphics.Color;
    6: import android.graphics.Paint;
    7: import android.graphics.Paint.Style;
    8: import android.graphics.RectF;
    9: import android.util.AttributeSet;
   10: import android.view.View;
   11:
   12: public class RcStick extends View {
   13:
   14:         private static final float STICK_SIZE = 0.3f;
   15:
   16:
   17:         private Paint paintOutline;
   18:         private Paint paintFill;
   19:         private int height;
   20:         private int width;
   21:         private int xPos, yPos;
   22:         private int stickRadius;
   23:         private RectF borders;
   24:
   25:         public RcStick(Context context, AttributeSet attrs) {
   26:                 super(context, attrs);
   27:                 initialize();
   28:         }
   29:
   30:         private void initialize() {
   31:                 paintOutline = new Paint();
   32:                 paintOutline.setAntiAlias(true);
   33:                 paintOutline.setStyle(Style.STROKE);
   34:                 paintOutline.setStrokeWidth(3);
   35:                 paintOutline.setColor(Color.parseColor("#E0E0E0"));
   36:
   37:                 paintFill = new Paint(paintOutline);
   38:                 paintFill.setStyle(Style.FILL);
   39:         }
   40:
   41:         @Override
   42:         protected void onSizeChanged(int w, int h, int oldw, int oldh) {
   43:                 super.onSizeChanged(w, h, oldw, oldh);
   44:                 width = w - 1;
   45:                 height = h - 1;
   46:                 borders = new RectF(1, 1, width, height);
   47:                 stickRadius = (int) (STICK_SIZE*Math.min(width, height)/2);
   48:                 setPosition(0, 0);
   49:         }
   50:
   51:         @Override
   52:         protected void onDraw(Canvas canvas) {
   53:                 super.onDraw(canvas);
   54:
   55:                 canvas.drawRoundRect(borders, stickRadius, stickRadius, paintOutli
ne);
   56:
   57:                 canvas.drawCircle(xPos, yPos, stickRadius, paintFill);
   58:         }
   59:
   60:         public void setPosition(float x, float y) {
   61:                 xPos = (int) ((width - 2 * stickRadius) * ((1 + x) / 2f))
   62:                                 + stickRadius;
   63:                 yPos = (int) ((height - 2 * stickRadius) * ((1 - y) / 2f))
   64:                                 + stickRadius;
   65:                 invalidate();
   66:         }
   67:
   68: }
```

```
    1: package com.droidplanner.widgets.SeekBarWithText;
    2:
    3: import android.content.Context;
    4: import android.content.res.TypedArray;
    5: import android.util.AttributeSet;
    6: import android.widget.LinearLayout;
    7: import android.widget.SeekBar;
    8: import android.widget.SeekBar.OnSeekBarChangeListener;
    9: import android.widget.TextView;
   10:
   11: import com.droidplanner.R;
   12:
   13: public class SeekBarWithText extends LinearLayout implements
   14:                 OnSeekBarChangeListener {
   15:
   16:         public interface OnTextSeekBarChangedListner {
   17:                 public void onSeekBarChanged();
   18:         }
   19:
   20:         private TextView textView;
   21:         private SeekBar seekBar;
   22:         private double min = 0;
   23:         private double inc = 1;
   24:         private String title = "";
   25:         private String unit = "";
   26:         private String formatString = "%2.1f";
   27:         private OnTextSeekBarChangedListner listner;
   28:
   29:         public SeekBarWithText(Context context) {
   30:                 super(context);
   31:                 createViews(context);
   32:         }
   33:
   34:         public SeekBarWithText(Context context, AttributeSet attrs) {
   35:                 super(context, attrs);
   36:                 createViews(context);
   37:                 TypedArray a = context.getTheme().obtainStyledAttributes(attrs,
   38:                                 R.styleable.SeekBarWithText, 0, 0);
   39:
   40:                 try {
   41:                         setTitle(a.getString(R.styleable.SeekBarWithText_title));
   42:                         setUnit(a.getString(R.styleable.SeekBarWithText_unit));
   43:                         setMinMaxInc(a.getFloat(R.styleable.SeekBarWithText_min, 0
),
   44:                                         a.getFloat(R.styleable.SeekBarWithText_max
, 100),
   45:                                         a.getFloat(R.styleable.SeekBarWithText_inc
, 1));
   46:                         setFormat(a.getString(R.styleable.SeekBarWithText_formatSt
ring));
   47:                 } finally {
   48:                         a.recycle();
   49:                 }
   50:         }
   51:
   52:         private void setFormat(String string) {
   53:                 if (string!=null) {
   54:                         formatString = string;
   55:                 }
   56:         }
   57:
   58:         private void createViews(Context context) {
   59:                 setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
   60:                                 LayoutParams.WRAP_CONTENT));
   61:                 setOrientation(VERTICAL);
   62:                 textView = new TextView(context);
   63:                 seekBar = new SeekBar(context);
   64:                 seekBar.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT
,
   65:                                 LayoutParams.WRAP_CONTENT));
   66:                 seekBar.setOnSeekBarChangeListener(this);
   67:                 addView(textView);
   68:                 addView(seekBar);
   69:         }
   70:
   71:         public void setMinMaxInc(double min, double max, double inc) {
   72:                 this.min = min;
   73:                 this.inc = inc;
   74:                 seekBar.setMax((int) ((max - min) / inc));
   75:         }
   76:
   77:         public void setUnit(String unit) {
   78:                 if (unit != null) {
   79:                         this.unit = unit;
   80:                 }
   81:         }
   82:
   83:         public void setTitle(CharSequence text) {
   84:                 if (text != null) {
   85:                         title = text.toString();
   86:                         updateTitle();
   87:                 }
   88:         }
   89:
   90:         private void updateTitle() {
   91:                 textView.setText(String.format("%s\t"+formatString+" %s", title, g
etValue(), unit));
   92:         }
   93:
   94:         public double getValue() {
   95:                 return (seekBar.getProgress() * inc + min);
   96:         }
   97:
   98:         public void setValue(double value) {
   99:                 seekBar.setProgress((int) ((value - min) / inc));
  100:         }
  101:
  102:         public void setAbsValue(double value) {
  103:                 if(value<0)
  104:                         value *= -1.0;
  105:                 seekBar.setProgress((int) ((value - min) / inc));
  106:         }
  107:
  108:         @Override
  109:         public void onProgressChanged(SeekBar seekBar, int progress,
  110:                         boolean fromUser) {
  111:                 updateTitle();
  112:         }
  113:
  114:         @Override
  115:         public void onStartTrackingTouch(SeekBar seekBar) {
  116:
  117:         }
  118:
  119:         @Override
  120:         public void onStopTrackingTouch(SeekBar seekBar) {
  121:                 if (listner != null) {
  122:                         listner.onSeekBarChanged();
  123:                 }
  124:         }
  125:
  126:         public void setOnChangedListner(OnTextSeekBarChangedListner listner) {
  127:                 this.listner = listner;
  128:         }
```

```
129:
130: }
```

```
    1: package com.droidplanner.widgets.spinners;
    2:
    3: import java.util.ArrayList;
    4: import java.util.List;
    5:
    6: import com.MAVLink.Messages.ApmModes;
    7:
    8: import android.content.Context;
    9: import android.view.View;
   10: import android.view.ViewGroup;
   11: import android.widget.ArrayAdapter;
   12: import android.widget.TextView;
   13:
   14: public class ModeAdapter extends ArrayAdapter<ApmModes> {
   15:         public ArrayList<ApmModes> modes = new ArrayList<ApmModes>();
   16:
   17:         public ModeAdapter(Context context, int resource, List<ApmModes> objects)
{
   18:                 super(context, resource, objects);
   19:                 modes.addAll(objects);
   20:         }
   21:
   22:         @Override
   23:         public View getView(int position, View convertView, ViewGroup parent) {
   24:                 TextView view = (TextView) super.getView(position, convertView, pa
rent);
   25:                 view.setText(modes.get(position).getName());
   26:                 return view;
   27:         }
   28:
   29:         @Override
   30:         public View getDropDownView(int position, View convertView, ViewGroup pare
nt) {
   31:                 return getView(position, convertView, parent);
   32:         }
   33:
   34: }
```

```
    1: package com.droidplanner.widgets.spinners;
    2:
    3: import android.content.Context;
    4: import android.widget.Spinner;
    5:
    6: import com.MAVLink.Messages.ApmModes;
    7: import com.droidplanner.R;
    8: import com.droidplanner.drone.Drone;
    9: import com.droidplanner.drone.DroneInterfaces.DroneTypeListner;
   10: import com.droidplanner.drone.DroneInterfaces.ModeChangedListener;
   11: import com.droidplanner.widgets.spinners.SpinnerSelfSelect.OnSpinnerItemSelectedLi
stener;
   12:
   13: public class SelectModeSpinner extends SpinnerSelfSelect implements
   14:                 OnSpinnerItemSelectedListener, DroneTypeListner, ModeChangedListen
er {
   15:
   16:         private ModeAdapter modeAdapter;
   17:         private Drone drone;
   18:         private Context context;
   19:
   20:         public SelectModeSpinner(Context context) {
   21:                 super(context);
   22:                 this.context = context;
   23:                 selectable = false;
   24:                 setBackgroundResource(R.drawable.modes);
   25:         }
   26:
   27:         public void buildSpinner(Context context, Drone drone) {
   28:                 this.drone = drone;
   29:                 setOnSpinnerItemSelectedListener(this);
   30:                 this.drone.setModeChangedListener(this);
   31:                 this.drone.setDroneTypeChangedListner(this);
   32:
   33:                 onDroneTypeChanged();
   34:         }
   35:
   36:         @Override
   37:         public void onDroneTypeChanged() {
   38:                 buildAdapter();
   39:         }
   40:
   41:         private void buildAdapter() {
   42:                 modeAdapter = new ModeAdapter(this.context,
   43:                                 android.R.layout.simple_spinner_dropdown_item,
   44:                                 ApmModes.getModeList(this.drone.type.getType()));
   45:                 setAdapter(modeAdapter);
   46:         }
   47:
   48:         @Override
   49:         public void onModeChanged() {
   50:                 try {
   51:                         this.forcedSetSelection(modeAdapter.getPosition(drone.stat
e.getMode()));
   52:                 } catch (Exception e) {
   53:                         e.printStackTrace();
   54:                 }
   55:         }
   56:
   57:         @Override
   58:         public void onSpinnerItemSelected(Spinner spinner, int position, String te
xt) {
   59:                 ApmModes newMode = modeAdapter.getItem(position);
   60:                 drone.state.changeFlightMode(newMode);
   61:         }
   62:
   63: }
```

```
    1: package com.droidplanner.widgets.spinners;
    2:
    3: import android.content.Context;
    4: import android.util.AttributeSet;
    5: import android.util.Log;
    6: import android.widget.Spinner;
    7:
    8: public class SpinnerSelfSelect extends Spinner {
    9:
   10:         public interface OnSpinnerItemSelectedListener {
   11:                 void onSpinnerItemSelected(Spinner spinner, int position, String t
ext);
   12:         }
   13:
   14:         OnSpinnerItemSelectedListener listener;
   15:
   16:         protected boolean selectable = true;
   17:
   18:         public SpinnerSelfSelect(Context context) {
   19:                 super(context);
   20:         }
   21:
   22:         public SpinnerSelfSelect(Context context, AttributeSet attrs) {
   23:                 super(context, attrs);
   24:         }
   25:
   26:         @Override
   27:         public void setSelection(int position) {
   28:                 Log.d("SPIN", "selected - " + position);
   29:
   30:                 if (selectable) {
   31:                         forcedSetSelection(position);
   32:                 }
   33:
   34:                 if (listener != null) {
   35:                         listener.onSpinnerItemSelected(this, position,
   36:                                         getItemAtPosition(position).toString());
   37:                 }
   38:         }
   39:
   40:         public void forcedSetSelection(int position) {
   41:                 super.setSelection(position);
   42:         }
   43:
   44:         public void setOnSpinnerItemSelectedListener(
   45:                         OnSpinnerItemSelectedListener listener) {
   46:                 this.listener = listener;
   47:         }
   48:
   49: }
```

```
    1: package com.droidplanner.widgets;
    2:
    3: import android.os.Handler;
    4: import android.os.SystemClock;
    5: import android.view.MenuItem;
    6:
    7: public class TimerView {
    8:
    9:         private MenuItem timerValue;
   10:
   11:         private Handler customHandler = new Handler();
   12:
   13:         private long startTime = 0L;
   14:
   15:         public TimerView(MenuItem propeler) {
   16:                 this.timerValue = propeler;
   17:                 timerValue.setTitle("00:00");
   18:         }
   19:
   20:         public void reStart() {
   21:                 startTime = SystemClock.elapsedRealtime();
   22:                 customHandler.postDelayed(updateTimerThread, 0);
   23:         }
   24:
   25:         public void stop() {
   26:                 customHandler.removeCallbacks(updateTimerThread);
   27:         }
   28:
   29:         private Runnable updateTimerThread = new Runnable() {
   30:
   31:                 public void run() {
   32:
   33:                         long timeInSeconds = (SystemClock.elapsedRealtime() - star
tTime)/1000;
   34:
   35:                         long minutes = timeInSeconds/60;
   36:                         long seconds = timeInSeconds%60;
   37:                         timerValue.setTitle(String.format("%02d:%02d", minutes,sec
onds));
   38:
   39:                         customHandler.postDelayed(this, 1000);
   40:                 }
   41:
   42:         };
   43: }
```

```
    1: package com.droidplanner.widgets.viewPager;
    2:
    3: import android.content.Context;
    4: import android.support.v4.view.ViewPager;
    5: import android.util.AttributeSet;
    6: import android.view.View;
    7:
    8: /*
    9:  * Simple extension to {@link ViewPager} that allows the definition of
   10:  * the region where swiping is enabled
   11:  *
   12:  * Based on the work of MartinHochstrasser on:
   13:  * https://github.com/MartinHochstrasser/StickyViewPager/
   14:  */
   15: public class MapViewPager extends ViewPager {
   16:
   17:         private int swipeRegionWidth = 40;
   18:         private Context context;
   19:
   20:         public MapViewPager(Context context) {
   21:                 super(context);
   22:                 this.context = context;
   23:         }
   24:
   25:         public MapViewPager(Context context, AttributeSet attrs) {
   26:                 super(context, attrs);
   27:                 this.context = context;
   28:         }
   29:
   30:         @Override
   31:         protected boolean canScroll(View v, boolean checkV, int dx, int x, int y)
{
   32:                 if (isInsideSwipeRegion(x, dx)) {
   33:                         return super.canScroll(v, checkV, dx, x, y);
   34:                 }
   35:                 return true;
   36:         }
   37:
   38:         /**
   39:          * Determines if the movement is inside the
   40:          *
   41:          * @return true if is inside the region
   42:          */
   43:         protected boolean isInsideSwipeRegion(float x, float dx) {
   44:                 if (dx > 0) {
   45:                         return x < swipeRegionWidth;
   46:                 } else {
   47:                         return x > (getWidth() - swipeRegionWidth);
   48:                 }
   49:         }
   50:
   51:         /**
   52:          * Sets the width of the swipeable region
   53:          *
   54:          * @param width
   55:          */
   56:         public void setSwipeMarginWidth(final int width) {
   57:                 swipeRegionWidth = (int) (width * context.getResources()
   58:                                 .getDisplayMetrics().density);
   59:         }
   60:
   61: }
```