Qt C++ and QML integration, context properties and GPS compass

This example creates an application that gives the user direction and distance to a specified location based on their current co-ordinate. It is written in both QML and Qt C++, QML and uses the Qt Mobility 1.0 Location module and in some parts a Maemo-specific location library.

26 Dec 2010

History

A request for map-less geolocator appeared. Essentially, the program should use the current GPS coordinate to show direction and distance to a user-chosen coordinate so that a user wouldn't need to open a map and use Internet access data plan to download map tiles when he needs just a general direction, not exact route.

These code snippets of the new program are the basis of the article with leading explanations as a tutorial and how-to.

Interface

Building blocks

Each QML file must have exactly one root element: Item, Rectangle, anything.

Dial

The most comfortable way to show direction to point is round dial. It shows azimuth, with North at the top, as orientation of the device and the human relatively to the device isn't known well enough to attempt otherwise. Besides, it includes labels with relative (from current coordinate to endpoint) distance and altitude.

The first line imports Qt. It's necessary in order to be able to use basic QML elements, such as Item.

```
//Dial.qml
import Qt 4.7

 Item
     {
     id: root //unique identifier
     property real value : 0
     property color needle : "blue"
     property real dist : 0
     property real alti : 0
     width: 200
     height: 200+dist.height+alti.height //Don't assume beforehand the height of labels; it can depend on different factors
     Rectangle
     {
         id: needle
         width: 2
         height: 99
         color: root.needle
         x: 99; y: 1 //Some space between the end of the needle and the border of the dial
         smooth: true //Takes resources, especially during animation
         transform: Rotation
           {
             id: needleRotation
             origin.x: 1
             origin.y: 99
             angle: Math.min(Math.max(0, root.value), 360) //The needle rotates automatically when value changes due to property bin
             Behavior on angle
                {
                   SpringAnimation//Cosmetic animation; unnecessary
                   {
                     spring: 1.4
                     damping: .15
                   }
                }
           }
     }
           Text
            {
               id: dist
               text: "distance: "+root.dist+" meters"
               x: 0; y: 200
            }
           Text
            {
               id: alti
               text: "relative altitude: "+root.alti+" meters"
               x: 0; y: 200+dist.height //should be done with anchors; works right now; just an example how it shouldn't be done
            }
 }
```

Coordinate

 (C) Copyright Nokia 2013. All rights reserved.

Note that in Qt Mobility 1.1 some new element will appear, which will almost implement this functionality. However, it doesn't store an altitude value. QML doesn't have its own way to request and receive current location.

```
//Coordinate.qml
import Qt 4.7

Rectangle {
     id: container
    property real cx: 0 //Real and double are interchangeable, equivalent in QML.
    property real cy: 0
    property real cz: 0
     width: (x.width+y.width+z.width)*1.1
     height: x.height*1.2 //Just leaving some gap.
     border
      {
        width: 1
        color: "#000000"
      } //To see the border of the element.
       Text
       {
           id: x
           anchors.left: parent.left
           anchors.top: parent.top
           text: "lat: "+container.cx //The label not only displays the value but also gives some guidance. Note: no type conversion
       } //Now, anchors! Vertical anchor isn't important here, but horizontal anchor is determining the layout.
       Text
       {
           id: y
           anchors.right: parent.right
           anchors.top: parent.top
           text: "lon: "+container.cy
       }
       Text
       {
           id: z
           anchors.horizontalCenter: parent.horizontalCenter
           anchors.top: parent.top
           text: "alt: "+container.cz+" meters"
       }
 }
```

Point-Of-Interest

This element uses above-described Coordinate.

```
//POI.qml
import Qt 4.7

Rectangle {
     id: container
    signal clicked //This element is interactive, it can receive clicks/taps
    property string name: "name"
    property real cx: 0
    property real cy: 0
    property real cz: 0
    property bool showname: true //whether the user looks at the name or at the coordinates
    property bool edit: false //It will be used later, not here.
     width: {if(showname){return (name.width)*1.1;}else{return (coord.width)*1.1;}} //Fit the contents
     height: (coord.height)*1.1 //Both labels have equal height.
              //But if you change layout of coordinate, you would better write the height just like the width: autofitting
     color: {if(activeFocus){return "#0000ff";}else{return "#ffffff";}}
       //Notifying the user the element gained activeFocus. Will be used later.
     border
      {
        width: 1
        color: "#000000"
      }
       Text
       {
           id: name
           anchors.horizontalCenter: parent.horizontalCenter
           anchors.top: parent.top
           text: parent.name
           visible: parent.showname
       }
       Coordinate
       {
           id: coord
           anchors.horizontalCenter: parent.horizontalCenter
           anchors.bottom: parent.bottom //The top/bottom anchors are remnants of showing both the name and the coordinates
           cx: parent.cx
           cy: parent.cy
           cz: parent.cz
           visible: !(parent.showname)
       }
     MouseArea
      {
         id: mouseArea
         anchors.fill: parent
         onClicked: {container.showname=!(container.showname); container.clicked();}
            //This area is necessary to get ability to receive clicks. And don't forget anchors.fill
      }
 }
```

Button

When the user wants to change the endpoint/goal, when he wants to add a new Point-Of-Interest to the list, he expects to click a button to do it. So here is a button.

```
//Button.qml
import Qt 4.7

Rectangle {
     id: container
    property string text: "Button"
```

 (C) Copyright Nokia 2013. All rights reserved.

```
    property bool enabled: true
     signal clicked
     width: buttonLabel.width*1.2
     height: buttonLabel.height*1.2
     color: {
                if (enabled)
                    return "#D0D0D0"
                else
                    return "#DCDCDC"
            }
     border
      {
        width: 1
        color: "#000000"
      }
     MouseArea
      {
         id: mouseArea
         anchors.fill: parent
         onClicked: if(container.enabled){container.clicked();}
      }
     Text
      {
         id: buttonLabel
         anchors.centerIn: container
     color: {
                if (container.enabled)
                    return "#000000"
                else
                    return "#8A8A8A"
            }
         text: container.text
      }
 }
```

Just a simple button. If you want more states than enabled/disabled, like pressed, use state and states.

Window

The main window/view is based on the above-mentioned building blocks.

```
//ui.qml
import Qt 4.7
import "content" //Imports the building blocks from neighboring directory ./content

Rectangle {
    id: root;
    anchors.fill: parent; //Fill the whole view/window
    color: "white"
    width: 400
    height: 400 //In fact, unnecessary and doesn't work; anchors take precedence
       Dial
       {
           id: dial
           anchors.centerIn: parent
           value: qlgps.pangle
           dist: qlgps.dist
           alti: qlgps.alti
```

qlgps is a context property given in C++ code; property binding updates dial automatically

```
           working: qlgps.con
```

Now, working property of the dial is used... And it's awkward.

Either remove needle property and move color-binding to Dial.qml, or remove working property, and bind color directly to qlgps.con

```
           needle: {if(working){return "#00ff00";}else{return "#ff0000";}}
       }
       POI
       {
           id: goal
           name: "Goal"
           anchors.horizontalCenter: parent.horizontalCenter
           anchors.top: root.top
           cx: qlgps.gx
           cy: qlgps.gy
           cz: qlgps.gz
       }
```

The endpoint/goal coordinates are at the top, above the dial.

To not take screen space, only the name "Goal" is displayed by default - click it to see the longitude, latitude and altitude.

To prevent accidental editing, a special small button is placed here for switching to the editing mode, which enables the other buttons.

```
       Button
       {
           id: edit
           text: {if(regoal.enabled){return "Stop editing";}else{return "Edit";}}
```

The text tells whether editing is enabled or not.

```
           enabled: true
           anchors.right: root.right
           anchors.bottom: root.bottom
          onClicked: {regoal.enabled = !(regoal.enabled); add.enabled = !(add.enabled); name.activeFocusOnPress = !(name.activeFocus
```

 (C) Copyright Nokia 2013. All rights reserved.

This piece of code is too long. It would be better to bind all these properties to one property, and change this one property.

```
        }
        Button
        {
            id: regoal
            text: "Set current coordinate as goal"
            enabled: false
            anchors.bottom: root.bottom
            anchors.horizontalCenter: root.horizontalCenter
           onClicked: {qlgps.regoal();}
        }
```

This button sets current coordinate as endpoint. It can be used when user doesn't know the surroundings well and wants to be able to return to hotel/car/underground station.

```
        Flow {id: flow; flow: Flow.TopToBottom; anchors.bottom: root.bottom; anchors.top: root.top; }
```

This Flow element is going to contain list of Points-Of-Interest to choose from. (Notice that a semicolon is necessary in QML wherever a line break is absent.)

```
        Button{
id: add
text: "Add"
enabled: false
anchors.bottom: root.bottom
anchors.left: root.left
onClicked: {
var idid = Qt.formatDateTime(new Date(), "_yyyy_MMMM_dd_dddd_hh_mm_ss_zzz");
Qt.createQmlObject ('import Qt 4.7; import "content"; POI{id: '+idid+'; name: "'+name.text+'"; cx: '+latitude.text+'; cy: '+longitud
          }
```

New Point-Of-Interest is added to the list. Its id is set from current time so that it is unique. Note how a separate local variable idid is created for storing it; otherwise, one millisecond of difference is enough to ruin destroy part of the code.

Note that in dynamically created element nothing that wasn't imported in static document can be imported. If needed, create component instead.

Note that edit mode is bound to another outer element with fixed id. When edit mode is enabled, the point can be deleted with 'D', set as endpoint with 'S'. In any mode clicking the point brings active focus to it. It's almost pointless in read-only mode, but at least it provides visible feedback. And it's necessity in edit mode to get key presses.

Ideally, pressing arrow up/down would go to previous/next point. But id of previous/next point is needed for it. It would be solved by implementing an array of ids. Besides, it would be a good idea to allow editing of data inside a point instead of deleting and recreating it.

Now, the input for a new Point-Of-Interest. Some restrictions are needed to ensure validity of each point.

```
        Text{id: nal; text: "Name: "; anchors.bottom: add.top; anchors.left: root.left;}
        TextInput{id: name; anchors.bottom: add.top; anchors.left: nal.right; width: 100; height: 33; activeFocusOnPress: false;}
        Text{id: altil; text: "Altitude: "; anchors.bottom: nal.top; anchors.left: root.left;}
        TextInput{id: altitude; anchors.bottom: name.top; anchors.left: altil.right; width: 100; height: 33; activeFocusOnPress: fals
        Text{id: longl; text: "Longitude: "; anchors.bottom: altil.top; anchors.left: root.left;}
        TextInput{id: longitude; anchors.bottom: altitude.top; anchors.left: longl.right; width: 100; height: 33; activeFocusOnPress:
        Text{id: latl; text: "Latitude: "; anchors.bottom: longl.top; anchors.left: root.left;}
        TextInput{id: latitude; anchors.bottom: longitude.top; anchors.left: latl.right; width: 100; height: 33; activeFocusOnPress:

        Button
        {
            id: setcurrenttopoi
            text: "<<--"
            enabled: true
            anchors.bottom: altitude.bottom
            anchors.top: latitude.top
            anchors.left: longitude.right
            anchors.right: dial.left
           onClicked: {altitude.text=qlgps.cz; longitude.text=qlgps.cy; latitude.text=qlgps.cx;}
        }
```

Comfortable tool to allow adding current coordinates as a point to the list.

```
        Button
        {
            id: saveto
            text: "Save to"
            enabled: true
            anchors.bottom: root.bottom
            anchors.left: add.right
           onClicked: {qlgps.setFlaw(flow.toString());}
        }
```

Save string of points to settings.

```
        Button
        {
            id: readfrom
            text: "Read from"
            enabled: true
            anchors.bottom: root.bottom
            anchors.left: saveto.right
           onClicked: {Qt.createQmlObject (qlgps.flaw(), flow);}
        }
```

Appending a string of points from the settings to the list. Overuse of this button may result in duplication or tripling of the list.

 (C) Copyright Nokia 2013. All rights reserved.

```
}
```

Note that "save to" button does not work, because flow.toString() function returns, instead of inner/outer QML code, just one word: QDeclarativeFlow.

Retrieving and using the location information (C++ code)

C++ part of the application

Headers

Most Qt headers, as well as custom class definition, are inside separate header file. It would be better to move these two headers, too.

```
#include <QtGui/QApplication>
#include <QtGui/QGraphicsObject>
#include "examplewidget.h"
```

Global variable

Defining a, b, c and app global variables. Note that declaration of a, b and c could probably be replaced with &(&'c'), but &&'c' didn't work at this time.

```
char a = 'c';
char * b = &a;
char ** c = &b;
     QDialGPS app(0, c);
```

Defining constructor of custom class

qDebug outputs some text in case there is need to know whether this function was executed

Properties are given default values, different from normal value, so that it would be known whether this property was changed after constructor.

Until connection is established, the application is presumed to be non-connected

```
QDialGPS::QDialGPS(int argc, char** argv) :  QApplication(argc, argv)
 {
 qDebug("constructor started");
angle = 99;
x=1;
y=2;
z=11111111;
goalx=1111;
goaly=1111;
goalz=11111111;
setCon(false);
 qDebug("constructor ended");
}
```

Defining a function of custom class

```
void QDialGPS::regoal()
{
 qDebug("regoaling");
if(x!=1111)
{
    setGx(x);
}
if(y!=1111)
{
    setGy(y);
}
if(z!=11111111)
{
    setGz(z);
}
 qDebug("regoaled");
}
```

Non-class functions

They use the global variable. Note that properties are set in special way here. angle, x, y, z, etc. are private, so instead of direct assignment with "=" operator function setProperty is used from outside of the class, signal is emitted, and interface is notified of the change.

If the goal wasn't set yet, assign it automatically.

```
static void changed(LocationGPSDevice *device, gpointer pointer)
{
 qDebug("changed");
```

 (C) Copyright Nokia 2013. All rights reserved.

```
app.setCx(device->fix->latitude);
app.setCy(device->fix->longitude);
if(device->fix->altitude)
 {
  app.setCz(device->fix->altitude);
 }
if(app.gx()==1111)
 {
   app.setGx(app.cx());
 }
if(app.gy()==1111)
 {
   app.setGy(app.cy());
 }
if(app.gz()==11111111)
 {
      app.setGz(app.cz());
 }
 app.setPangle(Coordination(app.cx(),app.cy()).azimuthTo(QGeoCoordinate (app.gx(),app.gy())));
 app.setDist(0);
 app.setAlti(0);
 qDebug("applied");
}
```

In fact, angle is always calculated by this formula, so it would be better to put it inside setPangle instead, like for setDist and setAlti

```
static void connected(LocationGPSDevice *device, gpointer pointer)
{
    qDebug("connection starting");
    app.setCon(true);
    qDebug("connection started");
}

static void disconnected(LocationGPSDevice *device, gpointer pointer)
{
    qDebug("connection ending");
    app.setCon(false);
    qDebug("connection ended");
}
```

Main function

app is already declared as global variable; there is no need to declare it as a local variable.

QDeclarativeView is the way to create interface from a QML file.

GPS request uses external C headers and non-Qt library. It could possibly be done in Qt, but how can GPS be turned on in Qt?

These notable checks whether an object exists before addressing to it seem paranoid, but they are better than sudden crash, especially if GPS is an auxiliary part of application.

The object app of custom class is notified of all changes in GPS device and changes its properties accordingly.

Thanks to context property, the QML file can see the custom object qlgps and access its properties and functions.

Note the order of the two lines, setContextProperty and setSource! If a context property is set after the source was set, the QML file will not see the property.

```
 int main(int argc, char *argv[])
 {
//     app=QDialGPS::QDialGPS(argc, argv);

     QDeclarativeView view;

    LocationGPSDevice *device;
    LocationGPSDControl *control;
    g_type_init ();
    control = location_gpsd_control_get_default();
    if (control)
        location_gpsd_control_start(control);

    device = (LocationGPSDevice*)g_object_new(LOCATION_TYPE_GPS_DEVICE, NULL);
    if (device) {
        g_signal_connect(device, "changed", G_CALLBACK(changed), 0);
        g_signal_connect(device, "connected", G_CALLBACK(connected), 0);
        g_signal_connect(device, "disconnected", G_CALLBACK(disconnected), 0);
    }

    view.rootContext()->setContextProperty("qlgps", &app);

     view.setSource(QUrl("/opt/usr/bin/qtdialgps/ui.qml"));

     view.setResizeMode(QDeclarativeView::SizeRootObjectToView);

     qDebug("connected?");
      #if defined(Q_WS_S60) || defined(Q_WS_MAEMO)
        view.showMaximized();
     #else
       view.setGeometry(100, 100, 800, 480);
       view.show();
     #endif

qDebug("view shown");

    int ret = app.exec();

    if (device)
        g_object_unref(device);
    if (control)
        location_gpsd_control_stop(control);

    return ret;
 }
```

 (C) Copyright Nokia 2013. All rights reserved.

Switching between S60, Maemo and other platforms is presumed to be good practice, this part can work on any platform.

Note how this last part of code destroys device and control when they are no longer needed, after execution of the application, which used them. Most likely, global variable app should also be destroyed here.

Header part of the application

Out of necessity, this line was added.

```
using namespace QtMobility;
```

Without it, some errors appeared while attempting to use its location-related classes.

Headers

There are obviously too much of them. However, it's difficult to decide which of them are redundant.

```
#include <QtDeclarative/qdeclarativeview.h>
#include <QtDeclarative/QDeclarativeContext>
```

QtDeclarative is necessary for showing QML file and setting its context property.

```
#include <QtCore/QObject>
#include <QtCore/QtCore>
#include <QtCore/QVariant>
#include <QtGui/QApplication>
#include <QtCore/QCoreApplication>
#include <QtCore/QtGlobal>
```

Qt Mobility Location module is used for precise calculation of azimuth and distance from one point to another.

```
#include <QtLocation/QGeoPositionInfoSource>
#include <QtLocation/QGeoSatelliteInfoSource>
#include <QtLocation/QGeoCoordinate>

#include <QtCore/QSettings>
#include <QDebug>
extern "C" {
#include <location/location-gps-device.h>
#include <location/location-gpsd-control.h>
}
```

Note the non-Qt libraries, most likely platform specific.

Custom class declaration

This custom class has a lot of variables inside. Q_PROPERTY uses signals to automatically synchronize with the QML file, which uses one object of this class as context property.

Some properties are representing not private variables of class, but calculations based on them.

The functions to be called from QML must either:

1. be marked as Q_INVOKABLE and be in public (not private);
2. be in public slots;
3. both.

```
class QDialGPS : public QApplication
{
    Q_OBJECT
    Q_PROPERTY(double pangle READ pangle WRITE setPangle NOTIFY pangleChanged)
    Q_PROPERTY(double dist READ dist WRITE setDist NOTIFY distChanged)
    Q_PROPERTY(double alti READ alti WRITE setAlti NOTIFY altiChanged)
    Q_PROPERTY(double gx READ gx WRITE setGx NOTIFY gxChanged)
    Q_PROPERTY(double gy READ gy WRITE setGy NOTIFY gyChanged)
    Q_PROPERTY(double gz READ gz WRITE setGz NOTIFY gzChanged)
    Q_PROPERTY(bool con READ con WRITE setCon NOTIFY conChanged)
    Q_PROPERTY(double cx READ cx WRITE setCx NOTIFY cxChanged)
    Q_PROPERTY(double cy READ cy WRITE setCy NOTIFY cyChanged)
    Q_PROPERTY(double cz READ cz WRITE setCz NOTIFY czChanged)

 public:
    QDialGPS(int argc, char** argv);
    double pangle() const { return angle; }
    void setPangle(const double &e) {
      if (e != angle)
      {
       angle = e;
       emit pangleChanged();
      }
    }
    double dist() const { return QGeoCoordinate(x,y).distanceTo(QGeoCoordinate (goalx,goaly)); }
    void setDist(const double &d) {
       emit distChanged();
    }
    double alti() const { if((goalz!=11111111)&&(z!=11111111)){ return goalz - z; }else{return 0;}}
    void setAlti(const double &d) {
       emit altiChanged();
    }
    bool con() const { return connected; }
    void setCon(const bool &e) {
```

 (C) Copyright Nokia 2013. All rights reserved.

```
      if (e != connected)
      {
       connected = e;
       emit conChanged();
      }
    }
    double gx() const { return goalx; }
    Q_INVOKABLE void setGx(const double &e) {
      if (e != goalx)
      {
       goalx = e;
       emit gxChanged();
      }
    }
    double gy() const { return goaly; }
    Q_INVOKABLE void setGy(const double &e) {
      if (e != goaly)
      {
       goaly = e;
       emit gyChanged();
      }
    }
    double gz() const { return goalz; }
    Q_INVOKABLE void setGz(const double &e) {
      if (e != goalz)
      {
       goalz = e;
       emit gzChanged();
      }
    }
    double cx() const { return x; }
    void setCx(const double &e) {
      if (e != x)
      {
       x = e;
       emit cxChanged();
      }
    }
    double cy() const { return y; }
    void setCy(const double &e) {
      if (e != y)
      {
       y = e;
       emit cyChanged();
      }
    }
    double cz() const { return z; }
    void setCz(const double &e) {
      if (e != z)
      {
       z = e;
       emit czChanged();
      }
    }
    Q_INVOKABLE QString flaw() const {QSettings settings("QDialGPS", "QDialGPS"); qDebug(qPrintable(settings.value("POI").toString()
    Q_INVOKABLE void setFlaw(QString eee) {
qDebug(qPrintable(eee));
       QSettings settings("QDialGPS", "QDialGPS");
QString set = eee.replace("Flow(.*){(.*)id: flow;(.*)flow: Flow.TopToBottom;(.*)anchors.bottom: root.bottom;(.*)anchors.top: root.to
qDebug(qPrintable(set));
       settings.setValue("POI", set);
    }
 public slots:
    Q_INVOKABLE void regoal();
 signals:
    void pangleChanged();
    void distChanged();
    void altiChanged();
    void gxChanged();
    void gyChanged();
    void gzChanged();
    void conChanged();
    void cxChanged();
    void cyChanged();
    void czChanged();
 private:
    double x;
    double y;
    double z;
    double angle;
    bool connected;
    double goalx;
    double goaly;
    double goalz;
};
```

Here a Regular Expression is used. .* denotes any number of any characters, and when in brackets like (.*), it captures text and is denoted in replacing expression as \1, $2, ... , \98, or $99, depending on number of capturing places and place in their list.

Future prospects

- Make Points of Interest save-able.
- Make editing and navigating the list of points easier.
- Add a second dial, showing current direction and speed of user's movement, to compare it with direction and distance to endpoint.

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.