QmlWebMapsLocation

Using location data and web maps in QML

Please Note: Qt 4.7.1 and Mobility API 1.1 was used in this example. Application was tested in N900 & Simulator

In this wiki article I describe how you can use web maps and location data in your Qt Quick application. We also see how to integrate CPP and QML code and QML Code and JS code running in webkit. We also see how easy it is to use web APIs. Image below shows the application UI.

Application can show device's location using:

1. OVI maps
2. Google maps
3. Bing (Microsoft) maps

You can find the application's latest source code from this .zip file File:Qmlwebmap.zip

Application code consists of three major parts:

1. CPP part (Provides Qt application environment and gets location updates from GPS sensors, sone parts of the UI are also here.)
2. QML part (Implements major part of the UI by using WebView element)
3. HTML part (HTML files used with maps)

Cpp

C++ code part has one class and main function. I have also included one UI form that has only combobox for map provder selection. Here's the definition of the class:

```
class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    Q_PROPERTY(QString mapProvider READ mapProvider WRITE setMapProvider NOTIFY
mapProviderChanged)
    QString mapProvider();
    void setMapProvider(QString newMapProvider);

private slots:
    void mapProviderSelected(int);
    void positionUpdated(QGeoPositionInfo);

signals:
    void mapProviderChanged(QString);
    void posUpdated(double lat, double lon);

private:
    Ui::Widget *ui;
    QDeclarativeView* m_view;
    QGeoPositionInfoSource* m_geoSource;
    QString m_mapProvider;
};
```

We have a class that inherits QWidget. The instance of this class will be our top-level widget. In the public section we define one property (mapProvider) that is used from QML side. It contains html file name that is used in webView.

Then we define two slots. First one is used when user selects map provider from combo box and second one is connected to QGeoPositionInfoSources' signal positionUpdated. This signal is emitted when device's location changes.

In the signal section we define own signals for map provider change & location update.

As member data we have our ui form, QDeclarativeView (runs QML), QGeoPositionInfoSource for location updates and a string for storing map provider.

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    m_mapProvider = "init.html";
    ui->setupUi(this);
    m_view = new QDeclarativeView(this);
    m_view->rootContext()->setContextProperty("cppEngine", this);
    m_view->setSource(QUrl("qrc:/qmls/main.qml"));
    ui->verticalLayout->addWidget(m_view);
    m_view->setResizeMode(QDeclarativeView::SizeRootObjectToView);

    ui->comboBox->addItem("Select Map Provider", QVariant("init.html"));
    ui->comboBox->addItem("OVI Maps", QVariant("ovimaps.html"));
    ui->comboBox->addItem("Google Maps", QVariant("gmaps.html"));
    ui->comboBox->addItem("Bing Maps", QVariant("bingmaps.html"));

    m_geoSource = QGeoPositionInfoSource::createDefaultSource(this);

    if (m_geoSource)
    {
        connect(m_geoSource, SIGNAL(positionUpdated(QGeoPositionInfo)), this,
SLOT(positionUpdated(QGeoPositionInfo)));
        m_geoSource->setUpdateInterval(10000);
        m_geoSource->startUpdates();
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    connect(ui->comboBox, SIGNAL(currentIndexChanged(int)),
            this, SLOT(mapProviderSelected(int)));
}
```

In constructor we create application UI and load qml file to QDeclarativeView. We also publish instance of this class (setContextProperty) to QML side, because we want to use signals, slots and properties of this class there. We also set resize mode to SizeRootObjectToView and our root element will automatically have a maximum size. We also create geo position info source and start to listen location updates.

```
void Widget::setMapProvider(QString newMapProvider)
{
    if (m_mapProvider == newMapProvider)
        return;
    else
    {
        m_mapProvider = newMapProvider;
        emit mapProviderChanged(m_mapProvider);
    }
}

void Widget::positionUpdated(QGeoPositionInfo info)
{
    emit posUpdated( info.coordinate().latitude(), info.coordinate().longitude());
}
```

In setMapProvider method we set new map provider and emit a signal about the change. In positionUpdated slot we emit new signal that is used on QML side.

QML

QML code is quite short:

```
Item {

    function mapHtmlFileName() {
         var url = "/htmls/" + cppEngine.mapProvider;
         return url;
         }

    function positionUpdated(lat, lon)
        {
        mapView.evaluateJavaScript("pan("+lat+","+lon+");");
        }

    Connections {
        target: cppEngine;
        onPosUpdated: positionUpdated(lat, lon)
    }

    WebView {
        id: mapView
        url: mapHtmlFileName()
        preferredWidth: parent.width
        preferredHeight: parent.height
        scale: 1
        smooth: false
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }
}
```

In Connection object we connect posUpdated signal to positionUpdated JavaScript method. In that method we call webView's evaluateJavaScript method. It will execute code in weViews JavaScript environment. We also use cppEngine's mapProvider property to define file that is loaded to webView.

HTML

In htmls' we have separate html files for each map provider (Google Maps, Bing Maps or OVI Maps). Each file defines JavaScript method pan() that is used from QML. pan() method then moves map to device's GPS location.

(C) Copyright Nokia 2014. All rights reserved.