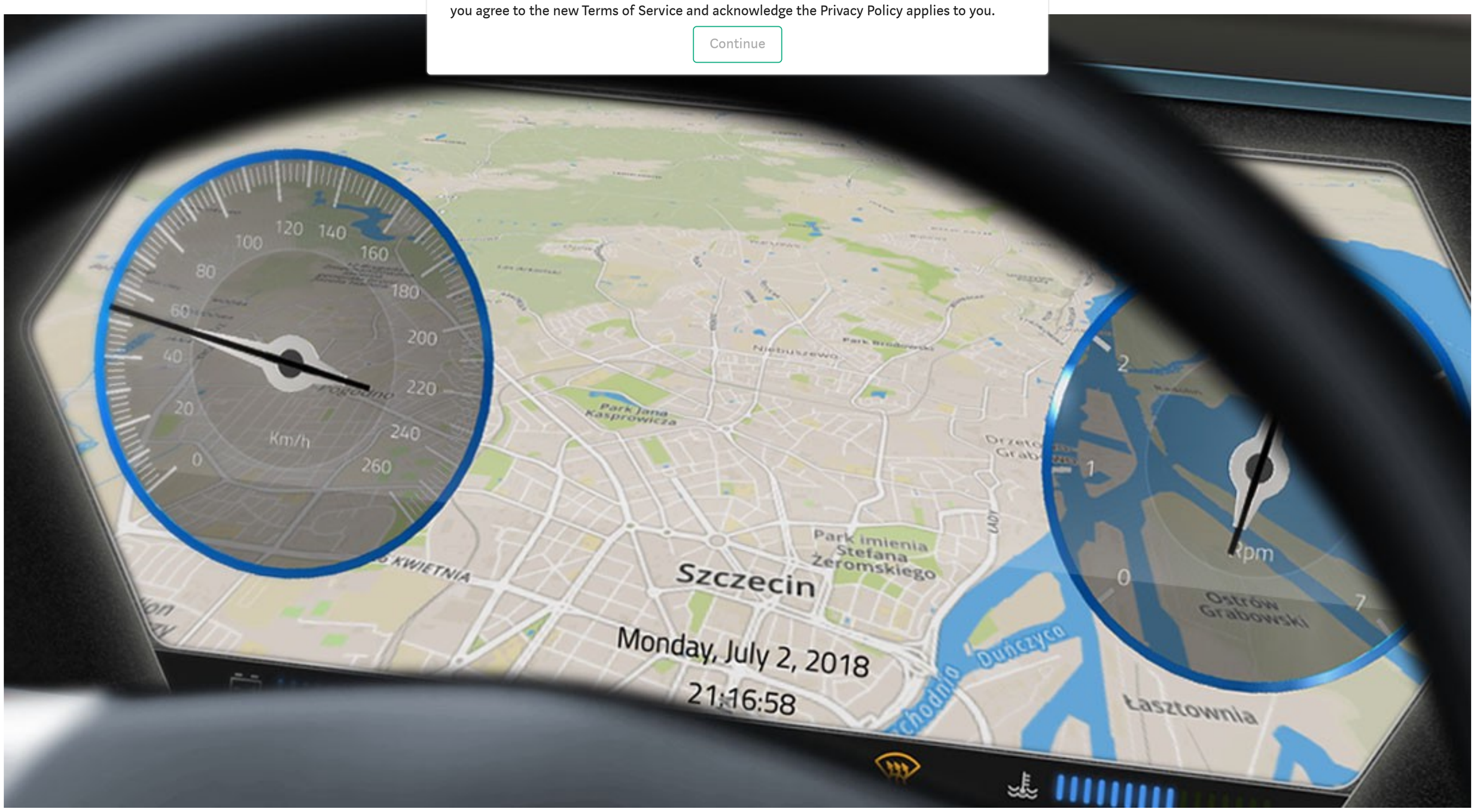


Qt 3D Studio — Digital cluster for a car — Integration (Part I)

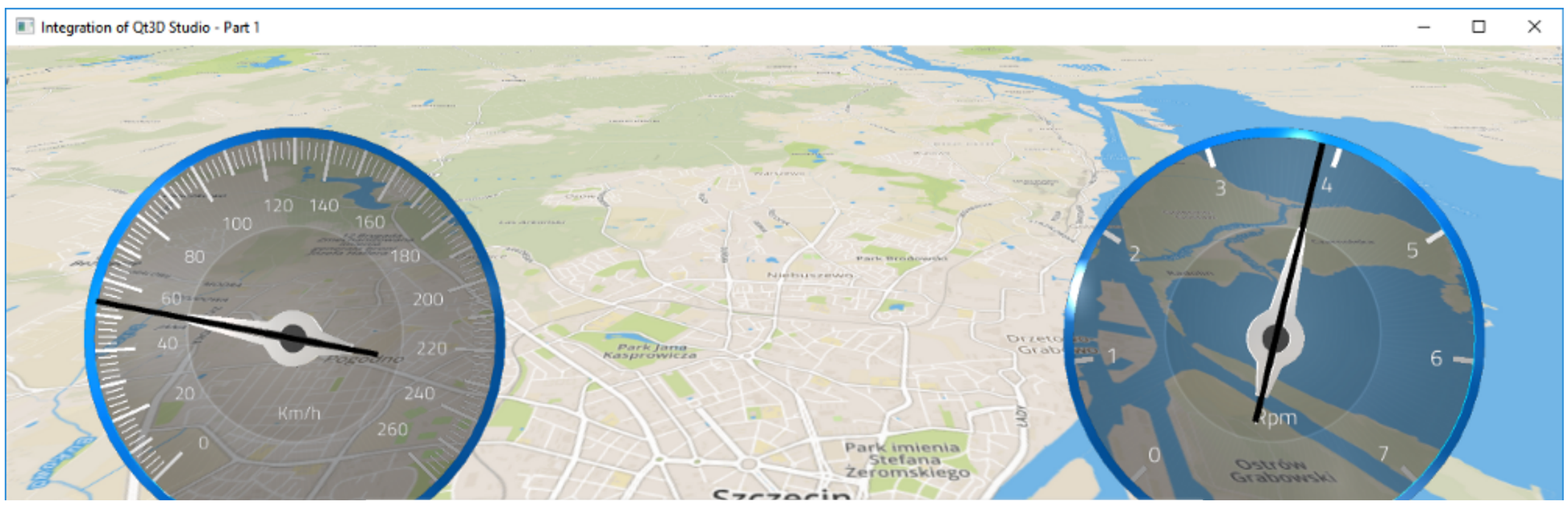
Oskar Lewandowski Follow
Oct 2, 2018 · 6 min read

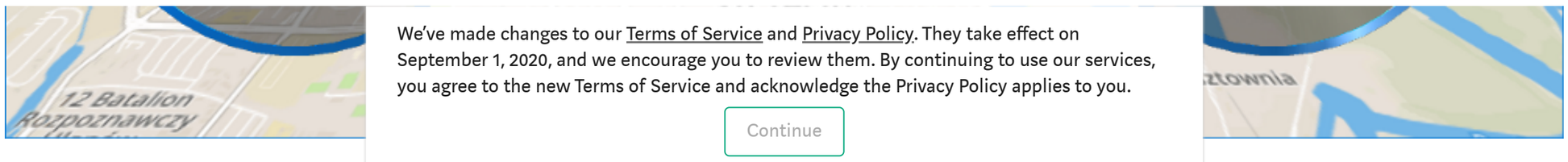
Introduction

Nowadays, most of the modern cars use digital instrument clusters, and that is why IT companies are pushed to develop more and better design tools, and whole toolchains. As always, the main goal is to produce high-quality product while reducing costs and enhancing the market appeal. We can find several dedicated design tools for the automotive industry like Rightware Kanzi, Altia 3D or Crank Storyboard Suite. But those software are not open source, provide only 30-day trial licenses. Qt 3D Studio is available for free and everyone can use it. It helps to learn and to create PoC or commercial product. It supports creating sharing knowledge community and increases number of trained engineers available on the IT market.

This tutorial walks step by step through the creation of QtQuick application containing presentation done in Qt 3D Studio and map provided by Mapbox plugin.

Going through this tutorial, you will be able to create Instrument Cluster looking like this:





Requirements

You need following prerequisites to complete the tutorial:

- Visual Studio 2015
- Qt 5.11.0
- Qt Creator (5.9 or newer)
- Qt 3D Studio 2.0
- Mapbox plugin (it is provided with Qt 5.9+ www.mapbox.com/qt/)
- OpenSSL 1.0.2 (or alternative one)
- mapbox.access_token — generated after creation of account on www.mapbox.com

Installing Qt tools

To install required Qt tools, you need to use one qt online installer. In this tutorial, Windows-x86–3.0.5 version was used (you can get it from here: https://download.qt.io/archive/online_installers/3.0/qt-unified-Windows-x86-3.0.5-online.exe).

Setting up Mapbox

Mapbox uses SSL protocol, thus we need to install openSSL

- Download OpenSSL 1.0.2 (or newer) and install (Link)
- Find ssleay32.dll, libeay32.dll (sometimes libssl32.dll is required)
- Copy to your folder: C:\Qt\5.11.0\msvc2015_64\bin or to the folder containing your binary file

If dlls are mismatched after project compilation, the following error will occur in the console:

Error Creating SSL Context() then you need to play with openSSL version (try newer or older one)

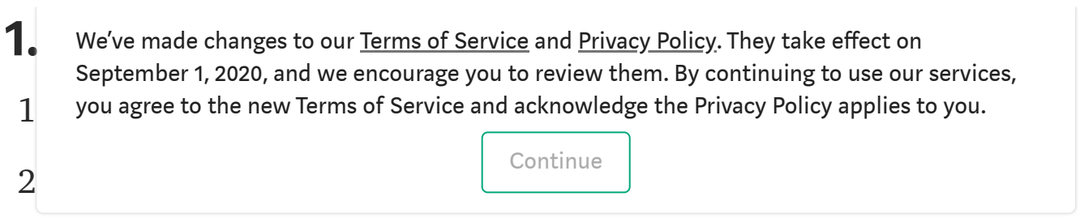
Issues with Windows version

For some reason, **mapboxgl** example provided on the website does not work for me on Windows platform. It worked when we provide account information and we switch plugin version to **mapbox**.

Even if you achieve running mapbox, it won't load styles (which is weird because before I wrote this tutorial, styles worked for me under Windows platform). Until Mapbox will not fix the problem with handling custom styles under Windows environment, I would recommend to use Linux version (in the meantime, I sent problem report to mapbox team). On the bright side, mapbox offers many cool stuff, such as: POI, 3D buildings, custom styles, or it can be easily integrated with other Qt plugin OSM (Open Street Map). It is exciting how many possibilities mapbox gives to designers and developers.

Tutorial chapters

1. Create QtQuick application
2. Integrate 3D Studio presentation
3. Add mapbox in background

1. 

1
2
3. Select Qt Quick Application — Empty
4. Name: Qt3DStudio_Integration
5. Leave Build System as it is
6. Minimal required Qt Ver.: 5.10 or higher
7. Deselect "Use Qt Virtual Keyboard" (we don't need it)
8. Select kit: Desktop Qt 5.10 MSVC2015 64bit (or another version if you installed different VS)
9. Next → Finish

2. Integrate 3D Studio presentation

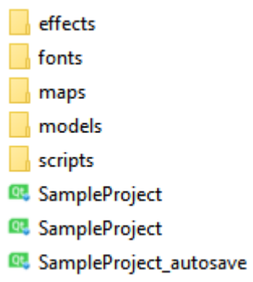
This tutorial is based on graphic assets provided by The Qt Company in examples* (Qt3DStudio\examples\studio3d\SampleProject.uip) and slightly modified for our purpose. Download it here: https://goo.gl/5wKbLx

Qt is a framework that has ambitions to close the gap between designers and developers. Usual (simplified) workflow goes as follows:

1. Designer creates visuals using 2D/3D design tools
2. Designer imports the visuals to Qt
3. Developer integrates and deploys to target device for tests

In this tutorial, it is assumed that designer prepared models/textures of gauges, imported them into new Qt 3D Studio project, used them in the presentation, added layers, animations, lights, and even set up specified kind of antialiasing that ought to be used by layers. Everything is ready for Developer to start integrating every part.

The output from Qt 3D Studio looks like below:



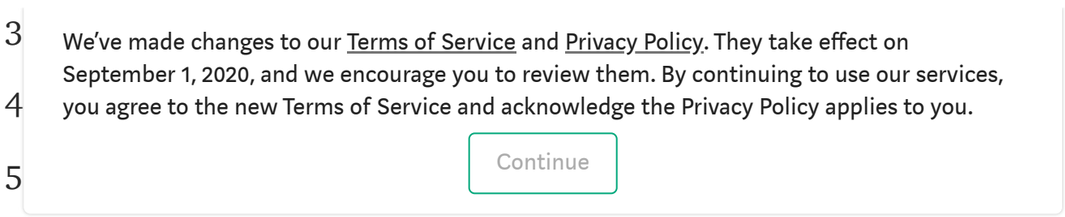
All files shall be placed inside folder named "Cluster".

First of all, we need to modify **main.cpp** code. The problem with generated version refers to the use of QQuickView with the different surface format as default which can lead to flickering. It should look like this:

```
#include <QApplication>
#include <QQuickView>
#include <QSurfaceFormat>
#include <q3dsruntimeglobal.h>

int main(int argc, char *argv[])
{
    qputenv("QSG_INFO", "1");
    QApplication app(argc, argv);

    QSurfaceFormat::setDefaultFormat(Q3DS::surfaceFormat());

    QQuickView viewer;
    viewer.setSource(QUrl("qrc:/main.qml"));

    viewer.setTitle(QStringLiteral("Integration of Qt3D Studio - Part 1"));
    viewer.setResizeMode(QQuickView::SizeRootObjectToView);
    viewer.resize(1280, 480);
    viewer.show();

    return app.exec();
}
```

Create a resource file:

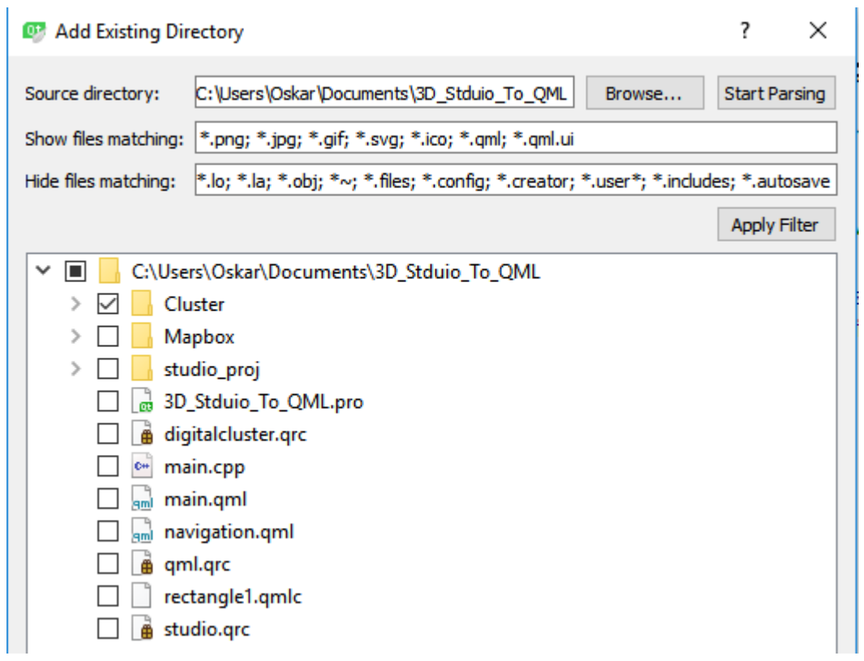
1. Right click on Resources
2. Add new



6. Next → Finish

Now import resources with Qt Studio presentation:

1. Right click on digitalcluster.qrc
2. Add existing directory
3. Select Cluster
4. Unselect main.cpp



To load the presentation to the application, we need to import Qt Studio module. Jump to **main.qml** file and add imports:

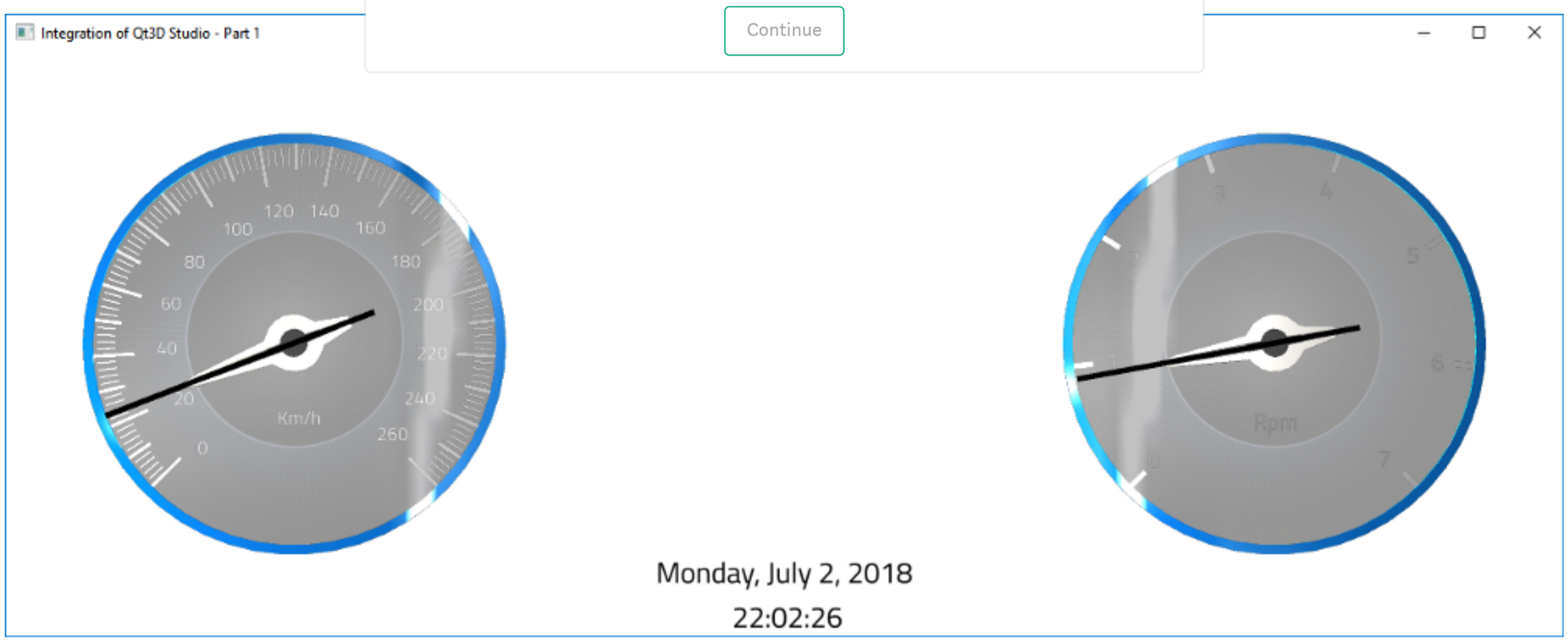
```
import QtStudio3D 2.0
```

A source code responsible for importing the presentation:

```
import QtQuick 2.11
import QtQuick.Window 2.11
import QtStudio3D 2.0
import QtLocation 5.9
import QtPositioning 5.8


Rectangle {
    id: root
    color: "lightGrey"

    Navigation {
        id: navi
        anchors.centerIn: parent

        width: parent.width
        height: parent.height
    }

    Studio3D {
        id: s3d
        focus: true
        anchors.margins: 0
        anchors.fill: parent

        Presentation {
            id: s3dpres
            source: "qrc:/Cluster/SampleProject.uip"

            SceneElement {
                id: scene
                elementPath: "Scene"
                currentSlideIndex: 2
            }
        }

        onRunningChanged: {
            console.log("Presentation ready!");
        }
    }
}
```

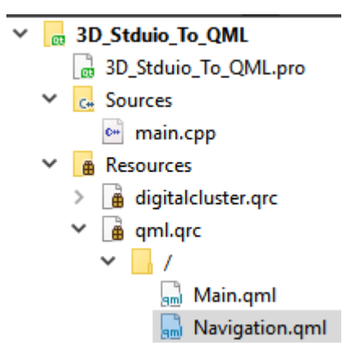
On top of the application, we have the rectangle and inside of it we create Studio3D object. It's responsible for importing the presentation.

- ***source:*** — is a relative path to SampleProject.uip
- ***elementPath*: "Scene"** — points to the element from presentation; until it has a unique name, no problems will occur
- ***currentSlideIndex*: 1** — says which slide will be active at the start



3. Add mapbox in background

Create new qml file in qml.qrc and name it Navigation.qml. Project structure shall look as below:



To start working with Mapbox we need to import two modules:

```
import QtLocation 5.6

import QtPositioning 5.6
```

Navigation.qml is a simple rectangle with plugin and map. First, declare Plugin:

```
import QtQuick 2.0
import QtLocation 5.6
import QtPositioning 5.6

Plugin {
        id: mapboxglPlugin
        name: "mapboxgl"

        PluginParameter {
            name: "mapboxgl.access_token"
            value: "pk.eyJ1Ijoib2xld2FuZG93c2tpIiwiYSI6ImNqajJ6OGRkNzE0eGozdm10aW
        }

        PluginParameter {
            name: "mapboxgl.mapping.additional_style_urls"
            value: "mapbox://styles/olewandowski/cjj32vya7306m2ro5ukm1w479"
        }
    }
```

id: boxglPlugin — determines which plugin shall be used

PluginParameter:

- **mapbox.access_token** — is your unique token generated after creating own account on: Sign Mapbox
- **Mapboxgl.mapping.additional_style_urls** — it is optional and points to map style provided by mapbox studio
- **Mapboxgl.mapping.cache.directory** — absolute path to map tile cache directory used as network disk cache

Mapbox Studio is a web application:

We can continue and add Map element:

```
25      Map {
26          anchors.fill: parent
27          plugin: mapboxglPlugin
28          center: QtPositioning.coordinate(53.431268, 14.568822);
29          zoomLevel: 14
30      }
31  }
```

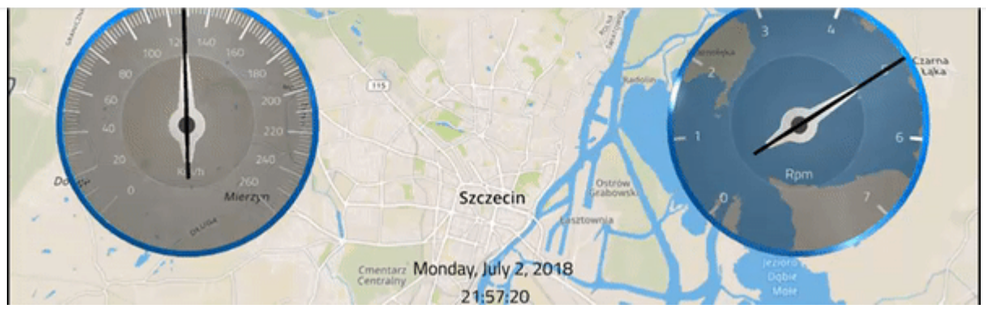
center: QtPositioning.coordinate(53.445727, 14.551184) — sets map coordinates

- **zoomLevel** — manipulates zoom level
- **bearing** — can rotate map
- **tilt** — sets angle of view on the map

To make the application more dynamic, add two variables, two functions, and timer:

```
31
32      property real naviTilt: 0
33      property real zoom: 12
34
35          function tiltIn()
36          {
37              if(naviTilt<60)
38                  naviTilt+=0.05
39          }
40          function zoomIn()
41          {
42              if(zoom < 14)
43                  zoom+=0.001
44          }
45          Timer {
46              id: loadRouteTimer
47              interval: 10
48              repeat: true
49              running: true
50              triggeredOnStart: true
51              onTriggered: {
52                  tiltIn()
53                  zoomIn()
54              }
55          }
56  }
```

The whole project with files will be zipped in the attachment, so the analysis of the source code will be easier.



Summary

We slightly touch Qt 3D Studio. This time, it was only simple presentation with animated gauges. Next time, I will extend functionality to handle data input to be able to set own speed and rpm values from qml. Also, there can be added sub presentation to handle additional screens, such as radio or settings.

Next episode: Part II — will cover the data input topic to manipulate needles rotation and show other useful information. Or even more.

PS. Qt Company has announced recently Qt Design Studio to build fluent workflow between developer and designer. It might be interesting to investigate: (http://blog.qt.io/blog/2018/06/26/qt-design-studio-the-new-age-of-ui-development/)

Here you can find part II:

https://medium.com/siili-automotive/qt-3d-studio-digital-cluster-for-a-car-part-ii-353aaecaaf9f

References

https://mapbox.com/qt/

https://doc.qt.io/qt3dstudio/index.html

http://blog.qt.io/blog/2018/06/27/introducing-qt-automotive-suite-5-11

Mapbox Qt Qt Quick Automotive Qml