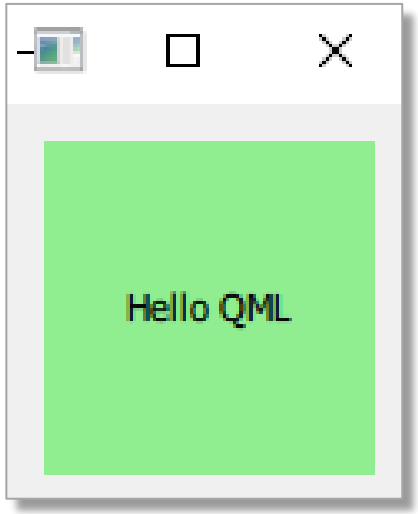
QtQuick и C++

Объединим возможности двух технологий

Использование QML с C++

Класс ***QQuickWidget***, наследник *QWidget* - интегрирован в *QML* и предоставляет среду для показа и визуализации *QML*-элементов. Экземпляр ***QQuickWidget*** следует использовать как обычный виджет в коде на *Qt/C++* для визуализации *QML*-элементов. Этот виджет (*QQuickWidget*) может быть расположен в области другого виджета, используя класс размещения или заданную позицию. ***QQuickWidget*** расположен в отдельном модуле **quickwidgets**, который необходимо включить в проектный файл.

Опция включения модулей проектного файла будет выглядеть следующим образом:
QT += quick qml widgets quickwidgets



Использование QML в C++

В конструкторе класса ***MyWidget*** создается объект класса ***QQuickWidget ,*** в конструкторе которого загружается файл «**main.qml**» из ресурса, в котором находится исходный текст *qml*-программы. Далее объект QQuickWidget размещается на поверхности виджета класса ***MyWidget*** при помощи класса размещения *QVBoxLayout*.

MyWidget.cpp

```
MyWidget::MyWidget(QWidget* pwgt/*=0*/) : QWidget(pwgt)
{
   QQuickWindow::setGraphicsApi(QSGRendererInterface::OpenGL);
   QQuickWidget* pv = new QQuickWidget(QUrl("qrc:/main.qml") );
   QVBoxLayout* pvbx = new QVBoxLayout;
   pvbx->addWidget(pv);
   setLayout(pvbx);
}
```

main.qml

```
import QtQuick 2.15

   Rectangle {
      color: "lightgreen"
      width: 100
      height: 100
      Text {
         objectName: "text"
         anchors.centerIn: parent
         text: "Hello QML"
         function setFontSize(newSize) {
                font.pixelSize = newSize
                return font.family + " Size=" + newSize
          }
       }
  }
```

Взаимодействие из C++ со св-ми QML-элементов

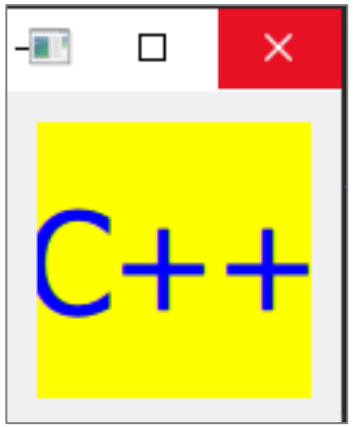
```
MyWidget::MyWidget(QWidget* pwgt/*=0*/) : QWidget(pwgt) {
    QQuickWidget* pv = new QQuickWidget(QUrl("qrc:/main.qml"));
    QVBoxLayout* pvbx = new QVBoxLayout;
    pvbx->addWidget(pv);
    setLayout(pvbx);
     // поиск св-ва «color»  и присвоение "yellow« у узлового элемента Rectangle
    QQuickItem* pqiRoot = pv->rootObject();
    if(pqiRoot) {
         pqiRoot->setProperty ( "color", "yellow");
        // поиск элемента с  objectName == "text"  - дочерний от узлового - элемент Text
        QObject* pObjText = pqiRoot->findChild<QObject*>("text");
        if (pObjText) {
            pObjText->setProperty ( "text", "C++") ;   // замена значения у элемента «text»
            pObjText->setProperty ( "color", “blue"); // изменение его цвета
            QVariant varRet;
           // вызов функции элемента Text
            QMetaObject::invokeMethod(pObjText, "setFontSize",
                                      Q_RETURN_ARG(QVariant, varRet), Q_ARG(QVariant, 52) );
          qDebug () << varRet;
        }
    }
}
```

Аргумент для возврата

Аргумент для вход. парам-ра

После размещения QML-элемента внутри виджета выясним как можно взаимодействовать с его свойствами и вызывать функции QML-элемента из C++?

Все функции в *QML*-программе представлены в мета объектной информации и, благодаря ей, могут быть вызваны из C++.

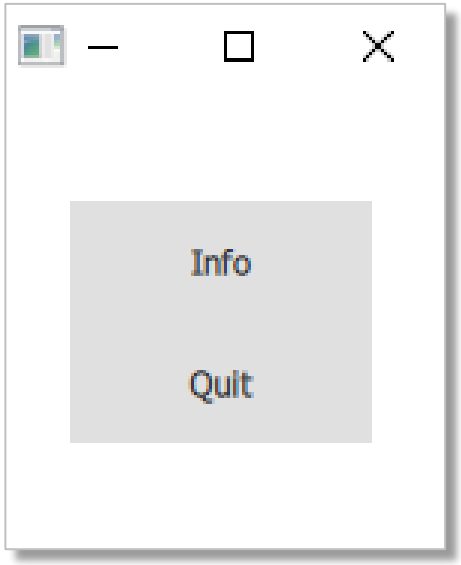


Изменение свойств QML-элементов из C++

Соединение QML-сигналов со слотами C++

Создадим окно с элементами *QML-кнопок* ***Info*** и ***Quit*** и соединим их сигналы со слотами C++ - объекта. Ранее мы использовали *QML*-элементы внутри виджетов. В этот раз мы полностью исключим их использование и уберем все ненужные модульные зависимости из проектного файла, оставив лишь только два модуля: ***QtQuick*** и ***QtQml***. Опция включения модулей нашего проектного файла будет выглядеть следующим образом:

QT += quick qml



Пример сигнал-слотового соединения

1.

«Класс C++ со слотами»
CppConnection.h

```
#pragma once
#include <QtCore>

class CppConnection : public QObject {
Q_OBJECT

 public:
    CppConnection(QObject* pobj 0) : QObject(pobj) {
     }
 public slots:
     void slotQuit () {
         qApp->quit ();
      }
     void slotInfo(const QString & str){
        qDebug() << str;
     }
};
```

2.

«Основная программа»
main.qml

```
import QtQuick 2.15
import QtQuick.Controls 2.2
import QtQuick.Window 2.15

Window {
    visible: true
    width: 150;  height: 150
    Column {
      anchors.centerIn: parent
      Button {
          signal infoClicked(string str)
          objectName: "InfoButton"
          text: "Info"
          onClicked: infoClicked ( "Information")
      }
     Button {
          signal quitClicked()
          objectName: "QuitButton"
          text: "Quit"
          onClicked: quitClicked()
     }
  }
}
```

Пример сигнал-слотового соединения

3.

«Основная программа C++»
main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlComponent>
#include "CppConnection.h"

int main(int argc, char** argv) {
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine eng;
    QQmlComponent comp(&eng, QUrl("qrc:/main.qml") );
    CppConnection cc;     // типичный C++ класс с описанием слотов
    QObject* pobj = comp.create(); // получаем указатель на узловой элемент в виде объекта QObject
                    // получаем указатели на элемент кнопки QuitButton через узловой элемент
    QObject* pcmdQuitButton = pobj->findChild<QObject*>("QuitButton");
    if (pcmdQuitButton) {
         QObject::connect(pcmdQuitButton, SIGNAL(quitClicked()), &cc, SLOT(slotQuit())) ;
   }
                   // получаем указатели на элемент кнопки InfoButton через узловой элемент
   QObject* pcmdInfoButton = pobj->findChild<QObject*>("InfoButton");
    if (pcmdInfoButton) {
        QObject::connect(pcmdInfoButton, SIGNAL(infoClicked(QString) ), &cc, SLOT(slotInfo(QString))) ;
    }
    return app.exec();
}
```

Вместо класса QQuickWidget

Использование компонентов языка C++ в QML

Именно благодаря этому направлению использования можно реализовать красивый пользовательский интерфейс с анимационными эффектами, а реализацию функциональных компонентов возложить на *C++.* При таком подходе задействуются самые выигрышные стороны обоих инструментов: и *QML*, и *C++.*

Для собственных наработок или модулей других разработчиков, базирующиеся на *C++* и *Qt* необходимо использовать класс контекста ***QQmlContext.***

В языке *QML* реализована возможность расширения при помощи C++. Благодаря ей можно осуществлять расширение этого языка новыми элементами из C++.

Если необходимо использовать уже существующие технологии *Qt*, такие как ***Qt3D***, ***QtCharts***, ***QtWebEngine***, а также прочие модули *Qt QML*, то для этого нужно просто воспользоваться директивой ***import***. Например, для *QtWebEngine*:

import QtWebEngine 1.5

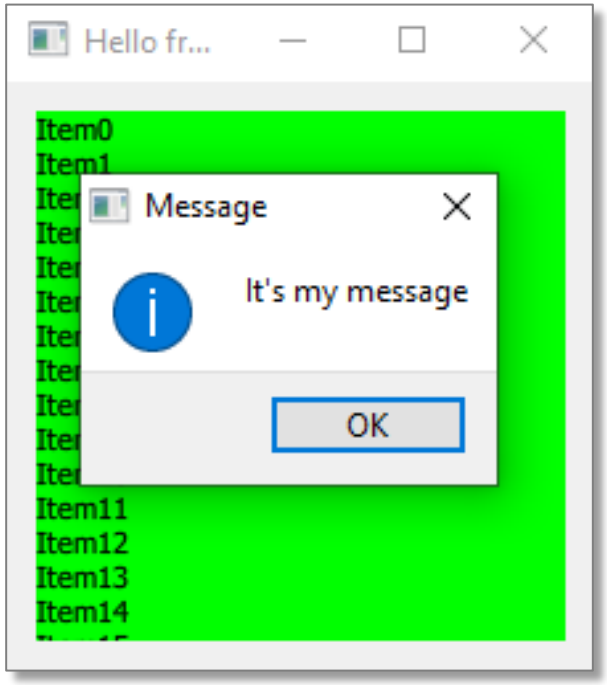
Использование компонентов языка C++ в QML

Чтобы получить доступ к объекту класса ***QQmlContext***, нужно из объекта класса ***QQuickWidget*** вызвать метод ***rootContext()***, который возвращает указатель на корневой контекст. Используя этот указатель, вы можете в дерево контекста ввести новые объекты классов, унаследованных от класса *QObject*. Публикация объектов в контексте осуществляется с помощью метода ***setContextProperty()***. Этот метод принимает два аргумента. Первый аргумент - это имя, под которым объект будет доступен в *QML*, второй аргумент - это адрес объект.
Если вы проделаете эту операцию, то свойства класса *QObject* станут свойствами *QML,* а слоты и методы, декларированные с помощью макроса *Q_ INVOKABLE*, станут методами, которые могут вызываться из вашего нового *QML*-элемента.

Класс ***QQuickWidget*** содержит также и объект класса ***QQmlEngine***, который предоставляет среду для *QML*-компонентов и является сердцевиной для исполнения *QML*-кода. Доступ к нему можно получить вызовом метода ***engine()*** .

Экспорт объектов и виджетов из C++ в QML

Механизм использования объектов библиотеки *Qt* в *QML*. *QML*-программа будет, помимо прямоугольной желтой области, содержать элемент представления списка и область мыши. При нажатии мышью на область, реализованную на *QML*, будет вызываться слот из виджета *MyWidget* и отображаться диалоговое окно с текстом: «***It's my message»***



Экспорт объектов …

«MyWidget.cpp»

```
MyWidget::MyWidget(QWidget* pwgt/*=0*/) : QWidget(pwgt) {
QQuickWindow::setGraphicsApi(QSGRendererInterface::OpenGL);
   QQuickWidget* pv = new QQuickWidget;
   pv->setSource(QUrl("qrc:/main.qml"));
   QVBoxLayout* pvbx = new QVBoxLayout;
   pvbx->addWidget(pv);
   setLayout(pvbx);
   QQmlContext* pcon = pv->rootContext();
   QStringList lst;
   for (int i = 0; i < 100; ++i) {
     lst << "Item" + QString::number(i);
   }
   QStringListModel* pmodel = new QStringListModel(this);
   pmodel ->setStringList(lst);
   pcon->setContextProperty("myModel", pmodel);
   pcon->setContextProperty("myText", "It's my text!");
   pcon->setContextProperty("myColor", QColor(Qt::yellow));
   pcon->setContextProperty("myWidget", this);
}
void MyWidget::slotDisplayDialog () {
    QMessageBox::information(0, "Message", "It's my message");
}
```

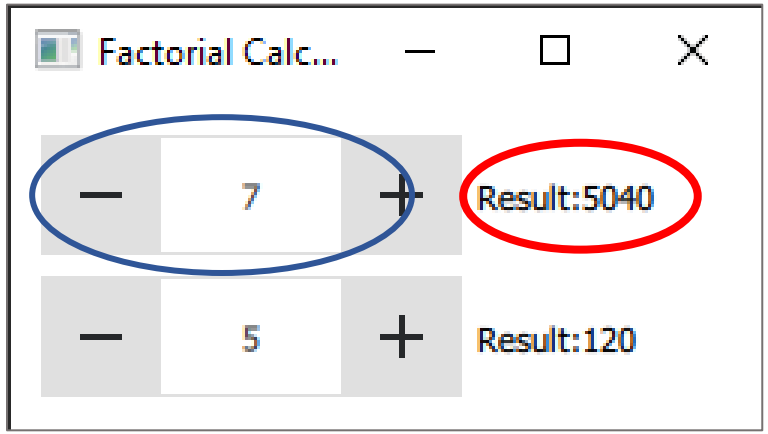
«main.qml»

```
import QtQuick 2.15
Rectangle {
    color: myColor
    width: 200
    height: 200
    Text {
       anchors.centerIn: parent
       text: myText
       ListView {
            anchors.fill:parent
            model: myModel
            delegate: Text {text: model.display}
       }
       MouseArea {
            anchors.fill: parent
            onPressed: { myWidget. setWindowTitle ( "Hello
                                from QML"); }
            myWidget.slotDisplayDialog();
       }
    }
}
```

Использование зарегистрированных объектов C++

Продемонстрируем возможность использования свойств ***Q_PROPERTY*** и метода, определенного как ***Q_INVOKABLE***.

Программа вычисляет значение факториала, исходя из значений, введенных в элементы счетчика, расположенного в левой части окна. Результаты вычисления отображаются в правой части окна.



Регистрация класса Calculation

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "Calculation.h "

int main(int argc, char** argv) {
    QGuiApplication app(argc, argv);
    qmlRegisterType<Calculation>("com.myinc.Calculation", 1, 0,
                                 "Calculation");
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();
}
```

Идентификатор модуля в QML-программе

Версия

Имя элемента

Использование зарегистрированных объектов C++

«Calculation.h»

```
class Calculation : public QObject {
Q_OBJECT
private:
    Q_PROPERTY(qulonglong input WRITE setInputValue READ inputValue NOTIFY inputValueChanged )
    Q_PROPERTY(qulonglong result READ resultValue  NOTIFY resultValueChanged )
    qulonglong m_nInput;
    qulonglong m_nResult;
public:
    Calculation(QObject* pobj = 0);
    Q_INVOKABLE qulonglong factorial(const qulonglong & n);  // рекурсивные вычисления факториала
     qulonglong inputValue ( ) const;      // чтение свойства input
     void setInputValue(const qulonglong &);   // запись свойства input
     qulonglong resultValue ( ) const;      // чтение свойства result
signals:
      void inputValueChanged (qulonglong);
      void resultValueChanged(qulonglong);
}
```

Использование зарегистрированных объектов C++

«Calculation.cpp»

```
#include "Calculation.h"

Calculation::Calculation(QObject* pobj): QObject(pobj),
                                m_nInput(0) , m_nResult(1) {
 }
qulonglong Calculation::factorial(const qulonglong & n) {
    return n ? (n * factorial (n - 1)) : 1;
}
qulonglong Calculation::inputValue() const {
   return m_nInput;
}
qulonglong Calculation::resultValue() const {
   return m_nResult;
}
void Calculation::setInputValue(const qulonglong & n) {
   m_nInput = n;
   m_nResult = factorial(m_nInput);
   emit inputValueChanged(m_nInput);
   emit resultValueChanged(m_nResult);
}
```

Использование зарегистрированных объектов C++

«main.qml»

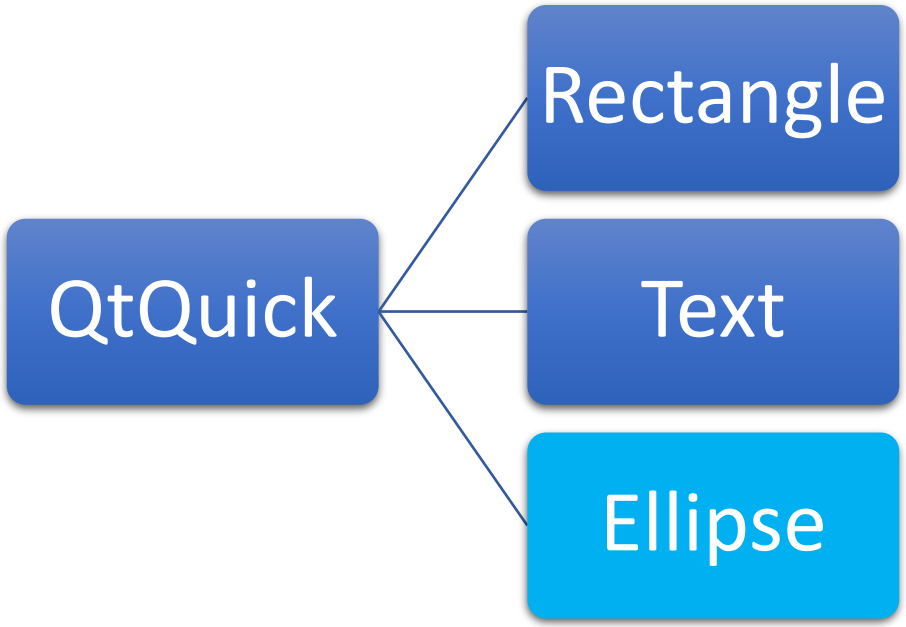
```
ApplicationWindow {
    title: "Factorial Calculation"
    width: 250; height: 40;
    visible: true
    Calculation {
        input: sbx.value
        onResultValueChanged: txt.text ="Result:" + result
    }
    RowLayout {
        SpinBox {
            id: sbx;
            value: 0
        }
        Text {
            id: txt
        }
    }
}
```

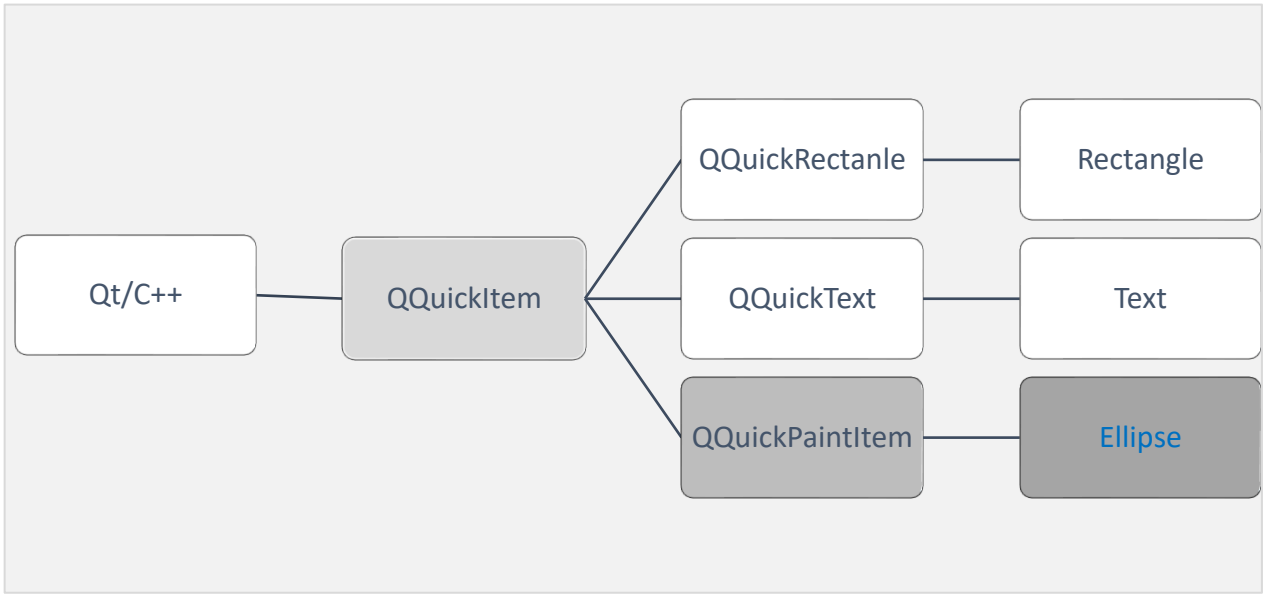
«main.qml»

```
import QtQuick 2.15
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.3
import com.myinc.Calculation 1.0
ApplicationWindow {
    title: "Factorial Calculation"
    width: 250 ; height: 80 ;  visible: true
    Calculation {
        id: calc }
    ColumnLayout  {
        anchors.fill: parent
        RowLayout { // 1. call of an invokable method
            SpinBox {
                id: sbx ;  value: 0 }
            Text {
                text: "Result:" + calc.factorial(sbx.value) }
        }
        RowLayout { // 2. using of the properties
            SpinBox {
                value: 0
                onValueChanged: calc.input = value }
            Text {
                text: "Result:" + calc.result
            }
        }
    }
}
```

Реализация визуальных элементов QML на C++

Библиотека базовых визуальных элементов QML (Rectangle, Text, …) может быть расширена!





Реализация визуальных элементов

«Ellipse.h»

```
#pragma once
#include <QQuickPainteditem>
class QPainter;

class Ellipse : public QQuickPaintedItem {
Q_OBJECT
private:
    Q_PROPERTY(QColor color WRITE setColorValue  READ colorValue)
    QColor m_color;
public:
     Ellipse(QQuickitem* pqi = 0);
     void paint(QPainter* ppainter);
     QColor colorValue ( ) const;
     void setColorValue(const QColor&);
};
```

«Ellipse.cpp»

```
#include <QPainter>
#include "Ellipse.h"

Ellipse::Ellipse(QQuickItem* pqi /*=0*/) : QQuickPaintedItem(pqi)
                                          , m_color (Qt::black) { }

void Ellipse::paint(QPainter* ppainter) {
    ppainter->setRenderHint(QPainter::Antialiasing, true);
    ppainter->setBrush(QBrush(colorValue()));
    ppainter->setPen(Qt::NoPen);
    ppainter->drawEllipse(boundingRect());
}
QColor Ellipse::colorValue() const {
   return m_color;
}
void Ellipse::setColorValue(const QColor& col) {
   m_color = col;
}
```

Регистрация класса Ellipse

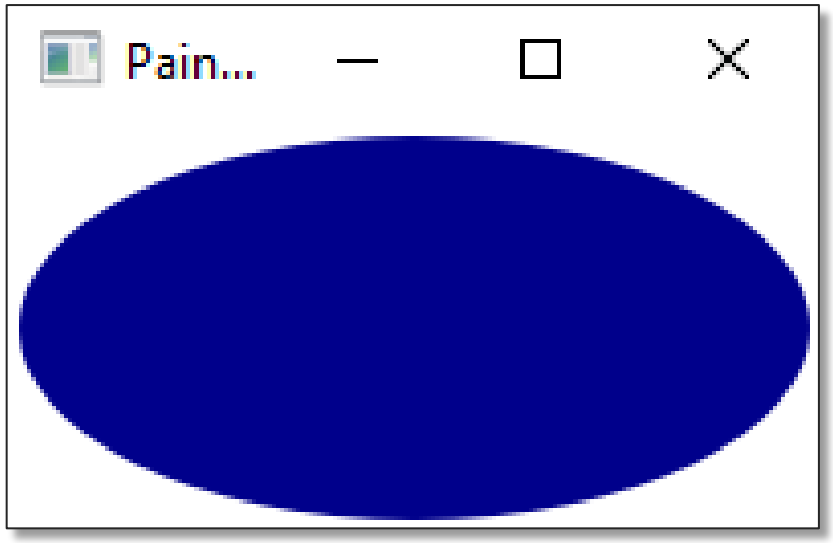
«main.cpp»

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "Ellipse.h"

int main(int argc, char** argv) {
    QGuiApplication app(argc, argv);
    qmlRegisterType <Ellipse>("com.myinc.Ellipse", 1, 0, "Ellipse");
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();
}
```

Использование элемента Ellipse в QML-программе

«main.qml»

```
import QtQuick 2.15
import QtQuick.Controls 2.2
import QtQuick.Window 2.2
import com.myinc.Ellipse 1.0

Window {
    title: "PaintElement"
    visible: true
    width: 200
    height: 100
    Ellipse {
      anchors.fill: parent
      color: “blue“
     }
}
```



Класс QQuickImageProvider

Для операций с растровыми изображениями можно использовать и другой подход, воспользовавшись классом ***QQuickimageProvider.***

В QML элемент от этого класса будет представлен как обычная ссылка на файл.

eng -объект предоставляет среду исполнения ***QML***-кода с элементом основного окна приложения ***ApplicationWindow***.
Метода **addimageProvider ()** добавляет объект класса ***ImageProvider,*** который унаследован от ***QQuickimageProvider*** и снабжен алгоритмом обработки изображения.
Метод **load ()** производит загрузку ***QML***-файла из ресурса.

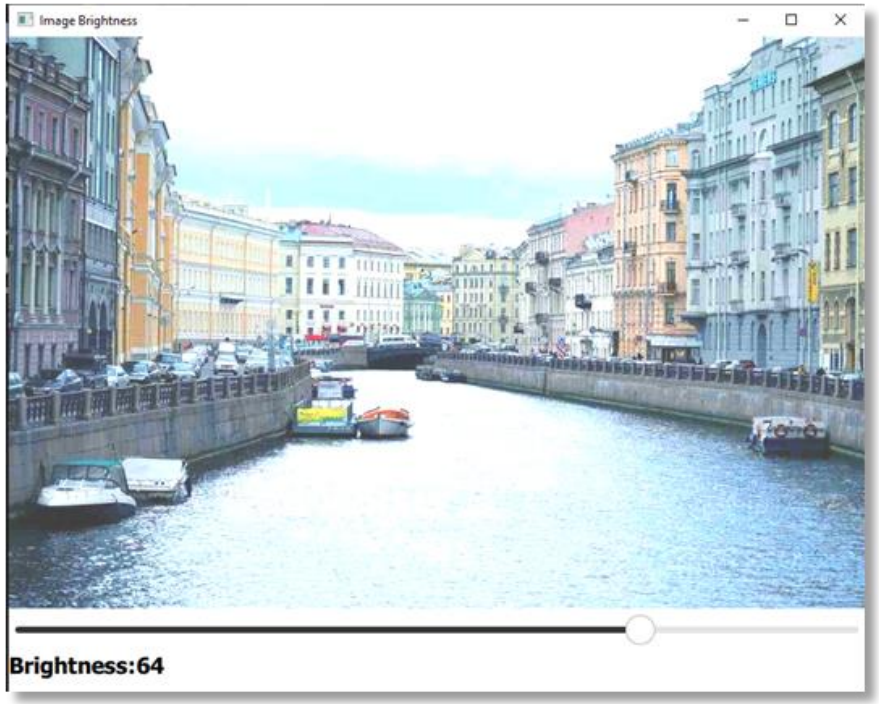
```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "ImageProvider.h"

int main(int argc, char *argv[]) {
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine eng;
    eng.addImageProvider(QLatin1String("brightness"), new ImageProvider);
    eng.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();
}
```

*В примере элементы управления и отображения растрового изображения реализованы на QML, а алгоритм изменения яркости - на C++.

Создание класса ImageProvider

«ImageProvider.h»

```
#pragma once
#include <QObject>
#include <QImage>
#include <QQuickImageProvider>

class ImageProvider : publlic QQuickimageProvider {
   private:
      QImage brightness(const QImage& imgOrig, int n);
   public:
      ImageProvider(); // конструктор
      QImage requestImage(const QString&, QSize*,
                                          const QSize&);
} ;
```



Метод **requestImage ()** является связующим звеном между QML и C++

ImageProvider.cpp

```
ImageProvider::ImageProvider():
QQuickImageProvider(QQuickimageProvider::Image)
{ }
```

```
QImage ImageProvider::requestImage(const QString& strid, QSize*
ps, const QSize&
/*requestedSize*/) {

    QStringList lst = strid.split(";");
    bool bOk = false;
    int nBrightness = lst.last().toInt(&bOk);
    QImage img = brightness(QImage(":/" + lst.first()), nBrightness);
    if (ps) {
       *ps = img.size();
    }
   return img;
}
```

```
QImage ImageProvider::brightness(const QImage& imgOrig, int n) {

  QImage imgTemp = imgOrig;
  qint32 nHeight = imgTemp.height();
  qint32 nWidth =  imgTemp.width();

   for (qint32 y = 0; y < nHeight; ++y) {
      QRgb* tempLine = reinterpret_cast<QRgb*>(imgTemp.scanLine(y) );
      for (qint32 x = 0; x < nWidth; ++x) {
           int r = qRed(*tempLine) + n;
           int g = qGreen(*tempLine) + n;
           int b = qBlue(*tempLine) + n;
           int a = qAlpha(*tempLine);
          *tempLine++ = qRgba (r > 255 ? 255 :  r < 0 ? 0 : r,
                               g > 255 ? 255 : g < 0 ? 0 : g,
                               b > 255 ? 255 : b < 0 ? 0 : b,
                               a );
      }
   }
 return imgTemp;
}
```

Использование класса ImageProvider

«main.qml»

```
ApplicationWindow {
    title: qsTr("Image Brightness")
    width: controls.width;  height: controls.height
    visible: true
    Column {
        id: controls
        Image {
            id: img
            source: "image://brightness/spb.png;" + sld.brightnessValue
        }
        Slider {
            id: sld
            width: img.width; value: 0.75;  stepSize: 0.01
            property int brightnessValue: (value * 255 - 127)
        }
        Text {
            width: img.width
            text: "<h1>Brightness:" + sld.brightnessValue + "</h1>“
        }
    }
}
```

*Элемент ***Image*** показывает растровое изображение. Обратите внимание, что **мы получаем изображение, которое вычисляется с помощью ранее реализованного класса *ImageProvider*, обычной строкой в свойстве *source***. В этой строке сразу после указания типа ссылки **image** мы приводим ссылку **brightness** на наш объект создания растровых изображений с изменением яркости. Далее указываем имя файла **“spb.png”**, который у нас содержится в ресурсе, и после знака **;** - значение яркости. Значение яркости мы берем из элемента ползунка с идентификатором ***sld***.

Домашка #13

Создание гибридного приложения.
Тему выбираем самостоятельно. Возможно игра, тест, обучающее приложение, и т.д.
GUI – реализуется в ***Qt Quick***, логика – в ***C++/Qt***.