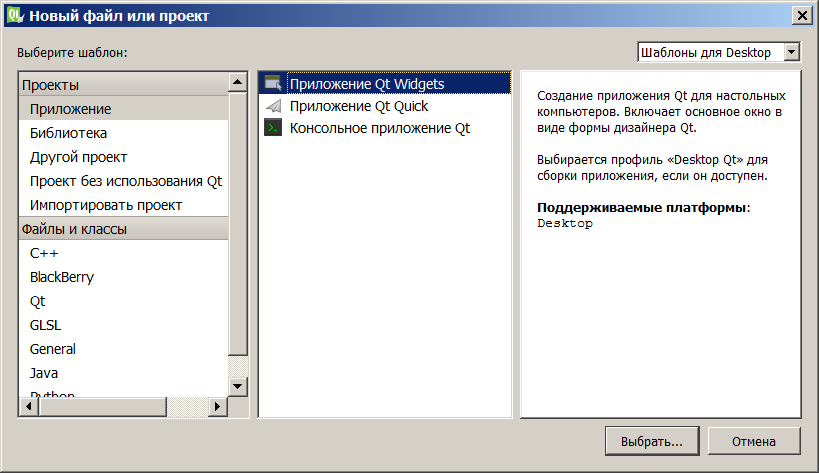
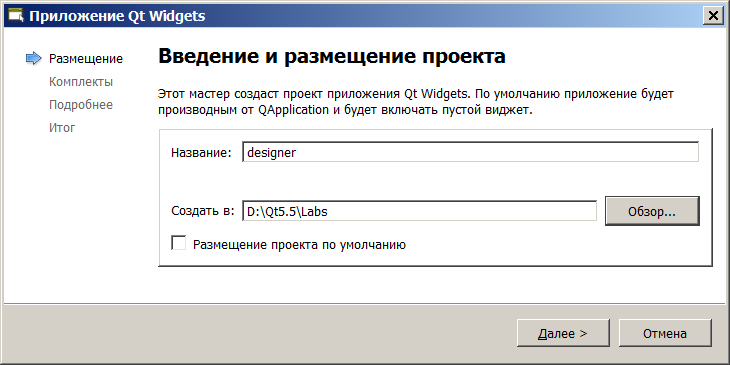
**Практика 4.**

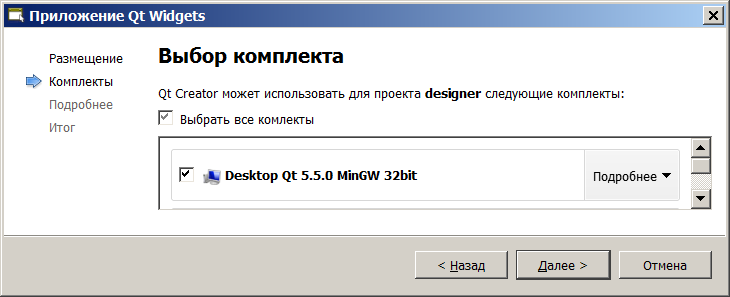
# Учимся использовать QtDesigner. QMainWindow. Ui. CentralWidget. Партнеры – buddy. Редактор сигналов/слотов. QAction. Редактор действий. Пользовательский виджет. Меню. ToolBar. Статусная строка. Ресурсы приложения. MessageBox

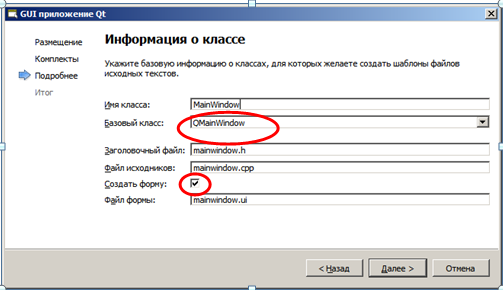
Разработаем с помощью дизайнера простейшее приложение масштабирования картинки. Масштаб будем задавать разными элементами управления, которые должны быть синхронизированы. При этом автоматизируем бОльшую часть действий из лабораторной работы №3 по настройке внешнего вида и свойств виджетов, а также их взаимодействия посредством QtDesigner-а.

## Заготовка приложения









Дизайнер создает файлы, предназначенные как для ручного редактирования, так и для вставки заготовок с его же (дизайнера) помощью:

* Файлы, предназначенные для ручного редактирования:
  + исходный файл main.cpp с заготовкой функции main
  + класс поддержки главного окна MainWindow (интерфейс – mainwindow.h + реализация - mainwindow.cpp)
* файл проекта –«руками» стоит редактировать только в случае уверенного владения директивами проекта
* заготовку для формы главного окна – mainwindow.ui – файл предназначен для редактирования **только** в графическом режиме
* файл ui\_mainwindow.h:
  + в IDE программисту он «не виден»,
  + редактировать вручную **не имеет смысла**. В этот файл изменения вносит **дизайнер**!
  + Но полезно туда заглядывать, чтобы понимать: в какой код превращается графическое редактирование.

В файле main.cpp – стандартная заготовка приложения:

* создание главного окна посредством сгенерированного дизайнером класса MainWindow
* запуск цикла обработки событий

#include"mainwindow.h"

#include<QApplication>

int main(int argc,char\*argv[])

{

QApplicationa(argc,argv);

MainWindow w;

w.show();

return a.exec();

}

Можно протестировать заготовку. Вы увидите пустую заготовку главного окна.

## Класс Ui\_MainWindow

Ui – “User interface”

* Этот класс создает дизайнер.
* Только дизайнеру позволено этот класс модифицировать.
* Все изменения, которые мы с помощью графического редактора вносим в форму, отображаются в коде данного класса.

На момент создания заготовки приложения дизайнером:

class Ui\_MainWindow

{

public:

QMenuBar \*menuBar;//меню

QToolBar \*mainToolBar; //tool-бар

QWidget \*centralWidget;//дочерний виджет главного окна, который занимает всю клиентскую область

QStatusBar \*statusBar;//статусная строка

void setupUi(QMainWindow\*MainWindow)//создание и настройка «по умолчанию» атрибутов главного окна

{

if(MainWindow->objectName().isEmpty())

MainWindow->setObjectName(QStringLiteral("MainWindow"));

MainWindow->resize(400,300);

menuBar=new QMenuBar(MainWindow);

menuBar->setObjectName(QStringLiteral("menuBar"));

MainWindow->setMenuBar(menuBar);

mainToolBar=new QToolBar(MainWindow);

mainToolBar->setObjectName(QStringLiteral("mainToolBar"));

MainWindow->addToolBar(mainToolBar);

centralWidget=new QWidget(MainWindow);

centralWidget->setObjectName(QStringLiteral("centralWidget"));

MainWindow->setCentralWidget(centralWidget);

statusBar=new QStatusBar(MainWindow);

statusBar->setObjectName(QStringLiteral("statusBar"));

MainWindow->setStatusBar(statusBar);

retranslateUi(MainWindow);//в этой функции сгруппированы вызовы всех методов, которые «завязаны» на перевод строк

QMetaObject::**connectSlotsByName**(MainWindow);автоматически соединяет сигналы всех дочерних объектов MainWindow со слотами самого MainWindow, имеющими предопределенную сигнатуру:  
void on\_<имя\_объекта>\_<имя\_сигнала>(параметры  
\_сигнала)

}//setupUi

void retranslateUi(QMainWindow\*MainWindow)

{

MainWindow->setWindowTitle(QApplication::translate("MainWindow",

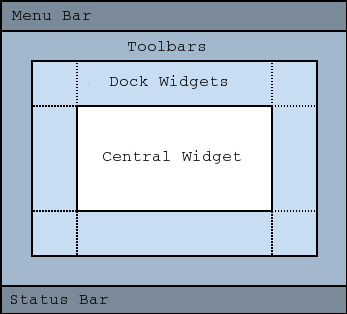
"MainWindow",0));

}//retranslateUi

};

Для поддержки функциональности и вида главного окна приложения библиотека Qt предоставляет класс QMainWindow. Класс QMainWindow обладает собственной компоновкой виджетов (layout-ом), в который пользователь может добавлять объекты QToolBar, QMenuBar, QStatusBar, а также объекты QDockWidget. Специфика этого layout-а заключается в том, что его центральную часть (клиентскую область) может занимать widget любого типа.

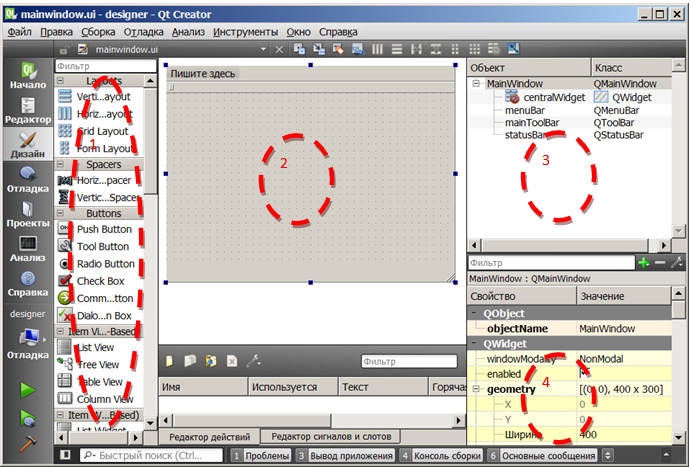
Замечание : не поддерживается создание главного окна приложения посредством класса QMainWindow **без** центрального виджета.



Начинаем наполнять заготовку содержимым.

## Изменяем свойства главного окна с помощью дизайнера

Перейдите в режим дизайнера – для этого достаточно кликнуть «Дизайн» или Формы/mainwindow.ui в панели «Проекты».



Обратите внимание:

1 – панель инструментов (можно отметить требуемый элемент управления и «перетащить его на форму»)

2 – собственно форма (так будет выглядеть Ваше главное окно)

3 – описание зависимостей между элементами управления и имена соответствующих объектов

4 – свойства выделенного элемента управления

### Заголовок окна

Сделайте активной всю форму. Редактируем свойство – WindowTitle.

Если Вы задали строчку по-русски (например, «Дизайнер»), то в результате в файле ui\_mainwindow.h в методе:

void **retranslateUi**(QMainWindow\*MainWindow)

{

MainWindow->setWindowTitle(QApplication::translate("MainWindow", "\320\224\320\270\320\267\320\260\320\271\320\275\320\265\321\200", 0));

}//retranslateUi

, где второй параметр, задаваемый строковым литералом, состоит из escape-последовательностей для Unicode-кодов русских букв.

Замечание: изменения вносятся после компоновки, а не после сохранения файла!

Но! Гораздо лучше задать ее по-английски **в дизайнере** и использовать возможность перевода. В файле ui\_mainwindow.h:

void retranslateUi(QMainWindow\*MainWindow)

{

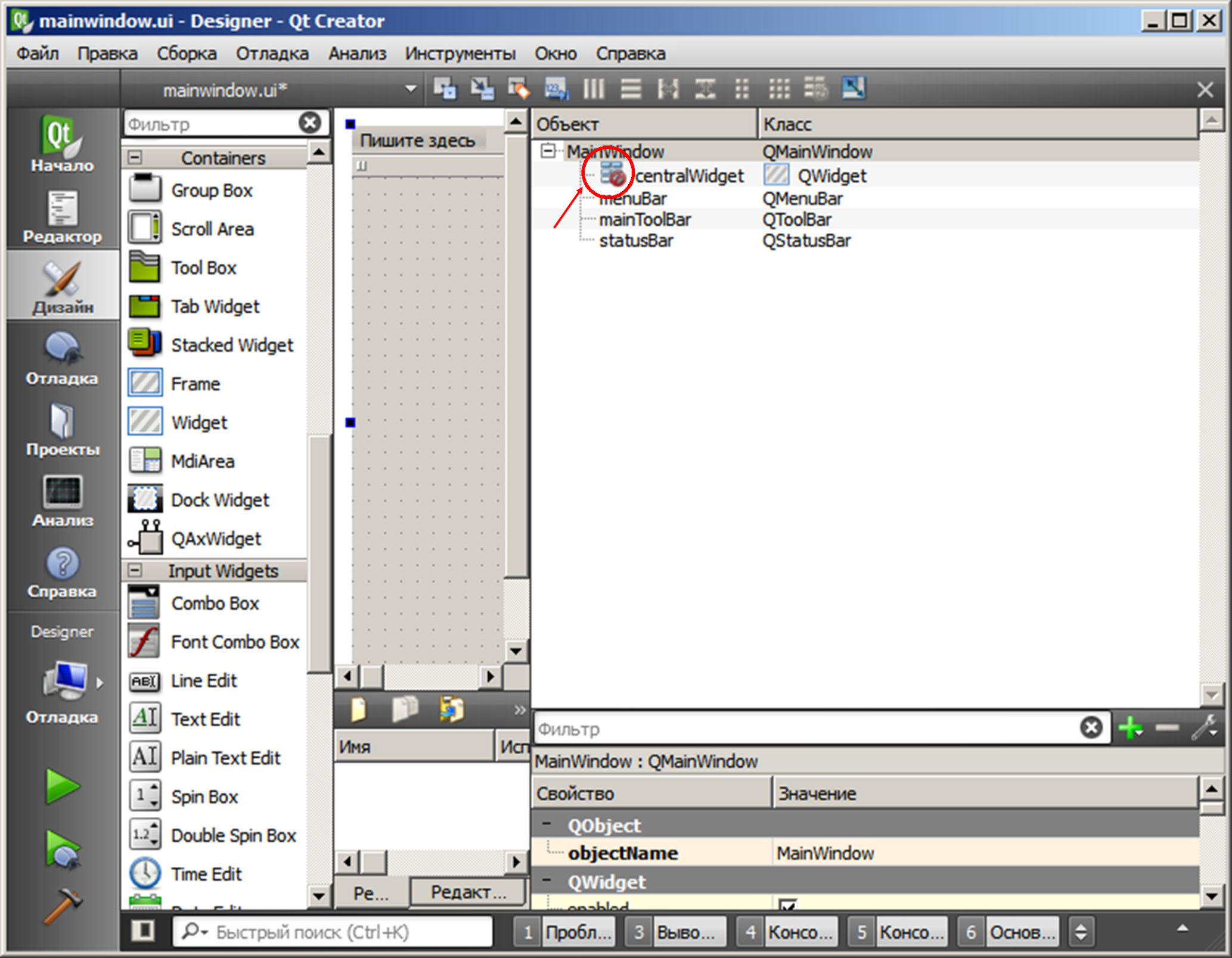
MainWindow-> setWindowTitle (QApplication::translate ("MainWindow",

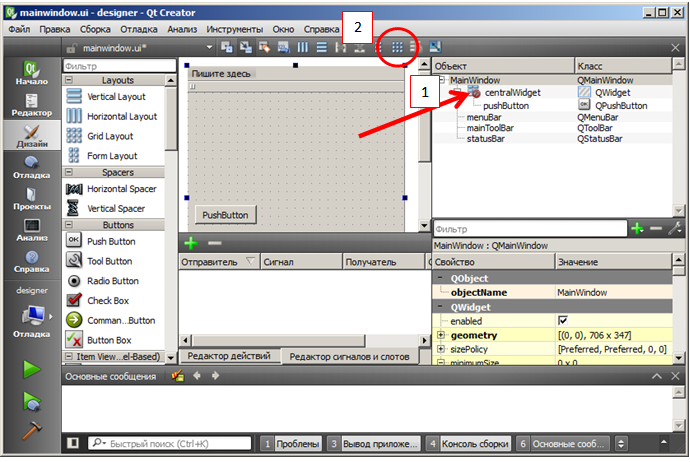
"**Designer**",0));

}

## Добавляем в centralWidget компоновку (layout) с помощью дизайнера

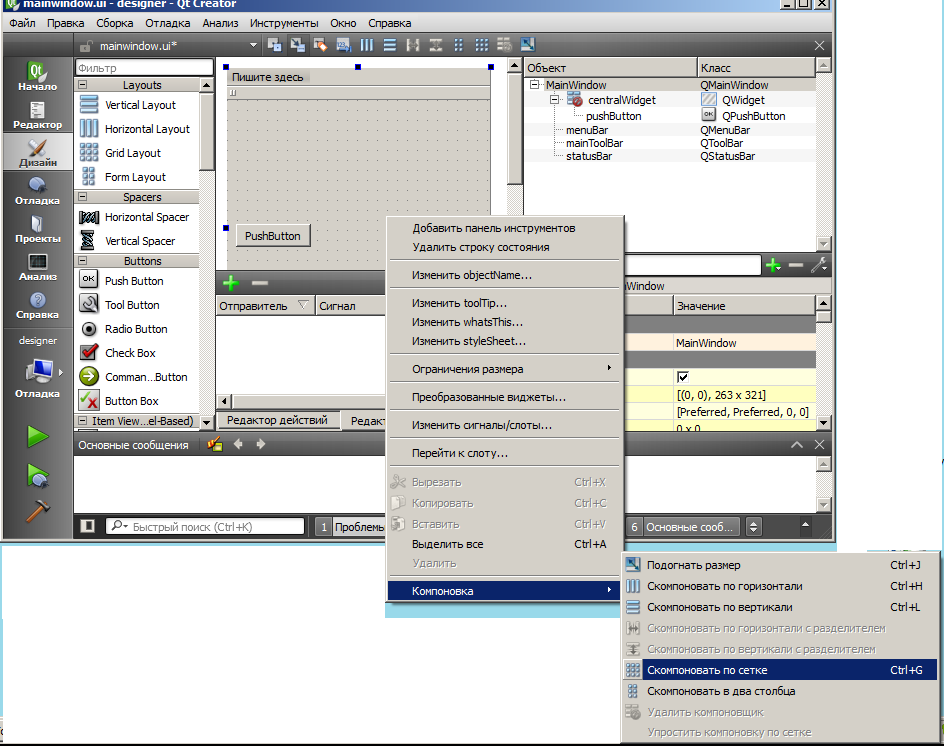
1. Перейдите в режим графического редактирования формы – Редактор/mainwindow.ui
2. Пока еще центральному виджету не соответствует никакая компоновка

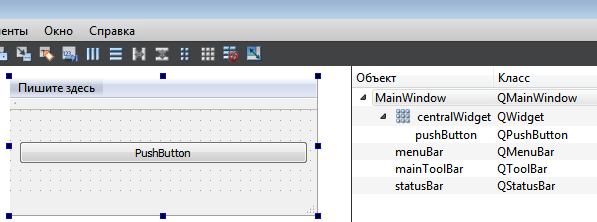


1. => чтобы «привязать» к центральному виджету компоновку, на форме должен быть **хотя бы один виджет!** => сначала с помощью дизайнера добавьте, например, PushButton (все равно потом пригодится!)  
   **Предупреждение (важное)** ! **Не** следует добавлять компоновку **для центрального виджета** с панели виджетов, так как в этом случае дизайнер не дает ассоциировать компоновку непосредственно с centralWidget, а создает еще одно промежуточное окно – владельца компоновки!
2. Два варианта добавления компоновки:  
   Первый вариант:  
   1-выделите Объект/centralWidget (при этом на toolbar-е значки компоновок станут активными)  
   2 – выберите grid-компоновку   
   

Второй вариант:

1.В свободной (от PushButton)области формы кликните правую кнопку мыши





1. В результате компоновка будет «привязана» к центральному виджету (какой бы вариант вы не использовали). При этом в файл ui\_mainwindow.h дизайнером будут внесены следующие изменения:

class Ui\_MainWindow

{

public:

QWidget \*centralWidget;

QPushButton \*pushButton;

QMenuBar \*menuBar;

QToolBar \*mainToolBar;

QStatusBar \*statusBar;

void setupUi(QMainWindow \*MainWindow)

{

if (MainWindow->objectName().isEmpty())

MainWindow->setObjectName(QString::fromUtf8("MainWindow"));

MainWindow->resize(400, 300);

centralWidget = new QWidget(MainWindow);

centralWidget->setObjectName(QString::fromUtf8("centralWidget"));

pushButton = new QPushButton(centralWidget);

pushButton->setObjectName(QString::fromUtf8("pushButton"));

pushButton->setGeometry(QRect(20, 90, 75, 23));

MainWindow->setCentralWidget(centralWidget);

menuBar = new QMenuBar(MainWindow);

menuBar->setObjectName(QString::fromUtf8("menuBar"));

menuBar->setGeometry(QRect(0, 0, 400, 21));

MainWindow->setMenuBar(menuBar);

mainToolBar = new QToolBar(MainWindow);

mainToolBar->setObjectName(QString::fromUtf8("mainToolBar"));

MainWindow->addToolBar(Qt::TopToolBarArea, mainToolBar);

statusBar = new QStatusBar(MainWindow);

statusBar->setObjectName(QString::fromUtf8("statusBar"));

MainWindow->setStatusBar(statusBar);

retranslateUi(MainWindow);

QMetaObject::connectSlotsByName(MainWindow);

} // setupUi

**. . .**

При этом должно получиться:

Проверьте функциональность приложения

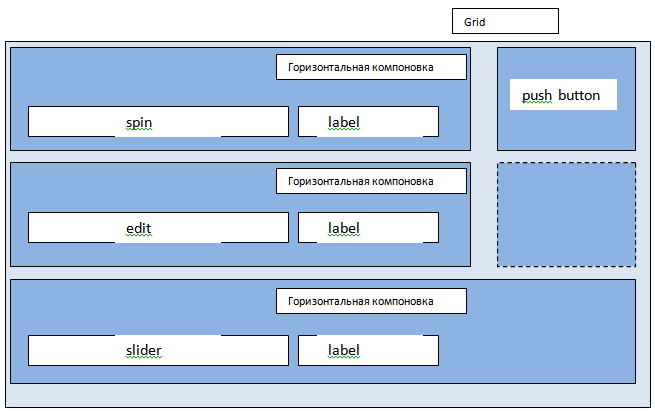
Замечания:

* После создания компоновки добавляем виджеты в уже существующую компоновку можно простым перетаскиванием с панели инструментов
* Можно поступить другим образом: добавить все требуемые виджеты на форму, отметить их (Ctrl+левая кнопка мыши), а затем в контекстном меню выбрать компоновку. В этом случае (на мой взгляд) порядок следования виджетов задавать труднее.
* Если (как в нашем случае) компоновка привязана к centralWidget, дизайнер не отрисовывает разметку компоновки. Для всех вложенных компоновок Вы увидите контуры вложенных компоновок – красным цветом.

## Добавляем элементы управления в компоновку и настраиваем их свойства

### Добавление элементов и их компоновка

Заготовка выглядит так:



Дальнейшую работу можно выполнять по- разному:

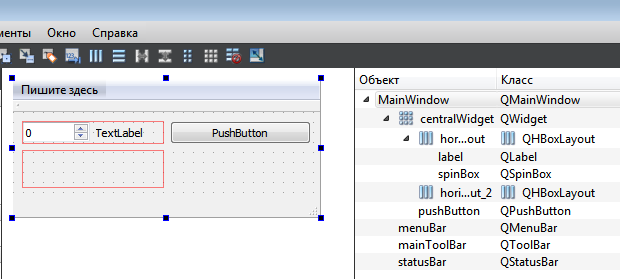
**Вариант 1.**

Можно использовать уже сформированную табличную компоновку и продолжить наполнять ее другими компоновками и элементами управления.

С помощью дизайнера сформируйте компоновки и элементы управления на форме (на центральном виджете) («перетащите» их с панели инструментов в компоновку) и задайте их свойства. При перетаскивании Вы увидите отмеченные синей линией места на форме, куда можно вставить виджет, а красным – непосредственно предполагаемое расположение. Нам понадобятся (кроме уже сформированной кнопки):

в левой колонке grid-а:

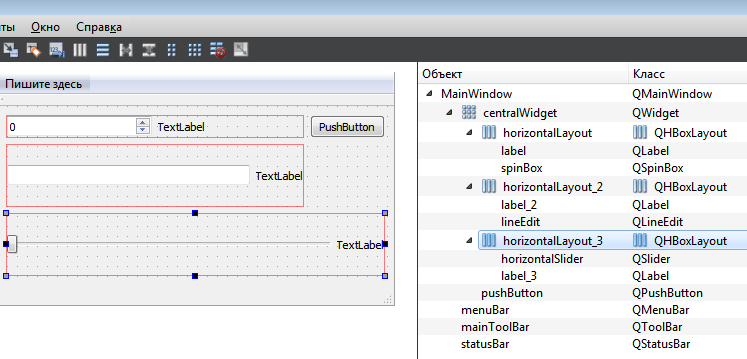
* Горизонтальная компоновка, в которую мы поместим Наборный счетчик (диапазон: 0-100) + Label для перевода фокуса в SpinBox



* Горизонтальная компоновка, в которую мы поместим поле редактирования (LineEdit), в которое позволим вводить только целые от 0 до 100   
  +Label для перевода фокуса в LineEdit
* Горизонтальная компоновка, в которую мы поместим горизонтальный слайдер (HorizontalSlider) для отображения масштаба диапазон: 0-100  
  + Label для перевода фокуса в слайдер  
  Так как слайдер должен занимать достаточное место, то «растянем» его на две колонки grid-а

в правой колонке grid-а:

* Кнопка (PushButton) , которую задействуем позже



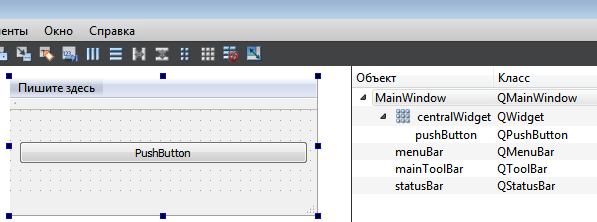
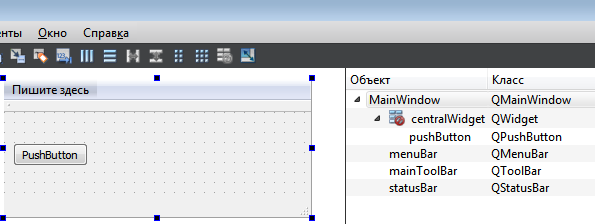
**Вариант 2.**

Возможно, Вам понравится другой вариант, при котором

* сначала на форму (на центральный виджет) помещаются элементы,
* Элементы компонуются необходимым образом,
* На форме (на центральном виджете) устанавливается главный компоновщик

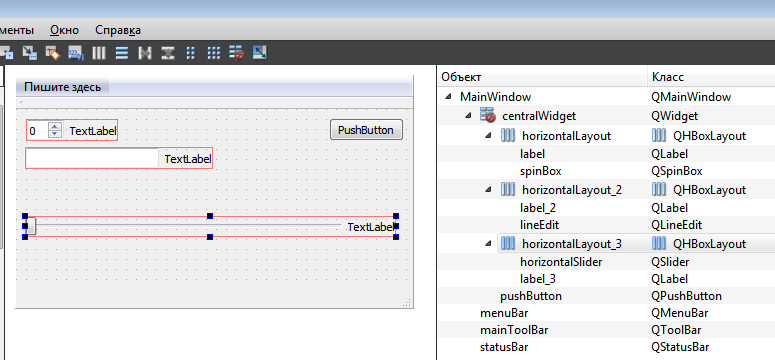
Замечание:

Прежде, чем будем но-новому компоновать элементы, надо **удалить** старый компоновщик с формы.

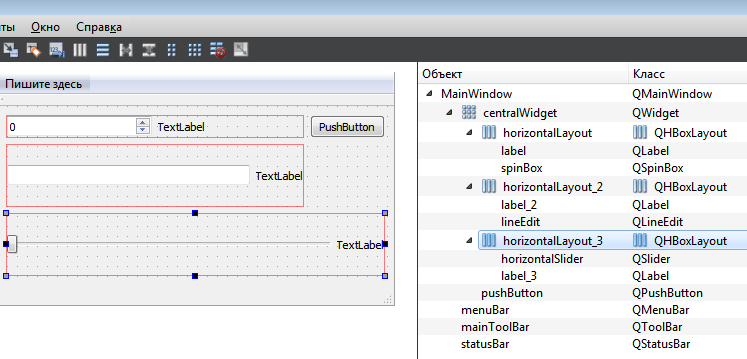
 

Для того, чтобы скомпоновать элементы каким –либо образом, можно:

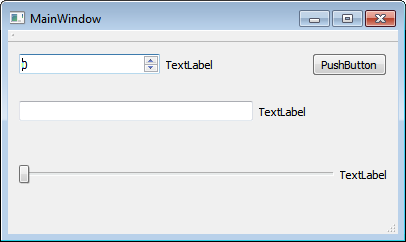
* Перетащить с панели элементов нужный компоновщик и потом в него поместить элементы
* Или выделить элементы, а потом выбрать на панели инструментов нужный компоновщик



На заключительном этапе необходимо выбрать центральный виджет и установить на нем компоновку. В результате, эффект будет тот же, что и при первом варианте.



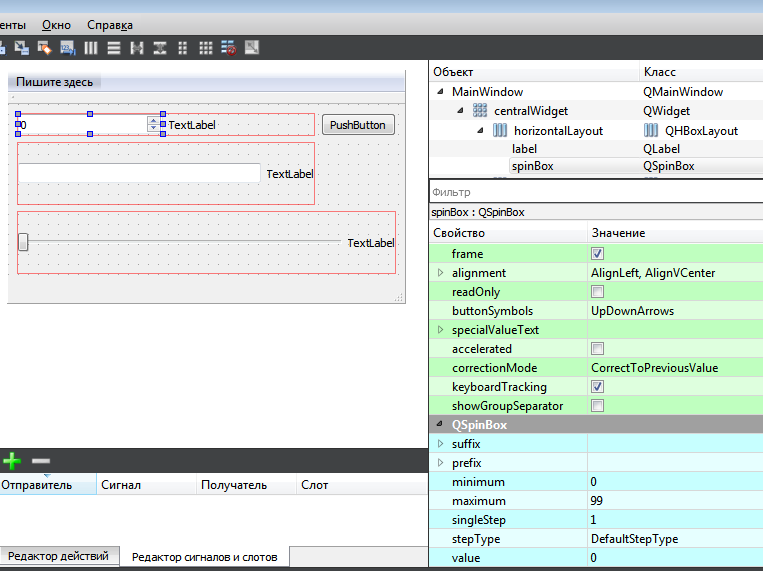
А вот как будет выглядеть приложение:



### Настройка свойств элементов управления

Настраиваем в дизайнере вид и свойства элементов управления:

* Желательно (хороший стиль) изменить имена элементов управления, заданные по умолчанию. Хорошо, когда имена несут какую-то смысловую нагрузку.
* Надписи на кнопке и label (во всяком случае, исходная строка, которая должна появиться при запуске приложения) легко задаются в дизайнере посредством свойства «text»
* Редактировать свойства компоновки можно тоже с помощью дизайнера, например:



* Поле ввода слишком длинное, а отображать оно будет проценты =>можно ограничить его длину - свойство maximumSize/ширина. Позволять вводить символы нельзя. Так как речь идет об отображении процентов, максимальное количество цифр не может быть больше двух (0 - 99).   
  Подсказка: в прошлой работе, чтобы обеспечить ввод только цифр, мы создавали валидатор. Попробуйте задать для LineEdit валидатор самостоятельно.

Замечание: валидатор создаем вручную в конструкторе MainWindow и там же устанавливаем его на поле редактирования. Доступиться до любого элемента на форме можно посредством указателя **ui,**  например, так: **ui-> lineEdit**

* Спин – скорее всего должен иметь такую же ширину
* Свойства слайдера – по желанию разработчика.
* …

Любые эксперименты по формированию интерфейса приветствуются!

Например, для слайдера можно:

* Задать диапазон
* tickPosition, tickInterval…
* Всплывающую подсказку toolTip, например, «Масштаб %»
* Текст в статусной строке - statusTip
* …

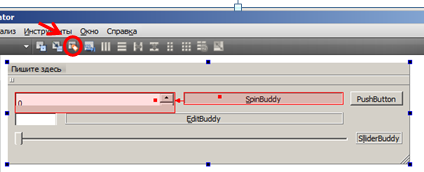
Замечания:

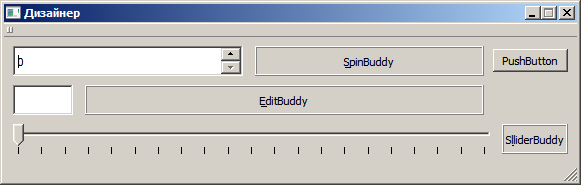
* при помещении элементов управления в область компоновки соответствующие объекты создаются динамически и добавляются в компоновку посредством addWidget() (см. файл ui\_mainwindow.h)
* при изменении свойств виджетов с помощью дизайнера вызов соответствующих методов добавляется в ui\_mainwindow.h

## Партнеры – buddy

Buddy – это простой и удобный способ перевести фокус ввода в элемент управления. Виджет label можно поставить в соответствие любому другому виджету (setBuddy()). Если в тексте label присутствует символ «&», например - &SpinBuddy, то во время работы программы при нажатии **«<Alt>+подчеркнутый символ»** на форме при выводе текста метки (label) следующий за «&» символ будет подчеркиваться, а назначенный посредством setBuddy() партнер label будет получать фокус ввода.

* Для сопоставления партнеров QtCreator предоставляет специальный редактор партнеров.
* В режиме редактирования виджетов задайте текст label, например: «&SpinBuddy»
* Перейдите в режим редактирования партнеров и сопоставьте любому элементу ввода метку label. Для этого соедините метку и соответствующий элемент (соединение выполняется от метки к элементу):

****Проверьте функциональность приложения. Теперь оно должно выглядеть так и реагировать на Alt + подчеркнутый символ



## Редактор сигналов/слотов

Горячие клавиши:

* F4 – редактирование сигналов/слотов
* F3 – редактирование виджетов

Напоминание: во всех элементах управления разными способами должно отображаться одно и то же значение =>при изменении значения с помощью любого из редактируемых элементов управления, в остальных элементах управления оно должно тоже изменяться.

Важно!

Дизайнер – это графический редактор, поэтому он позволяет автоматически генерировать код только в том случае, **когда связь элементов можно показать графически**. То есть дизайнер позволяет связать между собой с помощью сигналов/слотов только один виджет с другим виджетом. Если же требуется связать невизуальные объекты или хотя бы один из объектов является невизуальным, нужно уметь это делать «вручную» (см. предыдущую лаб. работу)

Создавать соединение посредством дизайнера можно двумя способами:

1 вариант - визуально

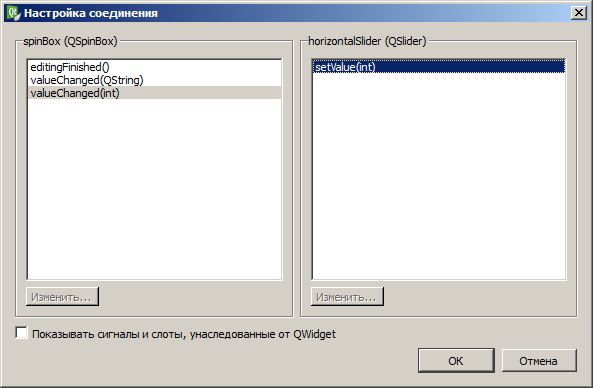
2 вариант с помощью редактора сигналов/слотов:

* Открыть закладку «редактор сигналов/слотов»
* Нажать «+»
* Выбрать отправителя и сигнал
* Выбрать получателя и слот

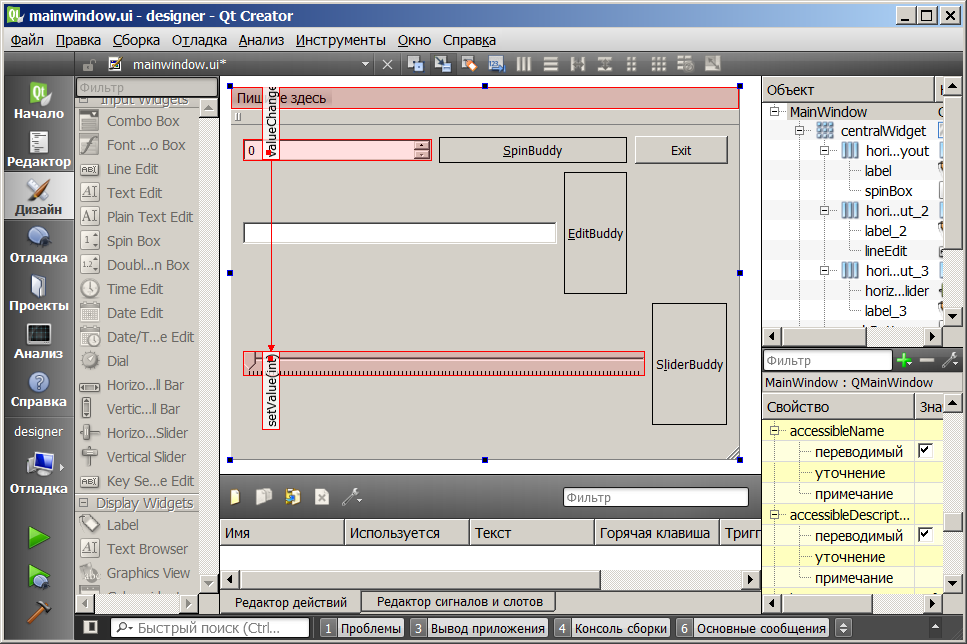
### Визуально соединяем наборный счетчик с ползунком

В качестве примера визуального соединения обеспечим коррекцию позиции слайдера при изменении значения наборного счетчика:

* Перейти в режим редактирования сигналов/слотов (меню «Правка», кнопка на панели инструментов или F4)
* Выделить мышью виджет, который будет генерировать сигнал (например, наборный счетчик) и соединить мышью с виджетом, который будет обрабатывать посредством своего слота генерируемый сигнал, например, слайдер
* При этом будет выведено диалоговое окно, в котором нужно выбрать требуемые сигнал и слот:



В результате соединение будет отображено графически:



а в файл ui\_mainwindow.h будет добавлен код:

…

retranslateUi(MainWindow);

**QObject::connect(spinBox,SIGNAL(valueChanged(int)),horizontalSlider,**

**SLOT(setValue(int)));**

…

Самостоятельно при изменении позиции ползунка посылаем сигнал наборному счетчику.

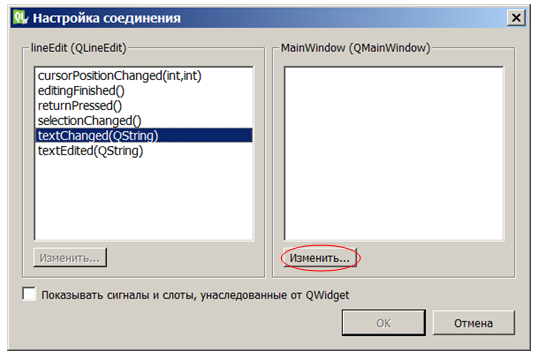
Напоминание:

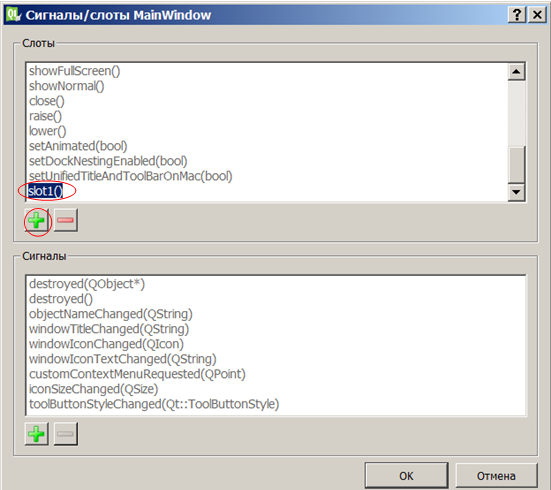
1. следует учесть, что при выполнении слота (при условии, что значение меняется) в свою очередь генерируется сигнал об изменении значения

2. поле вывода (QLabel)не генерирует сигнал при изменении значения!

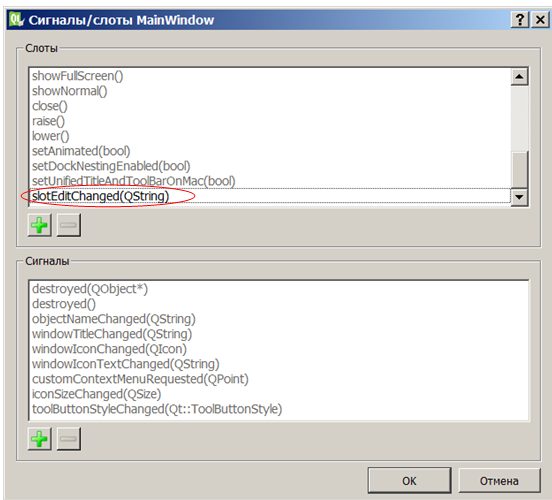
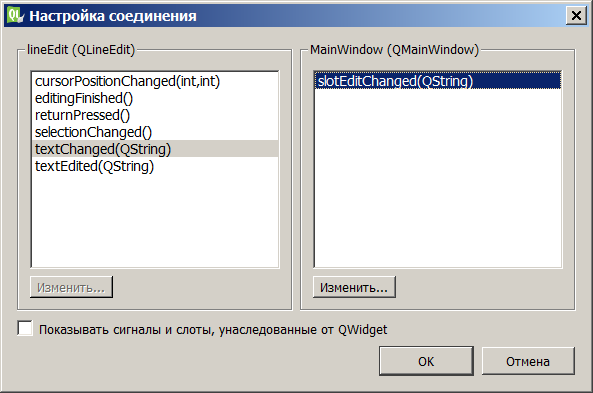
3. поле редактирования (**QLineEdit**)генерирует сигнал textChanged(QString), который формирует параметр типа QString, а слоты наборного счетчика и ползунка setValue(int)принимают в качестве параметра целое значение! В качестве посредника можно использовать класс MainWindow =>после того, как Вы добавите в класс MainWindow **требуемые сигнал и слот**, их можно будет «соединить» в режиме дизайнера, только имена пользовательских сигналов и слотов, а также типы параметров пользовательских сигналов/слотов придется задавать «вручную»:

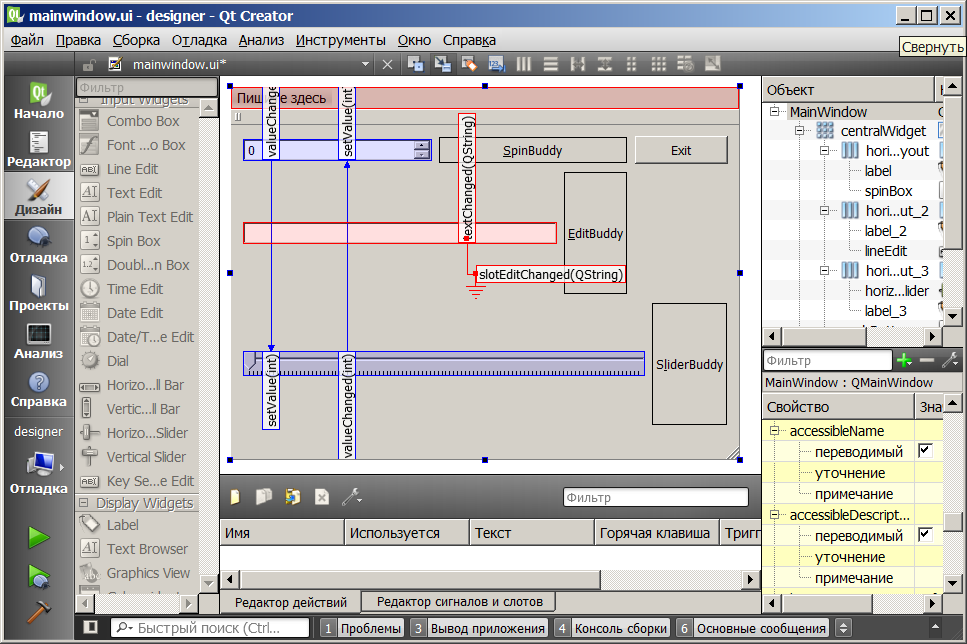
После визуального соединения поля редактирования с главным окном:





Заменить slot1() на сигнатуру своего слота:



С помощью дизайнера самостоятельно предусмотрите обработку пользовательского сигнала MainWindow

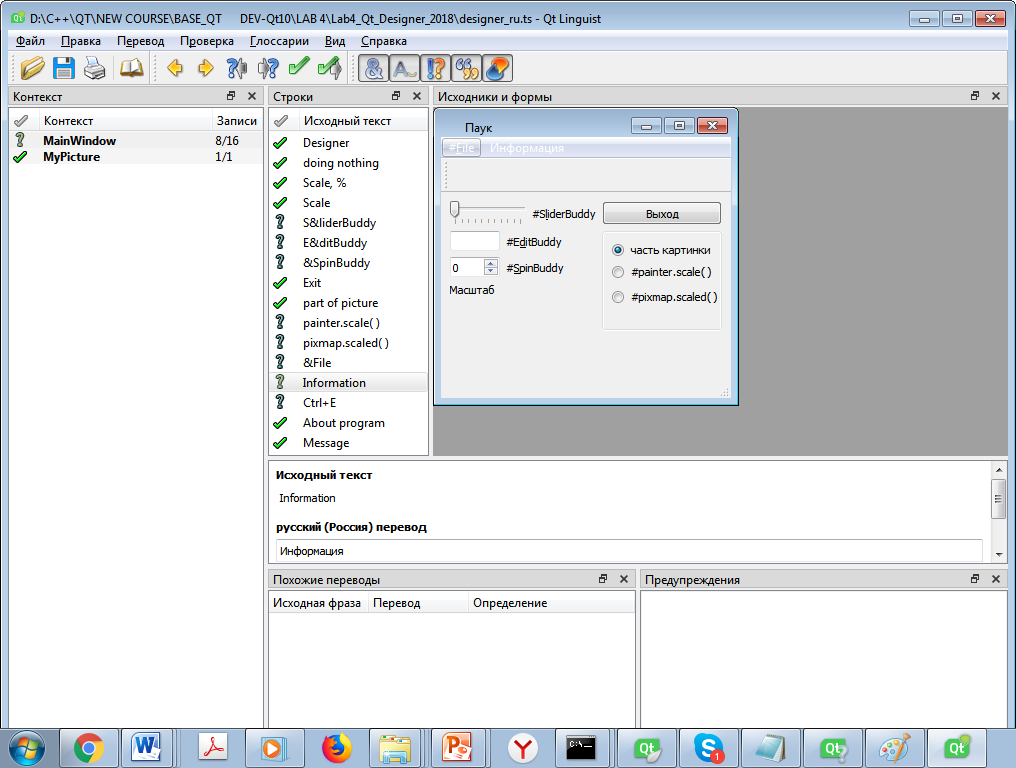
Самостоятельно сделайте такое соединение, которое будет вызывать завершение приложения по нажатию кнопки «Exit».   
Подсказка: у класса QMainWindow есть унаследованный от QWidget public слот – Close(), который при закрытии главного окна вызывает в свою очередь завершение приложения

## Перевод интерфейса на русский язык

Используя навыки, полученные в первой лабораторной работе, переведите элементы интерфейса на русский язык.

**Замечания:**

1. Не надо преводить строки меток, т.к. мы используем их в качестве партнеров-buddy для перевода фокуса на соответствующий элемент ввода.
2. Не надо строки текста, предназначенные для перевода «заворачивать» в метод **tr().**
3. При открытии файла **.ts** в QLinguist нам будет представлена форма нашего интерфейса. Переведенные строки сразу же отображаются на этой форме.
4. В результате мы увидим:



## Система Ресурсов Qt. Редактор ресурсов

Приложению обычно требуются специфические изображения, пользовательские диалоговые панели, иногда курсоры, иконки… Большинство таких ресурсов можно загружать из файлов по мере необходимости посредством вызова соответствующих функций Qt (см. работу №1), но при этом разработчик приложения должен предоставлять Заказчику кроме приложения все требуемые дополнительные файлы. Гораздо удобнее хранить все используемые приложением ресурсы непосредственно в исполняемом файле.  
Важным исключением являются приложения, которые оперируют большим количеством изображений большого размера.

Система ресурсов Qt предоставляет системо-независимое:

* хранение ресурсов в исполняемом файле в двоичном виде
* и их использование

Описания ресурсов приложения содержатся в XML-подобном формате в файле с расширением **.qrc**. Это текстовый файл, который содержит ссылки на файлы, содержащие собственно ресурсы. Конечно, можно (зная синтаксис) создать и наполнить содержимым такой файл «вручную», но гораздо удобнее делегировать формирование такого файла редактору ресурсов (у него меньше возможностей наделать ошибок).

Конечная цель редактора ресурсов – создать файл описания ресурсов в XML-формате. Такой файл:

* подключается в проект,
* обрабатывается специальным «компилятором» ресурсов **rcc**, который создает дополнительный файл .cpp (в нашем случае qrc\_MyImages.cpp), в который заносит информацию о ресурсе (в нашем случае об иконке) в следующем виде:

//qrc\_MyImages.cpp

static const unsigned char qt\_resource\_data[] = {

// D:/C++/QT/NEW COURSE/BASE\_QT DEV-Qt10/LAB 4/Lab4\_Qt\_Designer\_2018/Images\_lab4/icon1.ico

0x0,0x0,0x1,0xc,

0x0,

0x0,0x4,0x36,0x78,0x9c,0x63,0x60,0x60,0x64,0x60,0x62,0x10,0x10,0x10,0x60,0x60,

0x60,0x60,0x61,0xd0,0x60,0x64,0x60,0x50,0x3,0xb2,0x14,0x14,0x20,0xfc,0x17,0x4c,

0xc,0xc,0x7e,0x40,0x31,0xd,0x20,0xf,0x24,0xa2,0x0,0xc4,0x8c,0x40,0x71,0x10,

. . .

};

static const unsigned char qt\_resource\_name[] = {

// Icons

0x0,0x5,

0x0,0x4f,0xa6,0x53,

0x0,0x49,

0x0,0x63,0x0,0x6f,0x0,0x6e,0x0,0x73,

// Images\_lab4

0x0,0xb,

0xc,0x36,0x48,0xf4,

0x0,0x49,

0x0,0x6d,0x0,0x61,0x0,0x67,0x0,0x65,0x0,0x73,0x0,0x5f,0x0,0x6c,0x0,0x61,0x0,0x62,0x0,0x34,

// icon1.ico

0x0,0x9,

0x6,0x14,0x90,0xdf,

0x0,0x69,

0x0,0x63,0x0,0x6f,0x0,0x6e,0x0,0x31,0x0,0x2e,0x0,0x69,0x0,0x63,0x0,0x6f,

};

static const unsigned char qt\_resource\_struct[] = {

// :

0x0,0x0,0x0,0x0,0x0,0x2,0x0,0x0,0x0,0x1,0x0,0x0,0x0,0x1,

0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,

// :/Icons

0x0,0x0,0x0,0x0,0x0,0x2,0x0,0x0,0x0,0x1,0x0,0x0,0x0,0x2,

0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,

// :/Icons/Images\_lab4

0x0,0x0,0x0,0x10,0x0,0x2,0x0,0x0,0x0,0x1,0x0,0x0,0x0,0x3,

0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,

// :/Icons/Images\_lab4/icon1.ico

0x0,0x0,0x0,0x2c,0x0,0x1,0x0,0x0,0x0,0x1,0x0,0x0,0x0,0x0,

0x0,0x0,0x1,0x20,0x52,0x73,0xf3,0xc0,

};

. . .

* далее этот файл компилируется
* полученный объектный модуль линкуется с остальными объектными модулями
* таким образом ресурсы «интегрируются» в исполняемый файл
* в процессе работы приложения «достать» любой ресурс можно соответствующими функциями Qt

Замечание: для того, чтобы компилятор ресурсов «учитывал» файл .qrc, в файле проекта должна появиться директива:  
**RESOURCES += <имя\_файла>.qrc**Можно добавить эту директиву «вручную», но гораздо удобнее это сделать с помощью QtCreator.

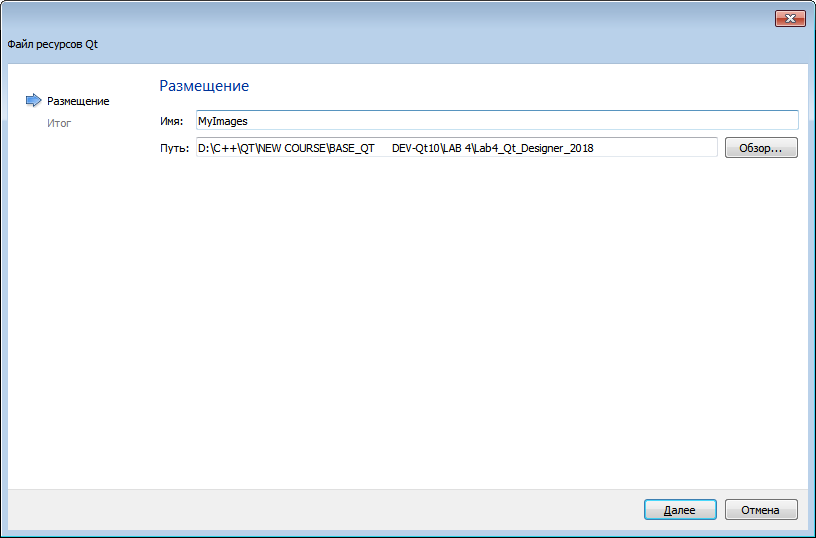
### Как использовать редактор ресурсов в QtCreator-е

1. Создаем файл .qrc и добавляем его в проект:

– «добавить новый»:



Далее следуем указаниям визарда, например:



В результате:

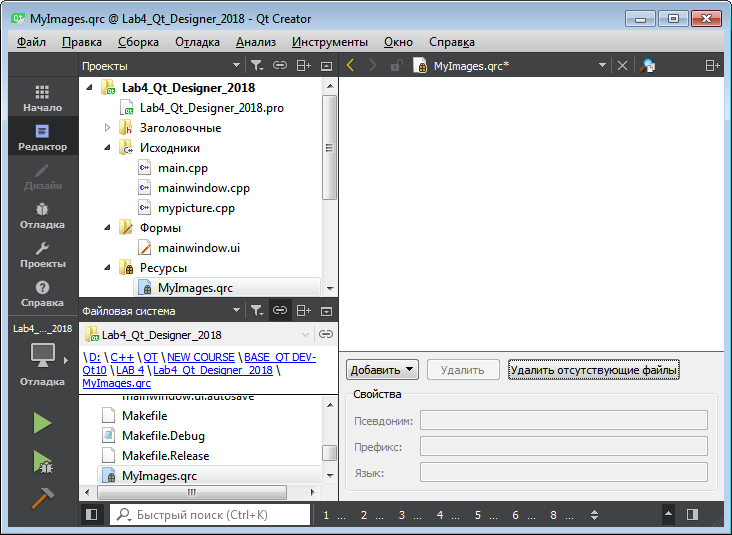
* в окне «Проекты» появится виртуальная папка «Ресурсы»
* будет создан практически пустой файл с заданным именем и расширением .qrc и помещен в виртуальную папку «Ресурсы»
* в файл проекта будет добавлена директива

**RESOURCES += \**

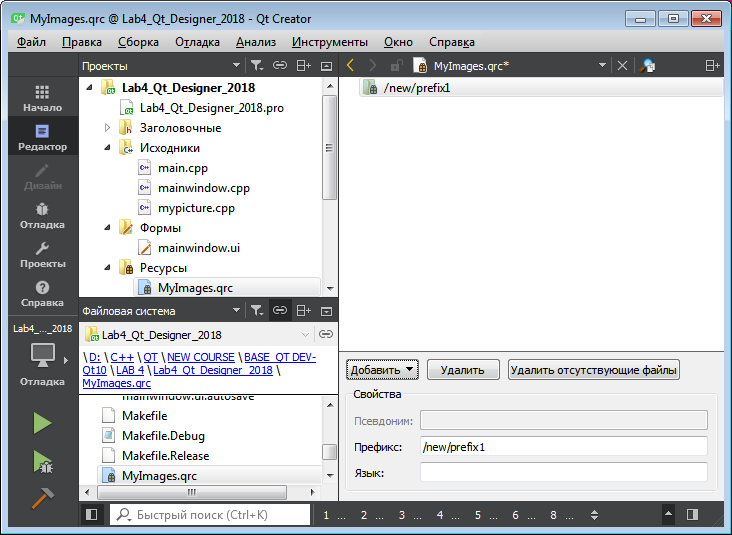
<имя>**.qrc**

2**. Важно!** Все, требуемые для создания ресурсов, файлы должны храниться или в той же директории, где создается файл .qrc, или в одном из подкаталогов. Создадим в папке проекта каталог, например, Images и перепишем туда иконку и картинки из моей папки (или целиком папку)

3. Для перехода в режим редактирования ресурсов дважды кликните файл\*.qrc или правую кнопку мыши/из контекстного меню –> открыть. Откроется редактор ресурсов



Нажмите кнопку «Добавить» и выберите **«Добавить префикс»**.



**Смысл префикса**:

В Qt существуют соглашения об именовании, благодаря которым можно однозначно обращаться к ресурсам в тексте программы.

Файлы ресурсов, перечисленные в файле .qrc, являются частью дерева исходников приложения. Указываемые пути являются относительными к каталогу, содержащему файл .qrc. Префикс – это способ задания относительного пути к файлам, то есть в функциях Qt, которые загружают ресурсы из исполняемого файла, можно задавать «виртуальную» спецификацию файла с помощью **“:/Префикс/имя\_файла”.**

По умолчанию, ресурсы доступны в приложении под тем же именем, под которым они находятся в дереве исходников, с помощью префикса «**:/»**. Например, путь «:/images/img.png» даст доступ к файлу **img.png**, чьё расположение в дереве исходников приложения images/img.png.

Замечание: при использовании префикса регистр **имеет значение**! (case sensitive).

С помощью задания префикса можно специфицировать различные ресурсы. Например, Вы можете для различных видов ресурсов задать разные префиксы: для иконок - один, для картинок -другой, для файлов переводов - третий.

Замените тот префикс, который предлагает редактор ресурсов**/new/prefix1**, например, на:

**/Icons**

1. Нажмите кнопку «Добавить» и выберите «Добавить файл» и выберите требуемый файл (например, иконку).

При этом в файле .qrc появится соответствующая ссылка.

<RCC>

<qresource prefix="/Icons">

<file>Images\_lab4/icon1.ico</file>

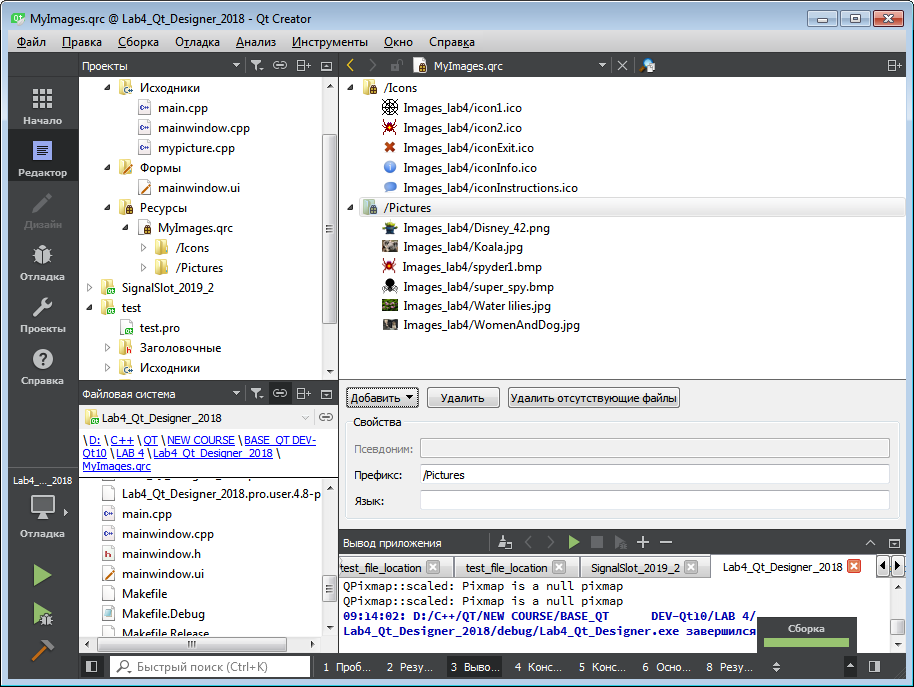
</qresource>

</RCC>

, где «**icon1.ico**» - спецификация файла.

Добавьте остальные нужные файлы.

У нас может получиться, например, такое:

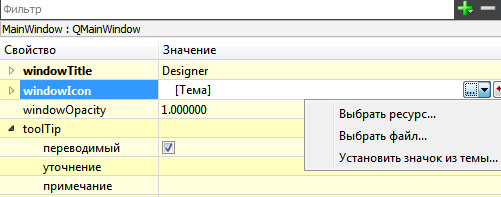


Ресурсы готовы к использованию.

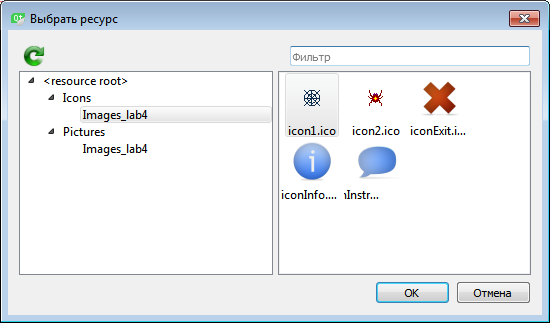
### Иконка

Пока мы добились того, что иконка будет храниться в исполняемом файле в двоичном виде, но никакого отношения к нашему главному окну она не имеет. Чтобы сделать иконку атрибутом главного окна приложения:

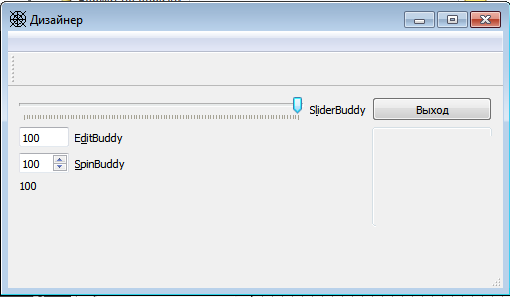
* откройте файл формы (mainwindow.ui) в режиме дизайнера
* сделайте активной форму
* в свойстве **windowIcon** (пока там иконка Qt по умолчанию)



– «Выбрать ресурс»



Теперь при запуске приложения Вы должны увидеть свою иконку.



### Картинка на кнопке

Изображения тоже удобно хранить в качестве ресурсов в исполняемом файле.

Не требует почти никаких знаний. Порядок действий:

* добавьте к уже имеющимся ресурсам изображение – самостоятельно
* открыть форму в режиме дизайнера и выделить кнопку
* в свойствах “icon” выбрать из ресурсов требуемое изображение
* размеры лучше задать 32\*32, иначе иконку будет трудно рассмотреть



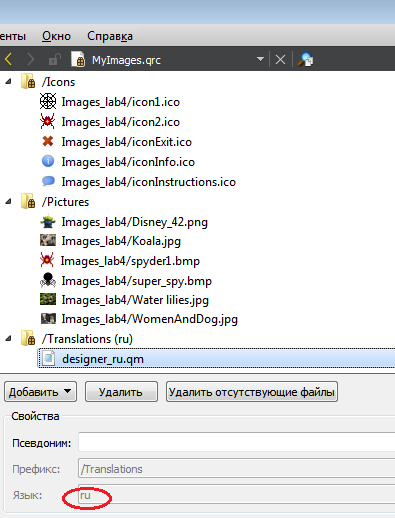
### Использование ресурсов для хранения файла переводов

В первой лабораторной работе мы использовали для перевода интерфейса файл с расширением «**.qm».**

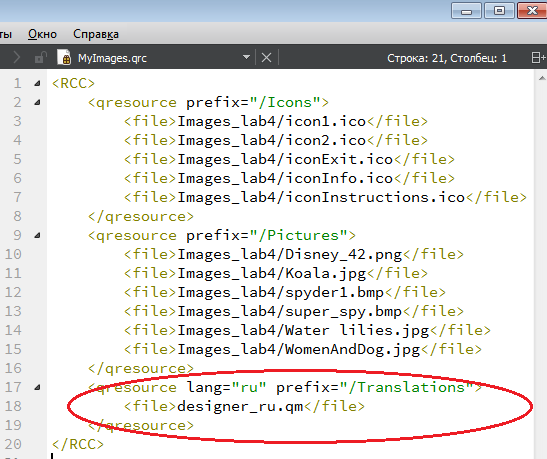
Для того, чтобы перевод выполнялся и в случае запуска приложения (без QtCreator) нам приходилось копировать этот файл в соответствующую папку. Это крайне неудобно.

Добавим в ресурсы нашего приложения файл переводов.

1. Откройте файл ресурсов в редакторе ресурсов
2. Задайте префикс, например «/Translations»
3. Добавьте файл перевода
4. Пометьте, что язык перевода будет русский (ru)



1. При этом, файл в XML-формате будет выглядеть так:

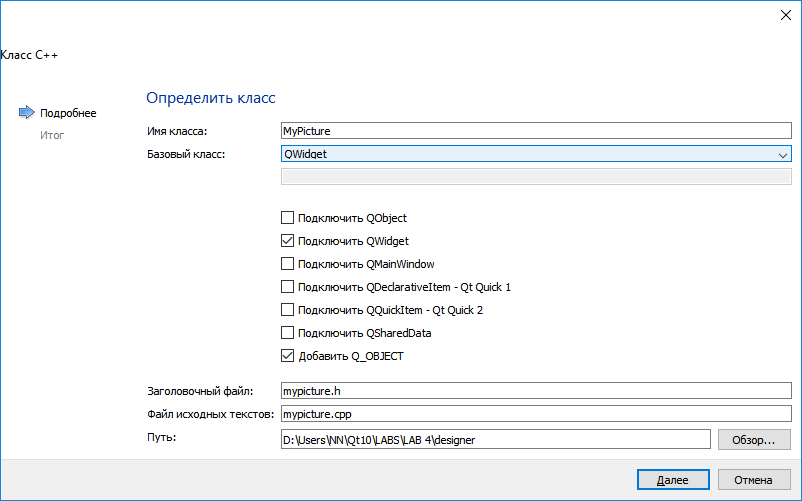


1. Остается только подкорректировать обращение к файлу **. qm** из функции main. Теперь нам надо указать имя файла ресурса перевода, указав необходимый префикс.

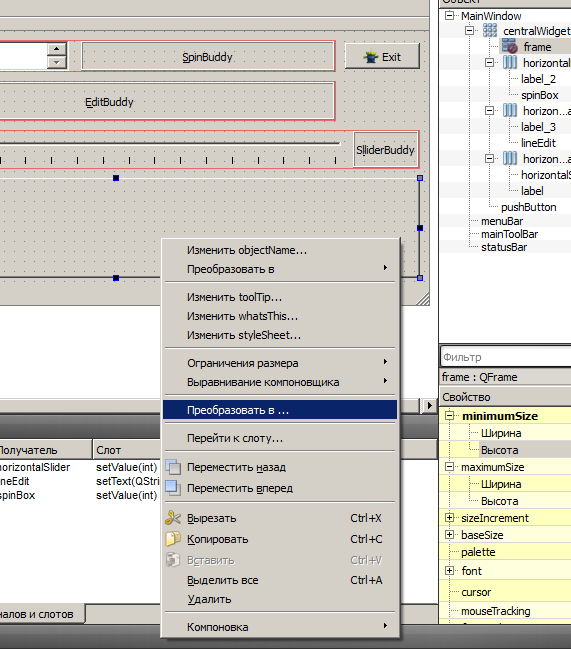
### Пользовательский виджет + загрузка изображения из ресурсов

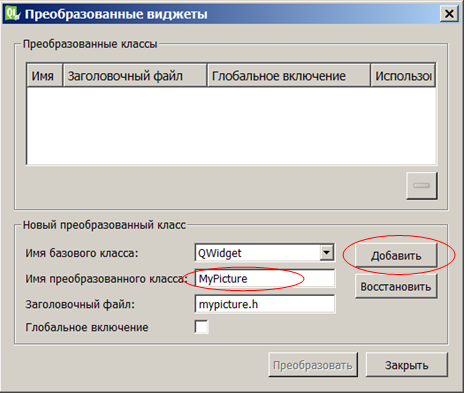
Задача: рисуем или часть изображения в зависимости от введенного значения, или масштабируем изображение

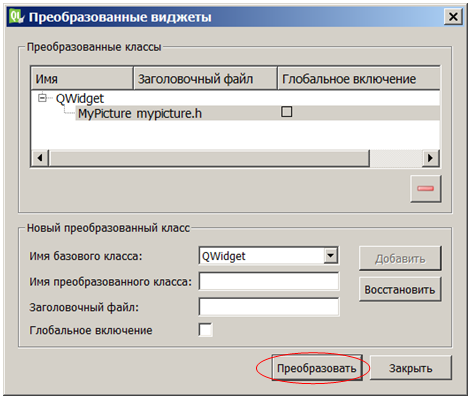
1. Создать класс, производный от QWidget (например, MyPicture)



1. В режиме дизайнера выбрать на панели инструментов (в секции «Containers»)–Widget (или Frame) и разместить его в компоновке. Подсказка – лучше задать минимальную высоту.
2. В режиме дизайнера сделать активным созданный виджет и посредством контекстного меню - выбрать «Преобразовать в …»



 Нажмите «Добавить» ->

 Нажмите «Преобразовать».

1. Чтобы отобразить любое изображение, хранящееся в качестве ресурса, нужно загрузить это изображение в объект типа QPixmap =>в классе MyPicture объявите переменную типа QPixmap (например, с именем pixmap)
2. В конструкторе MyPicture загрузите изображение в объект pixmap из ресурсов приложения

pixmap.load(“:/Pictures/Images\_lab4/WomenAndDog.jpg”);

1. Объявите в классе MyPicture переменную для хранения масштаба отображаемой картинки, например, double scale; (в конструкторе проинициализируйте переменную единицей)
2. Перегрузите виртуальный метод paintEvent() для обработки события о перерисовке окна.

void  QWidget::**paintEvent**([QPaintEvent](qthelp://org.qt-project.qtgui.532/qtcore/qtglobal.html) \**event*) [virtual protected]

Предусмотрите вывод картинки (pixmap). Выберите наиболее подходящий метод из:

void QPainter::**drawPixmap**(const [QPoint](qthelp://org.qt-project.qtwidgets.502/qtcore/qstring.html) &*point*, const [QPixmap](qthelp://org.qt-project.qtwidgets.502/qtwidgets/qwidget.html) &*pixmap*)

void QPainter::**drawPixmap**(const [QRect](qthelp://org.qt-project.qtgui.502/qtcore/qrect.html) &*target*, const [QPixmap](qthelp://org.qt-project.qtgui.502/qtgui/qpixmap.html) &*pixmap*, const [QRect](qthelp://org.qt-project.qtgui.550/qtgui/qpixmap.html) &*source*)

void QPainter::**drawPixmap**(const [QPoint](qthelp://org.qt-project.qtgui.532/qtcore/qpoint.html) &*point*, const [QPixmap](qthelp://org.qt-project.qtgui.532/qtcore/qtglobal.html) &*pixmap*, const [QRect](qthelp://org.qt-project.qtwidgets.502/qtgui/qpaintevent.html) &*source*)

### void QPainter::drawPixmap (const [QRectF](qthelp://org.qt-project.qtgui.550/qtcore/qrectf.html) &*target*, const [QPixmap](https://csspbstu.sharepoint.com/sites/22DEV-QT10.Qt/DocLib/qtgui/qwheelevent.html) &*pixmap*, const [QRectF](qthelp://org.qt-project.qtgui.502/qtcore/qrect.html) &*source*)

…

Подсказки:

1. для определения размеров:  
   изображения– QRect QPixmap::**rect**() const  
   любого виджета–без учета рамки – QRect QWidget::rect() const
2. Вариант 1. Можно использовать для выполнения преобразования (масштабирования) системы координат метод: void QPainter::**scale**([qreal](qthelp://org.qt-project.qtgui.550/qtcore/qrectf.html#qreal-typedef)*sx*, [qreal](qthelp://org.qt-project.qtgui.502/qtcore/qrect.html#qreal-typedef)*sy*) , а после этого рисуется исходная картинка. Но при этом подходе картинка не «вписывается» в виджет.
3. Вариант 2. Другое решение - масштабировать картинку (QPixmap) по размерам виджета. Для этого можно использовать метод:

### [QPixmap](qthelp://org.qt-project.qtgui.550/qtgui/qpixmap.html#QPixmap)  QPixmap::scaled (int width, int height,

### [Qt::AspectRatioMode](qthelp://org.qt-project.qtgui.502/qtgui/qpixmap.html#AspectRatioMode-enum) aspectRatioMode = Qt::IgnoreAspectRatio) const;

для сохранения пропорций картинки указывается Qt**::KeepAspectRatio.**

Создается масштабированная картинка на основе исходной: [QPixmap](qthelp://org.qt-project.qtgui.550/qtcore/qrectf.html#QPixmap)  tmp= pxm.scaled(this->width()\*scale/100, this->height()\*scale/100);

И на виджете рисуется масштабированная картинка в позиции (0,0).

1. Вариант 3. Третье решение - отображать часть картинки в зависимости от текущего масштаба. При scale=100 отображается полная картинка. При scale < 100 отображается фрагмент картинки. Для этого необходимо:

* Создать прямоугольник, соответствующий фрагменту картинки- *source*

QRectF source(0.0, 0.0, pixmap.width()\*scale/100, pixmap.height() \* m\_scale/100);

* Создать прямоугольник, задающий область отображения картинки –*target*

QRectF target(0.0, 0.0, this->width()\*m\_scale/100,this->height()\*m\_scale/100);

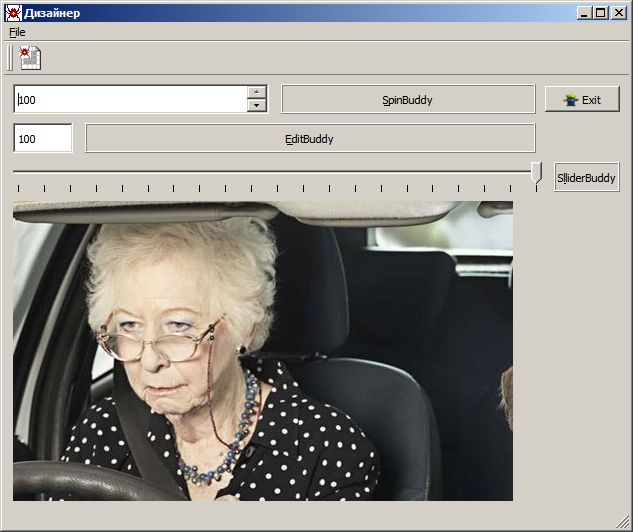
* Для отображения фрагмента картинки использовать

void QPainter::drawPixmap (const [QRectF](qthelp://org.qt-project.qtgui.502/qtcore/qpoint.html) & target, const [QPixmap](qthelp://org.qt-project.qtgui.550/qtgui/qpixmap.html) & pixmap, const [QRectF](qthelp://org.qt-project.qtgui.550/qtcore/qrectf.html) & source)

1. В классе MyPicture создайте слот, который будет реагировать на изменение величины (например, scaleChanged(int)), в котором:

* сформируйте значение scale
* вызовите перерисовку виджета MyPicture посредством repaint()

1. Свяжите сигнал (например, ползунка) и пользовательский слот виджетаMyPicture:



### 

Можно тестировать

## Действия Qt (класс QAction)

* В GUI-приложениях обычно **одно и то же действие**(=>один и тот же слот) можно инициировать несколькими разными способами:
  + из основного меню,
  + из контекстного меню,
  + посредством кнопки на toolbar-е,
  + при нажатии акселератора,
  + при нажатии обычной кнопки.
* Кроме того, большинству инициаторов одного и того же действия сопоставляются **одни и те же внешние атрибуты** (элементы интерфейса):
  + пиктограмма,
  + tooltip…
* Кроме того, при выполнении программы отправители могут изменять свое состояние, например, становиться неактивными (dimmed). **При этом все инициаторы одного и того же действия должны вести себя одинаково** (все dimmed)!

Если бы Qt не поддерживала концепцию единых действий, программисту приходилось бы для каждого отправителя сигнала выполнять одни и те же действия (соединить пункт меню с требуемым слотом, кнопку на toolbar-е с тем же слотом…), да еще синхронизировать всех отправителей.

В рамках концепции единых действий Qt объединяет всех инициаторов одного и того же действия посредство класса **QAction**.

### Класс QAction

Класс QAction:

* абстрагирует источник действия
* синхронизирует все источники. Например, если нужно сделать соответствующие данному действию пункт меню, кнопку на tool-bar-е… неактивными, достаточно вызвать   
  voidQAction::setDisabled(bool);
* при связывании объекта QAction (connect())
  + с меню – добавляет в это меню соответствующий пункт
  + с tool-bar-ом – добавляется соответствующая кнопка
* добавляет в пункт меню, кнопку на tool-bar-е:
  + tooltip,
  + иконку…
* Если при выполнении программы действие должно быть заблокировано/разблокировано, вызов setEnabled(false/true); блокирует/разблокирует все источники действия

### Создание действия.Редактор действий

Объекты-действия можно создавать двумя способами:

* Создать новое действие с помощью редактора действий, а потом «перетаскивать» это действие на меню или toolbar
* Создать пункт меню – при этом дизайнер автоматически создает объект-действие (далее это действие можно «перетащить» на toolbar)

## Меню. QMenuBar, QMenu

Меню-это интерфейсный элемент главного окна приложения (QMenuBar). Для окна можно создать только одно главное меню! И сколько угодно контекстных меню.

Сгенерированный Qt класс Ui\_MainWindow предоставляет поддержку меню посредством

ClassUi\_MainWindow

{

public:

…

**QMenuBar\*menuBar;**

…

voidsetupUi(QMainWindow\*MainWindow)

{

…

menuBar=newQMenuBar(MainWindow);

menuBar->setObjectName(QStringLiteral("menuBar"));

menuBar->setGeometry(QRect(0,0,609,18));

menu\_File=newQMenu(menuBar);//пунктменюассоциируетсясменю

menu\_File->setObjectName(QStringLiteral("menu\_File"));

MainWindow->setMenuBar(menuBar);//ассоциируетменюсглавнымокном

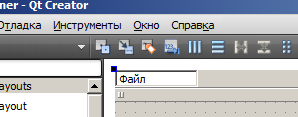
…

Замечание: если Вы не пользуетесь каркасом приложения, то меню можно создать с помощью контекстного меню дизайнера (в режиме дизайнера кликните правую кнопку мыши в области формы и выберите «Создать меню»)

Создадим пункт меню. **При его создании дизайнер автоматически также создает объект-действие**!

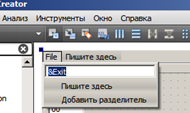
Последовательность действий:

* Перейдите в режим дизайнера
* Сначала задаем заголовок выпадающего меню верхнего уровня, например “&File” (там, где «Пишите здесь»)



&File

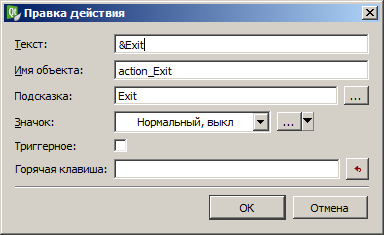
* Затем задаем название пункта меню. Важно! Так как при этом дизайнер создает соответствующее действие. Имя объекта-действия дизайнер будет формировать исходя из названия =>название нужно задавать по-английски.
  + После того, как действие создано, название можно будет изменить
  + А лучше пользоваться возможностью перевода посредством QtLinguist-а



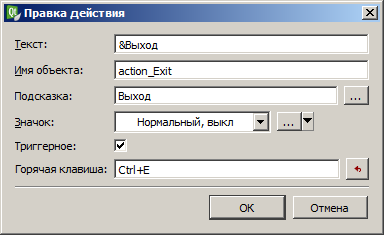
После того, как Вы нажали «Enter», будет создан объект QActionс именем action\_<название пункта меню>, в моем случае action\_Exit. Его можно увидеть в дереве объектов: menuBar/menu/ action\_Exit.

Действием уже можно пользоваться, но у него гораздо больше возможностей, чем то, что задано по умолчанию при создании объекта таким способом =>всегда можно что-то добавить или изменить с помощью редактора действий:

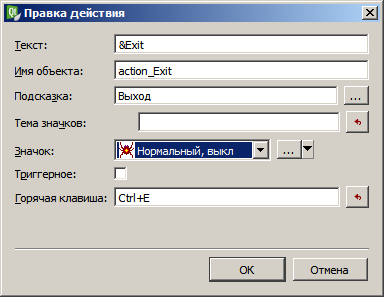
* Сделайте активной закладку «Редактор действий»
* Для редактирования созданного действия double-кликните в соответствующей строке
* Откроется диалоговое окно:



Вкотором можно задать текст, всплывающую подсказку, тип действия (триггерное/обычное) и акселератор.



Или



В результате в классе Ui\_MainWindow :

classUi\_MainWindow

{

public:

**QAction\*action\_Exit;**

…

voidsetupUi(QMainWindow\*MainWindow)

{

action\_Exit=newQAction(MainWindow);

action\_Exit->setObjectName(QStringLiteral("action\_Exit"));

action\_Exit->setCheckable(true);

//Следующие строчки зависят от Ваших установок, которые Вы указали при редактировании действия. Я –проассоциировала с действием иконку, хранящуюся в ресурсах приложения

QIconicon;

icon.addFile(QStringLiteral(":/AllMyImages/icon1.ico"),QSize(),QIcon::Normal,QIcon::Off);

MainWindow->setWindowIcon(icon);

action\_Exit->setIcon(icon);

### Соединение действия со слотом

Осталось вызвать слот MainWindowclose() при выборе пользователем пункта меню “Exit”. У действия (QAction)есть сигнал triggered(), у QMainWindowесть слот close(). Но! установить соединение визуально НЕвозможно, так как объект действие – это абстракция (у него нет визуального представления) =>

* Можно их связать (connect() ), пользуясь редактором действий
* ИЛИ! Написать connect “руками” (например, в конструкторе MainWindow)

Учимся пользоваться редактором действий:

* Открыть закладку «редактор сигналов/слотов»
* Нажать «+»
* Выбрать отправителя и сигнал
* Выбрать получателя и слот

В результате в файл ui\_mainwindow.hбудет добавлен

QObject::connect(action\_Exit,SIGNAL(triggered()),MainWindow,SLOT(close()));

Проверьте работоспособность пункта меню

## Tool Bar

Панелей инструментов (toolbar)(в отличие от главного меню) может быть несколько. Сгенерированный Qt класс Ui\_MainWindow предоставляет поддержку одного (главного) toolbar-а посредством

classUi\_MainWindow

{

public:

…

QToolBar\*mainToolBar;

…

voidsetupUi(QMainWindow\*MainWindow)

{

…

toolBar=newQToolBar(MainWindow);

toolBar->setObjectName(QStringLiteral("toolBar"));

MainWindow->addToolBar(Qt::TopToolBarArea,toolBar);

menuBar->addAction(menu\_File->menuAction());

menu\_File->addAction(action\_Exit);

toolBar->addAction(action\_Exit);

retranslateUi(MainWindow);

Если объект-действие уже существует, то для создания соответствующей ему кнопки на toolbar-е достаточно:

* Сделайте активной закладку «Редактор действий»
* «Перетащите» существующее действие на строку toolbar-а



Если объект-действие не существует:

* Сделайте активной закладку «Редактор действий»
* Создайте новое действие – откроется такой же диалог
* «Перетащите» новое действие на строку toolbar-а

В любом случае

Замечание: точно также можно было поступить с меню (сначала создать действие, а потом «перетащить» его на строку меню)

## Завершение приложения по кнопке “Exit”

Так как объект-действие для совершения требуемой операции уже создан, можно его использовать, делегируя ему выполнение операции. Создадим соединение **с помощью редактора сигналов и слотов**.

* Переключитесь в режим дизайнера
* Сделайте активной закладку «Редактор сигналов и слотов»
* Добавьте новое соединение – нажмите «+»
* В колонке «отправитель» выберите имя объекта-кнопки (pushbutton), в колонке «сигнал» - clicked(), в колонке «получатель» - action\_Exit, в колонке «слот» - trigger()

В результате в файл ui\_mainwindow.hбудет добавлено:

QObject::connect(pushButton,SIGNAL(clicked()),action\_Exit,SLOT(trigger()));

## Статусная строка

Статусная строка (подсказка внизу главного окна) поддерживается классом QStatusBar. Строка состоит из трех частей:

* Temporary – занимает бОльшую часть статусной строки и предназначена для вывода подсказки или информацию о состоянии приложения
* Normal – располагается справа от Temporaryи может быть ее перекрыта
* Permanent – присутствует всегда (если строка видна) и обычно используется для индикаторов

В файле Ui\_MainWindow.hQtгенерирует для поддержки статусной строки:

class Ui\_MainWindow

{

public:

…

QStatusBar\*statusBar;

…

void setupUi(QMainWindow\*MainWindow)

{

…

statusBar=newQStatusBar(MainWindow);

statusBar->setObjectName(QString::fromUtf8("statusBar"));

MainWindow->setStatusBar(statusBar);

…

}

Так как исходно мы никакой подсказки не задавали, то строка пока пуста.

Начальное значение подсказки легче всего задать в свойствах главного окна – statusTip, например: «Ничего не делаю…». Тогда в класс Ui\_MainWindow будет добавлено:

void retranslateUi(QMainWindow\*MainWindow)

{

…

#ifndef QT\_NO\_STATUSTIP

MainWindow->setStatusTip(QApplication::translate("MainWindow","\320\275\320\270\321\207\320\265\320\263\320\276\320\275\320\265\320\264\320\265\320\273\320\260\321\216...",0,QApplication::UnicodeUTF8));

#endif//QT\_NO\_STATUSTIP

Важно: Подсказка, заданная посредством Statusbar – возникает только тогда, когда курсор находится в клиентской области главного окна

Для изменения Temporary части статусной строки (подсказки) в процессе выполнения программы можно использовать слот   
QStatusBar::showMessage(constQString&*message*, int*timeout* = 0)

Задание: при изменении «масштаба» изображения новое значение «масштаба» должно отображаться в StatusBar

## Окно сообщений – QMessageBox

Простейший модальный диалог, функциональность которого поддерживается классом Qt – QMessageBox. Сначала нужно решить, где (в методе какого класса) удобно/логично создавать объект QMessageBox? Лучшим кандидатом является главное окно (MainWindow), так как оно является контейнером в частности для меню иtool-bar-а.

# Создаем действие

* Создайте в главном меню выпадающее меню, например «Information»
* А в нем пункт меню, например, «About»
* При этом дизайнер автоматически создает

QAction\*actionAbout;

voidsetupUi(QMainWindow\*MainWindow)

{

…

actionAbout=newQAction(MainWindow);

actionAbout->setObjectName(QStringLiteral("actionAbout"));

…

menuBar->addAction(menu\_File->menuAction());

menuBar->addAction(menu\_Information->menuAction());

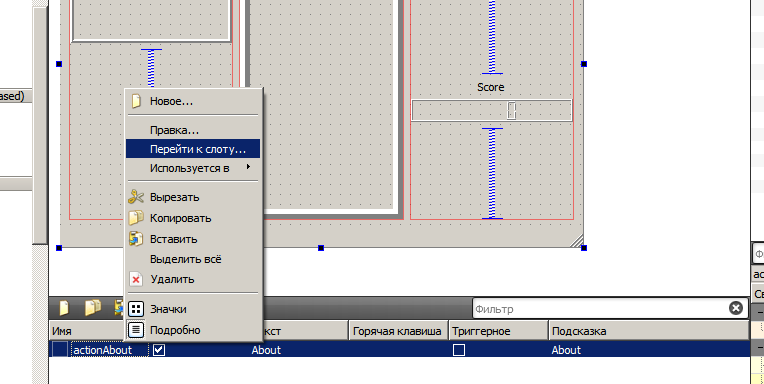
menu\_Information->addAction(actionAbout);

…

}

# Автоматизируем создание слота для обработки действия

* Перейти в режим дизайнера
* Выбрать закладку «редактор действий»
* Сделать активным действие actionAbout
* Нажать правую кнопку мыши и в контекстном меню выбрать «Перейти к слоту»



* При этом
  + автоматически создается слот в объекте-владельце действия (в нашем случае это MainWindow)с именем, которое дизайнер генерирует, используя название действияon\_actionAbout\_triggered()
  + аметодQMetaObject::connectSlotsByName(MainWindow); связывает действие со слотом во время выполнения

Соединение установлено!

# Собственно вывод сообщения

Заполняем созданную дизайнером заглушку

voidMainWindow::on\_actionAbout\_triggered()

{

}

Окна сообщений поддерживаются классом QMessageBox.

Способы использования класса QMessageBox:

1. Для отображения простенького окна сообщений достаточновызвать статический метод QMessageBox::about(). В этом методе Вы можете задать только владельца MessageBox, заголовок и собственно текст сообщения.  
   void QMessageBox::**about**([QWidget](qthelp://org.qt-project.qtgui.550/qtcore/qt.html) \**parent*, const [QString](qthelp://org.qt-project.qtgui.550/qtgui/qpixmap.html) &*title*, const [QString](qthelp://org.qt-project.qtwidgets.502/qtcore/qstring.html) &*text*)[static]  
     
   или  
   StandardButtonQMessageBox::information(QWidget\*parent,constQString&title,

constQString&text,StandardButtonsbuttons=Ok,

StandardButtondefaultButton=NoButton)[static]

* Чтобы использовать дополнительные возможности QMessageBox, требуетсяСоздать объект типа QMessageBox. Замечания:
  + В конструкторе  
    QMessageBox( QWidget \* *parent* = 0 )  
    в качестве параметра можно задать окно, которое будет являться владельцем MessageBox или 0 по умолчанию (тогда владельцем будет Desktop). При этом будет создаваться окно сообщения с кнопкой “OK”
  + конструктор  
    QMessageBox( Icon *icon*, constQString& *title*, constQString& *text*, StandardButtons *buttons* = NoButton, QWidget \* *parent* = 0, Qt::WindowFlags *f* = Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint)  
    позволяет задать иконку + текст заголовка + текст сообщения + набор кнопок +…
* для модального вызова диалога используется метод QMessageBox::exec()

# Дополнительное задание

### Использование события wheelEvent на пользовательском виджете.

**void** QWidget::**wheelEvent**([**QWheelEvent**](qthelp://org.qt-project.qtgui.532/qtgui/qpixmap.html) \**event*);

События колесика мышки отправляются виджету, который находится под курсором мыши. Если прокрутить колесико мышки, находясь над spinBox или slider, то значения в этих элементах управления будут изменяться.

В том случае, если виджет обрабатывает это событие, то для получения дельты события колеса можно использовать метод angleDelta (), который возвращает дельту в градусах колеса.

**[Q](../qtcore/qpoint.html)Point** QWheelEvent::**angleDelta**() const.

Метод возвращает относительную величину поворота колеса в восьмых долях градуса.

Положительное значение означает, что колесо было повернуто вперед от пользователя; отрицательное значение указывает, что колесо было повернуто назад к пользователю.

Большинство типов мышек работают с шагом 15 градусов, и в этом случае значение дельты кратно 120; т.е. 120 единиц \* 1/8 = 15 градусов.

Нам хотелось бы менять масштаб изображения картинки, прокручивая колесико мышки над ней.

Можно рассчитать «дельту», например, таким образом:

int delta=event->angleDelta().y()/8/15; //==1 или -1.

А можно в нашем случае поступить более грубо. Проверять знак результата метода angleDelta()и изменять масштаб на 1 или -1;

Если Вы намерены обрабатывать это событие в нашем пользовательском виджете, то после обработки следует вызвать метод QEvent::accept (), если событие обработано.

Вы должны вызвать QEvent::ignore (), если не обрабатываете событие wheel; это гарантирует, что событие будет отправлен родительскому виджету.

Для того, чтобы об измененном значении масштаба стало известно элементам управления, можно в классе пользовательского виджета объявить сигнал, а в обработчике события **wheelEvent** сгенерировать этот сигнал.