

ПРОГРАММИРОВАНИЕ В QT 5



Оглавление

Введение	3
Урок № 1. Установка среды разработки Qt	5
Урок № 2. Создание первой программы	13
Урок № 3. Создание приложения в Qt Designer	21
Урок № 4. Работа с OpenGL	29
Урок № 5. Работа с мультимедиа.....	38
Урок № 6. Работа с сетью	47
Урок № 7. Работа с потоками	55
Урок № 8. Работа с базами данных	63

Введение

Qt представляет собой комплексную среду разработки программ на языке программирования C++, предназначенную для создания кроссплатформенных приложений с графическим пользовательским интерфейсом по принципу «написал программу – компилируй ее в любом месте». Qt позволяет программистам использовать один и тот же исходный код приложений, которые будут работать в средах от Windows 98 до Vista, Mac OS X, Linux, Solaris и во многих других операционных системах.

Qt создал себе репутацию средства разработки кроссплатформенных приложений, но благодаря своему интуитивному и мощному программному интерфейсу во многих организациях Qt используется для одноплатформенных разработок. Многие сложные системы программного обеспечения на таких рынках, как средства анимации 3D, цифровая обработка фильмов, автоматизация проектирования электронных схем, разведка нефтяных и газовых месторождений, финансовые услуги и формирование изображений в медицине, строятся при помощи Qt. Если свои средства к существованию вы получаете благодаря успешному программному продукту для Windows, который создан при помощи Qt, вы можете легко создать новые рынки для систем Mac OS X и Linux просто путем перекомпиляции программного продукта.

На сегодняшний день Qt – это продукт, широко используемый разработчиками всего мира. Из числа некоторых активных пользователей Qt можно назвать такие известные компании, как: Adobe, AT&T, Cannon, HP, Bosch, Boeing, IBM, NASA, NEC, Pioneer, Sharp, Siemens, Sony, Xerox и многие другие.

Используя ту или иную программу можно не догадаться, что при её разработке использовалась библиотека Qt. Ниже перечислены самые известные проекты, созданные с применением этой технологии:

- рабочий стол KDE, используемый в ОС Linux;
- редактор трёхмерной графики Autodesk Maya;
- интернет-мессенджер Skype;
- программа для обработки изображений Adobe Photoshop Album;
- сетевая карта мира Google Earth;
- программа для виртуализации операционных систем VirtualBox;
- медиа плеер VLC.

Предоставляемая система расширений позволяет создавать модули, расширяющие функциональные возможности ваших приложений. Кроме того, использование библиотеки возможно с другими языками программирования. Соответствующие модули есть для языков C#, Java, Python и Perl.

Qt прекрасно документирована, благодаря чему всегда можно почерпнуть любую интересующую информацию об этой библиотеке, используя программу Qt Assistant. Qt — это библиотека с открытым исходным кодом, это значит, что можно в любое время взглянуть и детально разобраться в том, как работает та или иная часть этой библиотеки.

Qt — прекрасный и несложный в освоении инструмент для разработки графических кроссплатформенных приложений. Возможности данной библиотеки отлично подходят для создания сложных профессиональных программ.

Урок № 1

Установка среды разработки Qt.

Цель урока: Установить средства разработки Qt и изучить инструменты, входящие в состав библиотеки.

Теоретическая часть

Qt (произносится как *кьют*) — кроссплатформенный инструментальный разработки ПО на языке программирования C++. Qt позволяет запускать написанное с его помощью приложения в большинстве современных операционных систем путём простой перекомпиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно – ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

Существуют версии библиотеки для Microsoft Windows, систем класса UNIX с графической подсистемой X11, Android, iOS, Mac OS X, Microsoft Windows CE, QNX, встраиваемых Linux-систем и платформы S60. Идет портирование на Windows Phone и Windows RT. Также идёт портирование на Haiku и Tizen.

До недавнего времени библиотека Qt также распространялась ещё в одной версии: *Qt Embedded*. Теперь эта платформа переименована в *Qt Core* и распространяется как отдельный продукт. Qt Core обеспечивает базовую функциональность для всей линейки платформ, предназначенных для разработки приложений для встраиваемых и мобильных устройств (КПК, смартфонов и т. п.).

Отличительная особенность Qt – использование *Meta Object Compiler (MOC)* – предварительной системы обработки исходного кода (на самом деле, Qt — это библиотека не для чистого C++, а для его особого наречия, с которого и «переводит» MOC для последующей компиляции любым стандартным C++ компилятором). MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как *слоты* и *сигналы*. Кроме того, это позволяет сделать код более лаконичным. Утилита MOC ищет в заголовочных файлах на C++ описания классов, содержащие специальные макросы, и создаёт дополнительный исходный файл на C++, содержащий *метаобъектный* код.

Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть Qt Linguist — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки. В инструментарий также входит Qt Assistant — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кроссплатформенную справку для разрабатываемых на основе Qt приложений. В комплект Qt входит среда разработки Qt Creator, которая включает в себя редактор кода, справку, графические средства Qt Designer и инструменты отладки приложений. Qt Creator может использовать GCC или Microsoft Visual C++ в качестве компилятора и GDB в качестве отладчика. Для Windows библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW.

Библиотека разделена на несколько модулей, ниже перечислены основные из них:

- *QtCore* — классы ядра библиотеки, используемые другими модулями;
- *QtGui* — компоненты графического интерфейса;
- *QtNetwork* — набор классов для сетевого программирования;
- *QtOpenGL* — набор классов для работы с OpenGL;
- *QtSql* — набор классов для работы с базами данных;
- *QtScript* — классы для работы с Qt Scripts;
- *QtSvg* — классы работы с данными Scalable Vector Graphics (SVG);
- *QtXml* — модуль для работы с XML;
- *QtDesigner* — классы для создания своих собственных виджетов;
- *QtUiTools* — классы для обработки в приложении форм Qt Designer;
- *QtAssistant* — справочная система;
- *QtTest* — классы для работы с модульными тестами;
- *QtWebKit* — модуль WebKit, интегрированный в Qt;
- *Phonon* — модуль для поддержки воспроизведения и записи видео и аудио;
- *ActiveQt* — модуль для работы с ActiveX и COM под Windows.

Следует заметить, что в настоящее время библиотека активно развивается, регулярно появляются новые версии и происходит расширение функциональности. Большое внимание уделяется и производительности библиотеки, так что Qt заслуженно является одним из самых популярных фреймворков в мире разработки программного обеспечения.

Практическая часть

1. Для того, чтобы приступить к работе с Qt, необходимо установить весь необходимый инструментарий на локальный компьютер. Установка возможна в двух режимах: веб-установка, при которой выбранные инструменты постепенно загружаются и устанавливаются на компьютер, и обычная установка, с явной загрузкой установочного пакета и последующей распаковкой. Первый способ предпочтительней, так как в этом случае загружаются только выбранные компоненты библиотеки. Ниже будет рассмотрен именно этот режим. Итак, чтобы начать процедуру установки, нужно зайти на страницу загрузки библиотеки qt.io/download-open-source и нажать на кнопку «Download Now» (рис. 1). Небольшая программа установки будет загружена на компьютер.

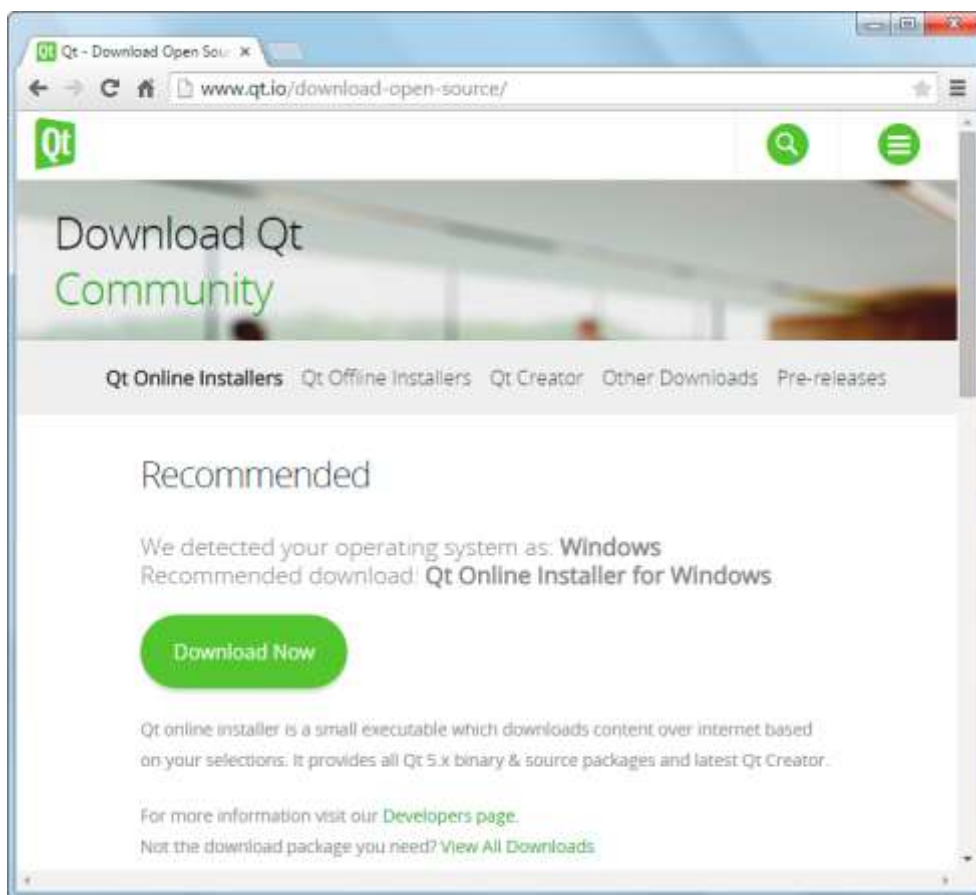


Рис. 1. Страница загрузки последней версии Qt.

2. Далее необходимо запустить загруженную программу на выполнение. Откроется мастер установки Qt – в его первом окне нужно нажать кнопку «Далее». Программа установки проверит доступность удалённого сервера загрузки компонентов библиотеки (рис. 2).

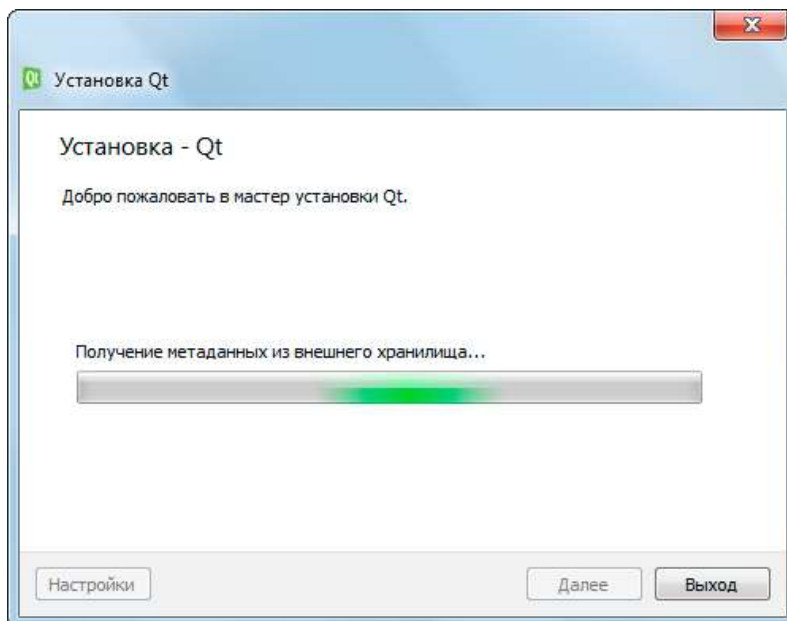


Рис. 2. Окно приветствия мастера установки Qt.

3. В следующем окне мастера установки можно настроить путь установки библиотеки (рис. 3). Однако этот параметр лучше не менять – среди устанавливаемых инструментов присутствуют такие, которые не могут работать с путями файловой системы, в названии которых есть пробелы и символы, отличные от букв английского алфавита.

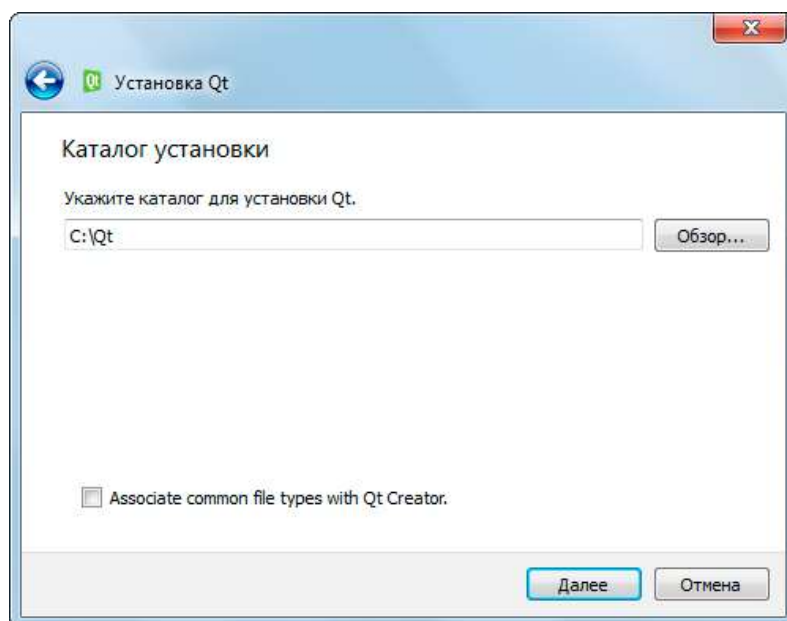


Рис. 3. Окно выбора каталога установки Qt.

4. Далее необходимо выбрать компоненты, которые будут установлены на компьютер. Чтобы установить наиболее полезные из них и одновременно сэкономить свободное пространство на жёстком диске, лучше расставить флажки так, как показано на рис. 4. В этом случае Qt будет работать на базе компилятора MinGW, который загрузится и установится автоматически.

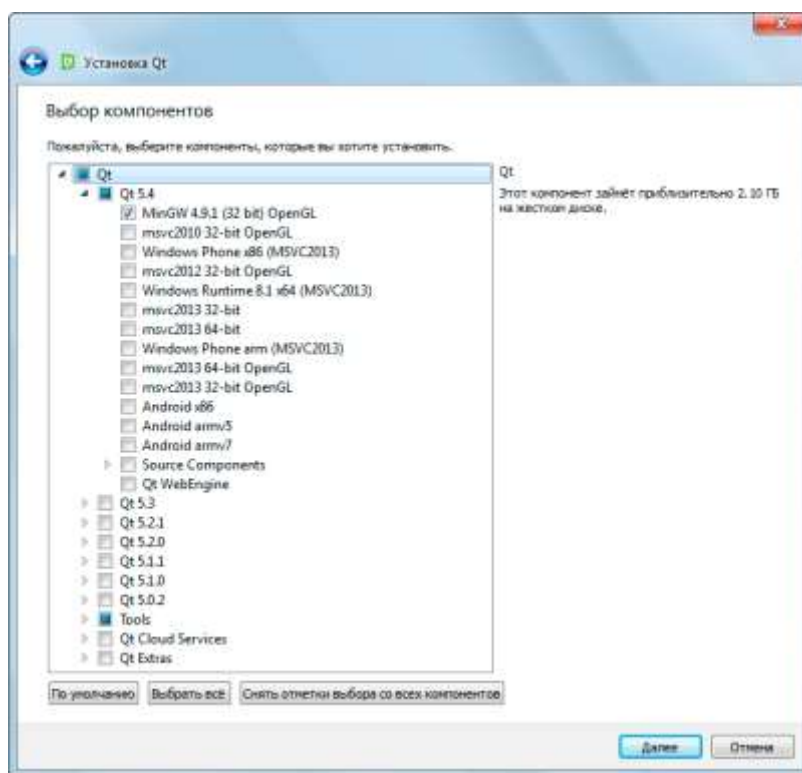


Рис. 4. Выбор компонентов Qt для установки.

5. В следующем окне мастера установки необходимо принять условия лицензионного соглашения (рис. 5). Следует помнить, что все компоненты библиотеки, которые будут загружены и установлены, распространяются на условиях свободной лицензии. Это нужно учитывать, особенно при распространении собственных программ.



Рис. 5. Окно мастера установки с текстом лицензии.

6. В последнем окне мастера установки приведена сводная информация о выбранных инструментах и зависимостях, которые также будут установлены. Для продолжения установки необходимо нажать на кнопку «Установить» (рис. 6).

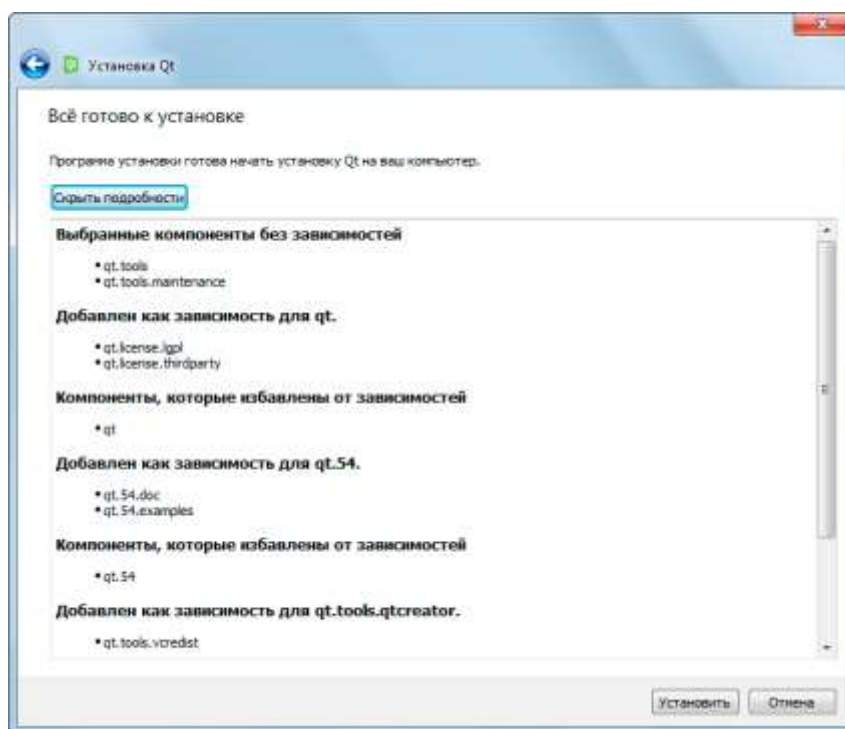


Рис. 6. Окно сводной информации о выбранных компонентах.

7. В зависимости от скорости Интернет-соединения и мощности компьютера, установка инструментов Qt может занять некоторое время. На рис. 7 показано окно с индикатором прогресса установки.

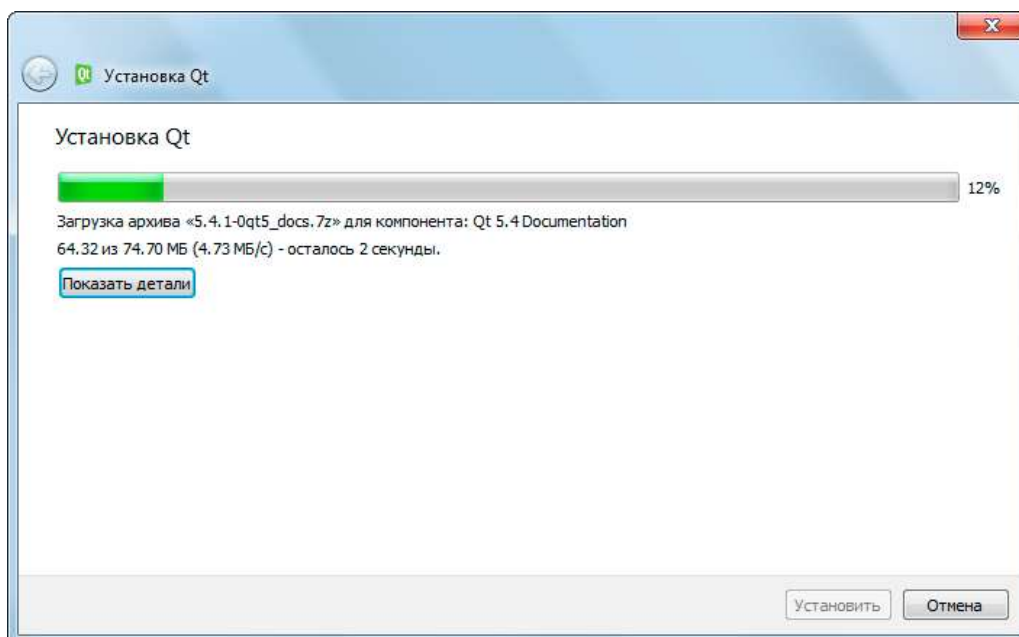


Рис. 7. Окно с индикатором прогресса установки Qt.

8. В случае удачной установки Qt на компьютер, на завершающем этапе будет предложено запустить среду разработки Qt Creator (рис. 8).

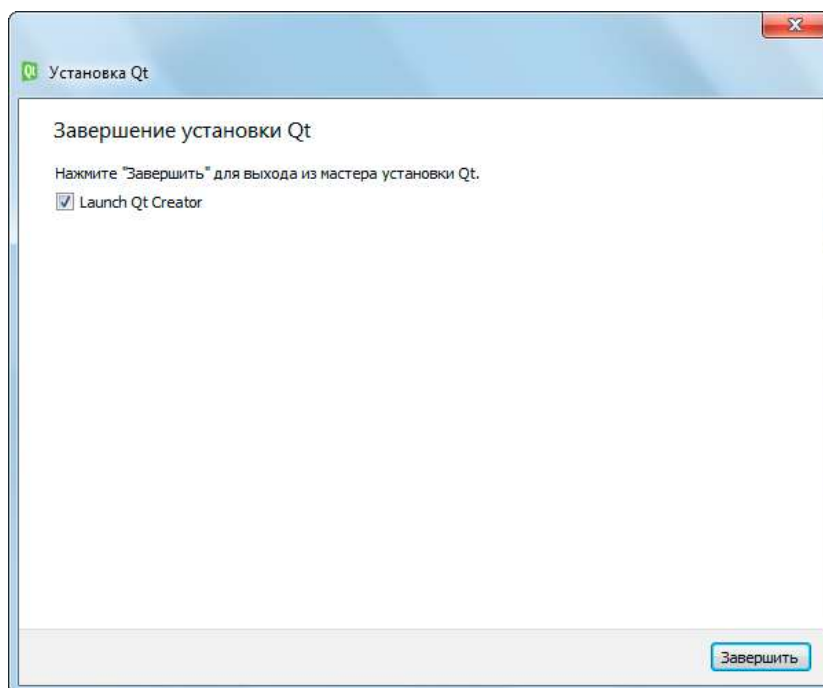


Рис. 8. Завершающее окно мастера установки.

9. Если все предыдущие команды были выполнены правильно, на компьютере запустится стартовое окно Qt Creator, как на рис. 9.



Рис. 9. Главное окно Qt Creator.

Вопросы

1. Какие основные преимущества предоставляет Qt программистам?
2. Какие функциональные модули входят в состав библиотеки?
3. Какой инструментарий входит в поставку Qt?

Урок № 2

Создание первой программы.

Цель урока: Изучить основные принципы работы с библиотекой Qt, создать первый проект графической программы, используя Qt Creator.

Теоретическая часть

Формы с использованием классов Qt могут создаваться вручную или с использованием специального инструмента Qt Designer. При создании форм вручную программист кодирует текст программы, включая по мере необходимости вызовы объектов классов Qt. При использовании Qt Designer программист графически компоует внешний вид и связи сигналов и слотов формы, а компилятор интерфейса UIС 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 MOC. MOC обрабатывает исходный текст программы, подставляя вместо специфических конструкций реализацию заказанных свойств на C++. Соответственно, на выходе MOC получается исходный код C++.

Компиляция и сборка программы осуществляется компилятором C++ и компоновщиком, доступными в рамках платформы, где осуществляется сборка проекта (рис. 10).

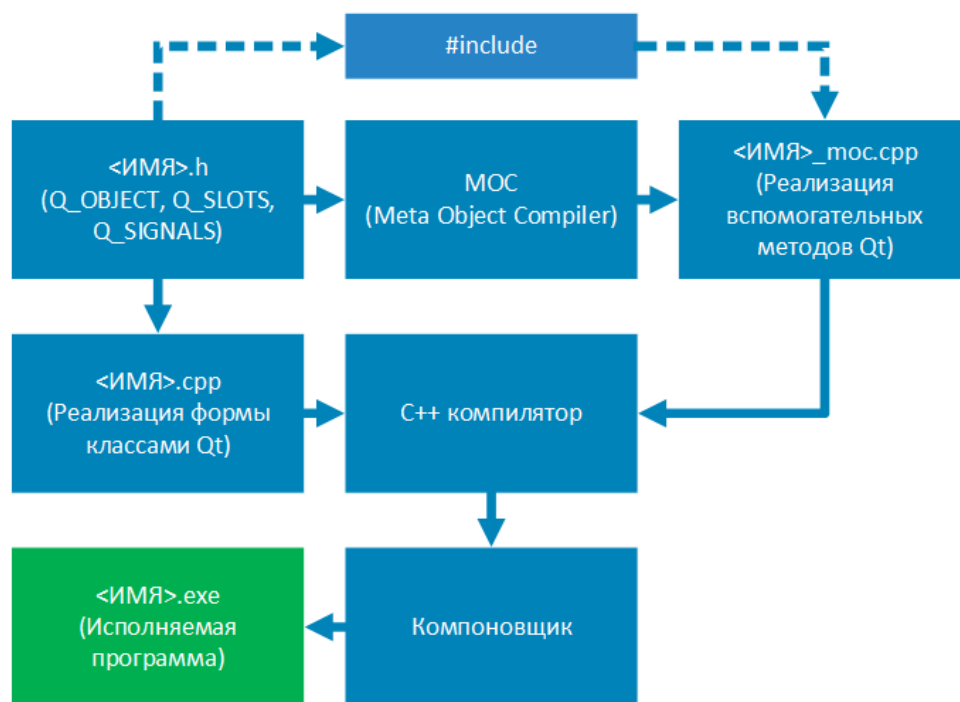


Рис. 10. Схема сборки приложения, реализованного вручную.

Ключевым механизмом взаимодействия объектов в Qt являются сигналы и слоты. Каждый объект, интегрированный в систему управления Qt, то есть описанный как `QObject`, может иметь типизированные слоты, обеспечивающие прием и обработку типизированных сигналов от других объектов, и собственные сигналы, прием которых могут осуществлять другие объекты. Связь между сигналами и слотами конкретных объектов устанавливается посредством функции `QObject::connect` (рис. 11).

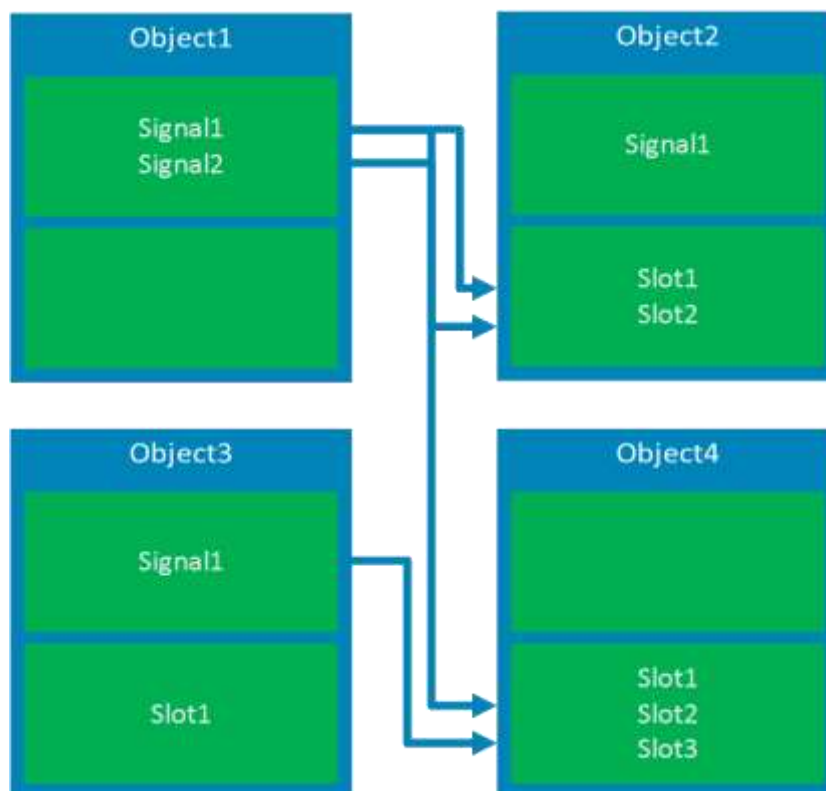


Рис. 11. Схема связывания сигналов и слотов объектов.

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

По правилам Qt, один слот может принимать несколько сигналов, а один сигнал может транслироваться на несколько слотов. Причем во взаимодействии участвуют не классы, а конкретные объекты, поэтому схема связывания сигналов и слотов может быть динамически изменена.

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

Практическая часть

1. Для того, чтобы приступить к созданию первого приложения, необходимо создать проект в Qt Creator. Для этого в главном окне среды разработки нужно нажать кнопку «Новый проект». Появится окно мастера создания проекта, как на рис. 12. При разработке на Qt программисты имеют возможность выбирать среди множества типов проектов, которые доступны в Qt Creator. Однако самым распространённым является проект с названием «Приложение Qt Widgets». Этот тип проекта предполагает использование графических окон, диалогов и виджетов Qt, а также редактор визуальной компоновки Qt Designer. Далее нужно нажать кнопку «Выбрать...».

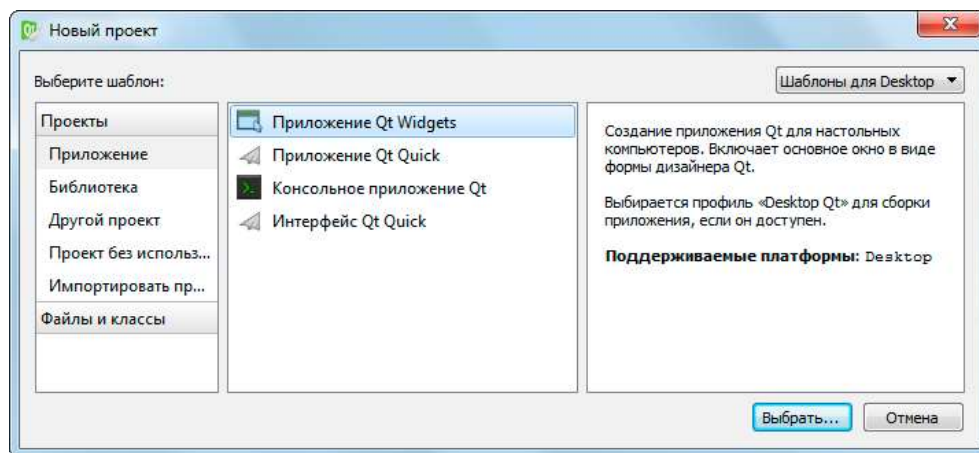


Рис. 12. Окно выбора типа проекта.

2. В следующем окне нужно указать имя проекта, а также папку на жёстком диске компьютера для размещения проекта (рис. 13). Необходимо, чтобы путь размещения файлов проекта не содержал символов, отличных от арабских цифр и букв английского алфавита. Подходящим именем этого проекта будет FirstApp.

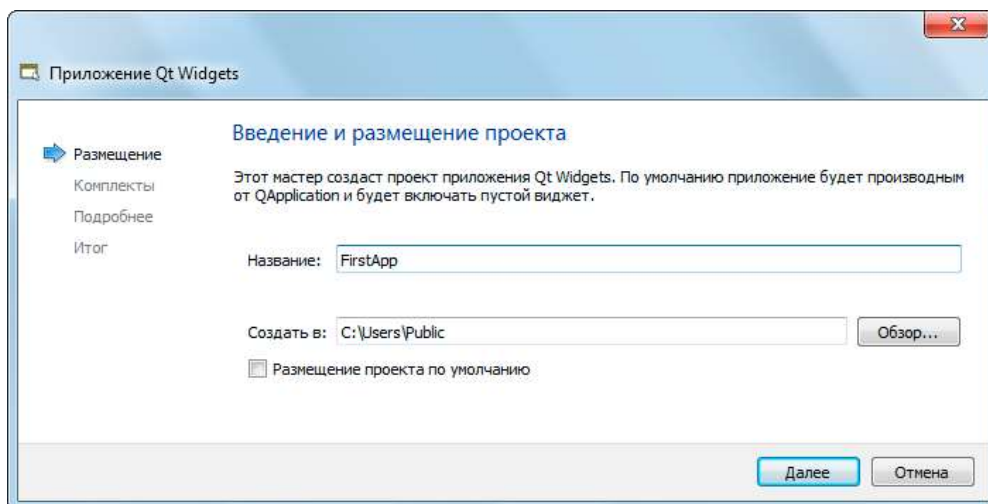


Рис. 13. Выбор имени и размещения проекта.

3. Далее мастером создания проекта будет предложено выбрать комплект сборки будущего приложения. Сборка проекта подразумевает процесс компиляции исходного кода программы в объектный файл с последующей компоновкой этого файла и вспомогательных библиотек в исполняемую программу. Необходимые комплекты сборки Qt можно выбрать на этапе установки библиотеки. В данном случае доступен компилятор MinGW. Кроме выбора системы сборки можно выбрать режимы сборки приложений: режим *отладки* и режим *выпуска*. Режим отладки полезен при использовании инструментов отладки в процессе написания исходного кода, когда необходимо проверить работу программы, постепенно выполняя код приложения и проверяя значения переменных, массивов и так далее. Данный режим позволяет выводить удобные сообщения об ошибках и различного рода исключениях, возникающих во время выполнения программы. Кроме того, во время компиляции программы в режиме отладки в объектный файл приложения помещается символьная информация об именах классов, функций и переменных. Эта информация используется при формировании сообщений и полезна только при разработке или тестировании приложения. Режим выпуска, напротив, создаёт только инструкции для выполнения программы и предназначен для сборки приложения перед окончательным выпуском, например, для продажи конечному пользователю. Этот режим создаёт, как правило, более быстрый и компактный код, так как не включает служебную информацию. На рис. 14 изображено окно мастера создания проекта с выбранным комплектом сборки.

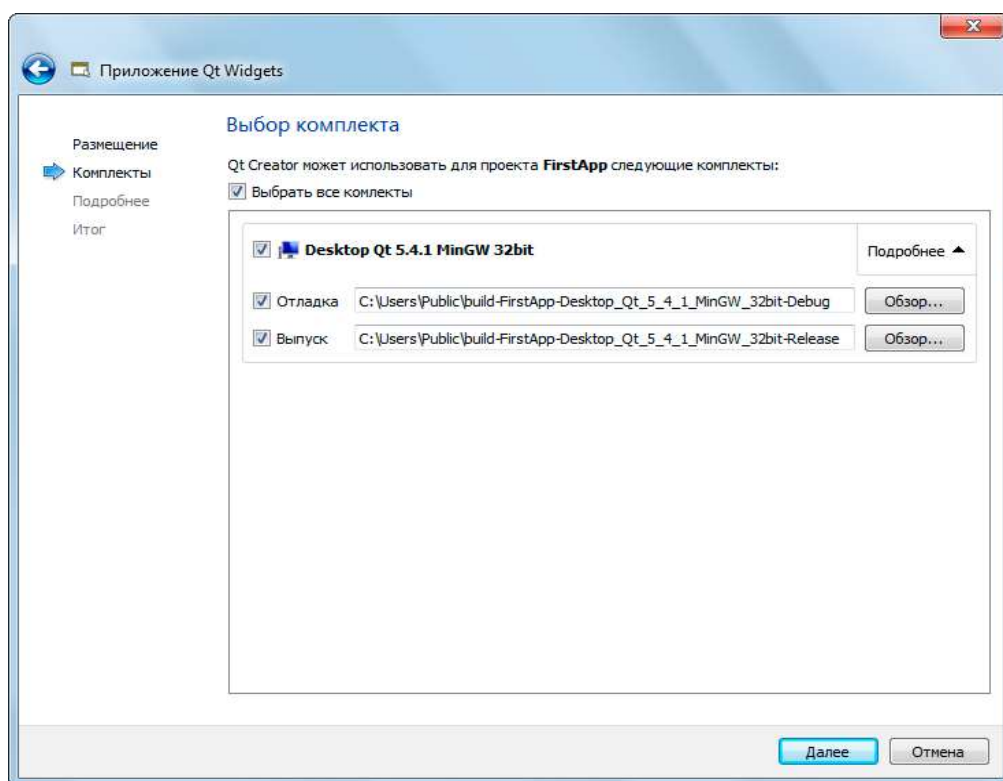


Рис. 14. Выбор комплекта сборки приложения Qt.

4. В следующем окне необходимо выбрать имя класса, реализующего графическую форму приложения Qt. Настройки, необходимые для выполнения данного урока, приведены на рис. 15.

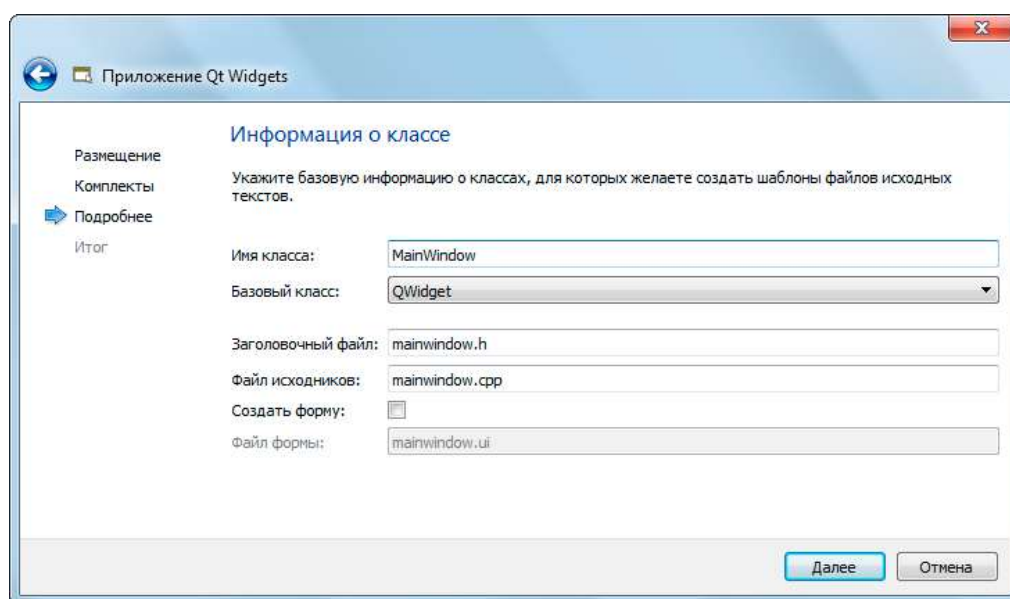


Рис. 15. Указание информации о классе формы приложения.

5. В завершающем окне мастера предлагается добавить проект в систему контроля версиями, а также приведён список файлов, которые будут созданы на данном этапе (рис. 16). В этом окне нужно нажать на кнопку «Завершить».

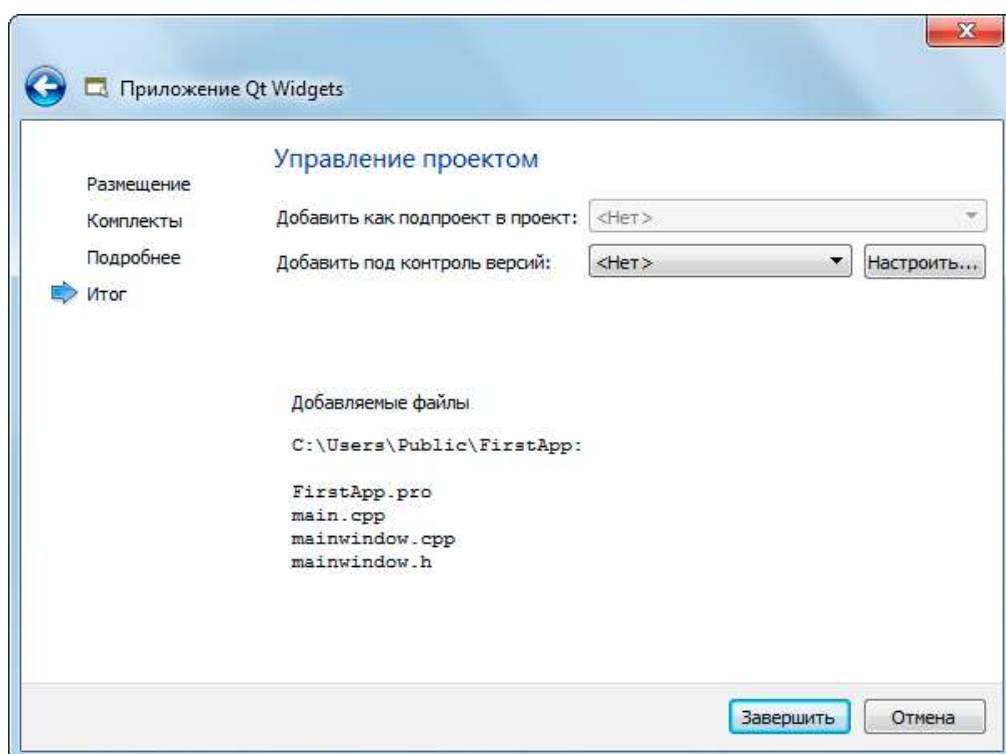


Рис. 16. Завершающий этап создания нового проекта Qt Widgets.

6. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода, а также специальный файл проекта, FirstApp.pro, который содержит настройки сборки и компиляции, не влияющие на логику программы (рис. 17). Изменять содержимое этого файла приходится нечасто и в данном уроке настраивать проект также не нужно.

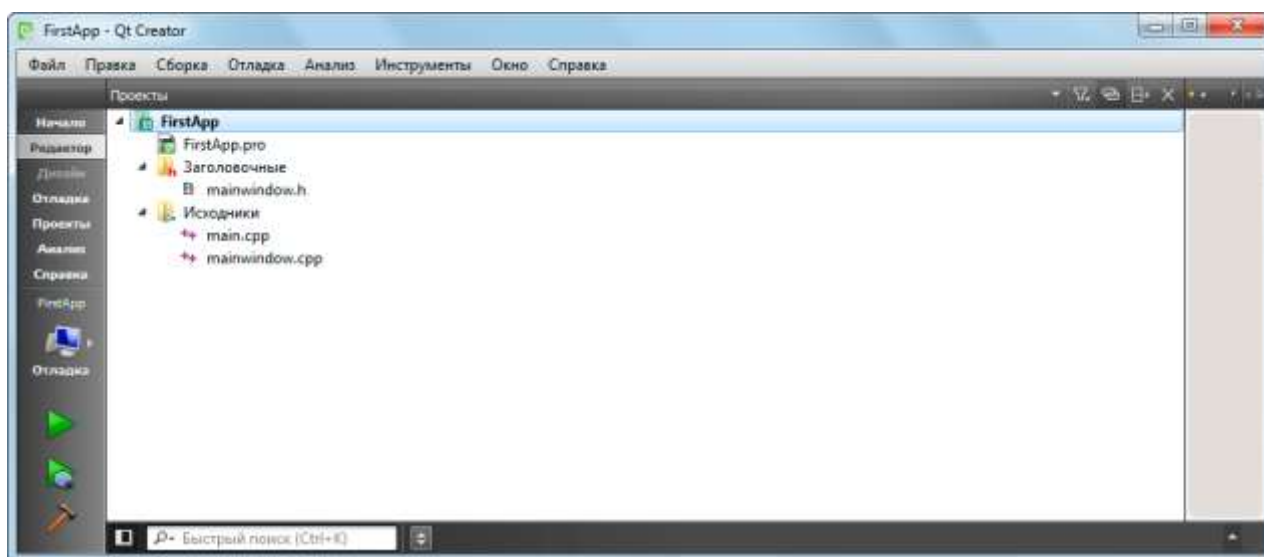


Рис. 17. Секция окна Qt Creator со структурой проекта.

7. На рис. 18 приведено объявление класса `MainWindow` из файла mainwindow.h.

```
1.  #ifndef MAINWINDOW_H
2.  #define MAINWINDOW_H
3.
4.  #include <QWidget>
5.  #include <QSlider>
6.  #include <QComboBox>
7.  #include <QSpinBox>
8.  #include <QPushButton>
9.  #include <QHBoxLayout>
10. #include <QMessageBox>
11.
12. class MainWindow : public QWidget
13. {
14.     Q_OBJECT
15.
16. public:
17.     MainWindow(QWidget* = 0);
18.
19. private slots:
20.     void ShowMessage();
21.
22. private:
23.     QSpinBox spinBox;
24.     QSlider slider;
25.     QPushButton button;
26.     QHBoxLayout layout;
27. };
28.
29. #endif
```

Рис. 18. Объявление класса формы и дочерних виджетов.

8. Файл исходного кода `mainwindow.cpp` содержит определение класса `MainWindow` и составляет основную часть программы (рис. 19). В данном примере такие элементы, как ползунок, счётчик и кнопка помещаются в так называемый *контейнер компоновки*, который выравнивает содержимое с заданными параметрами. Интуитивно понятно, что класс `QHBoxLayout` выравнивает компоненты формы по горизонтали, а два его метода `setMargin` и `setSpacing` задают отступы снаружи контейнера компоновки и между его элементами соответственно. Далее этот контейнер компоновки устанавливается для формы с помощью метода `QWidget::setLayout`. При появлении окна формы содержимое компоновки также будет выведено на экран.

```
1. #include "mainwindow.h"
2.
3. MainWindow::MainWindow(QWidget *parent) : QWidget(parent)
4. {
5.     setWindowTitle("Главное окно");           // Заголовок окна
6.     setLayout(&layout);                         // Установка выравнивания по горизонтали
7.     slider.setRange(0, 100);                    // Установка границ ползунка
8.     spinBox.setRange(0, 100);                   // Установка границ счётчика
9.     slider.setOrientation(Qt::Horizontal);       // Горизонтальное положение ползунка
10.    button.setText("Сообщение");                 // Текст кнопки
11.    layout.setMargin(5);                         // Внешние отступы (5 пикселей)
12.    layout.setSpacing(5);                       // Отступы между элементами
13.    layout.addWidget(&spinBox);                  // Добавление счётчика на форму
14.    layout.addWidget(&slider);                   // Добавление ползунка
15.    layout.addWidget(&button);                   // Добавление кнопки
16.    /*-----
17.        Установка связей между сигналами и слотами
18.    -----*/
19.    connect(&button, SIGNAL(released()), this, SLOT(ShowMessage()));
20.    connect(&slider, SIGNAL(valueChanged(int)), &spinBox, SLOT(setValue(int)));
21.    connect(&spinBox, SIGNAL(valueChanged(int)), &slider, SLOT(setValue(int)));
22. }
23.
24. void MainWindow::ShowMessage()
25. {
26.     int value = spinBox.value();                // Получение значения счётчика
27.     QMessageBox::information(this, "Сообщение", // Вывод информационного сообщения
28.         "Вы выбрали " + QString::number(value)); // QString::number – числа в строку
29. }
```

Рис. 19. Определение класса формы.

9. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы. (рис. 20). Эта функция выводит на экран созданную форму.

```
1. #include "mainwindow.h"
2. #include <QApplication>
3.
4. int main(int argc, char *argv[])
5. {
6.     QApplication a(argc, argv);                // Создание объекта приложения
7.     MainWindow w;                               // Создание объекта класса MainWindow
8.     w.show();                                   // Вывод формы на экран
9.     return a.exec();                           // Запуск приложения
10. }
```

Рис. 20. Главная функция программы.

10. Чтобы запустить программу на выполнение, необходимо нажать на кнопку «*Запустить*» слева, на панели выбора режимов, или использовать сочетание клавиш «*Ctrl + R*». Исходные файлы проекта будут скомпилированы, после чего система сборки Qt автоматически создаст исполняемый файл приложения. Через некоторое время окно программы появится на экране (рис. 21).

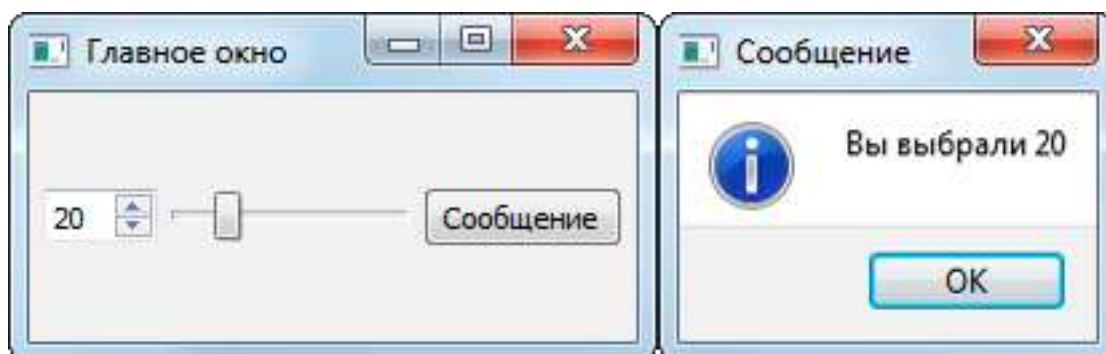


Рис. 21. Результат работы программы.

Вопросы

1. Какие способы создания форм существуют в Qt?
2. Как реализуется механизм взаимодействия объектов в Qt?
3. Какой компонент Qt расширяет возможности языка C++?

Урок № 3

Создание приложения в Qt Designer.

Цель урока: Изучить принципы работы с графической средой разработки интерфейсов Qt Designer, создать проект графической программы, используя Qt Creator и Qt Designer.

Теоретическая часть

При необходимости быстрого получения результата, проведения экспериментов при разработке пользовательского интерфейса и общей оценке выполненной работы следует использовать редактор интерфейсов Qt Designer. Формы, созданные в этом визуальном редакторе, обрабатываются специальным компилятором `uic`, который преобразует их в код программы на языке C++. Таким образом, Qt Designer не накладывает никаких ограничений на систему сборки, используемую для компиляции программ на Qt.

Сборка приложения, созданного с использованием Qt Designer, практически ничем не отличается от сборки приложения, реализованного вручную (рис. 22). Работа компилятора `uic` заключается в создании заголовочного файла с определениями всех используемых на форме виджетов – того же самого, что создал бы программист при разработке приложения без визуального редактора.

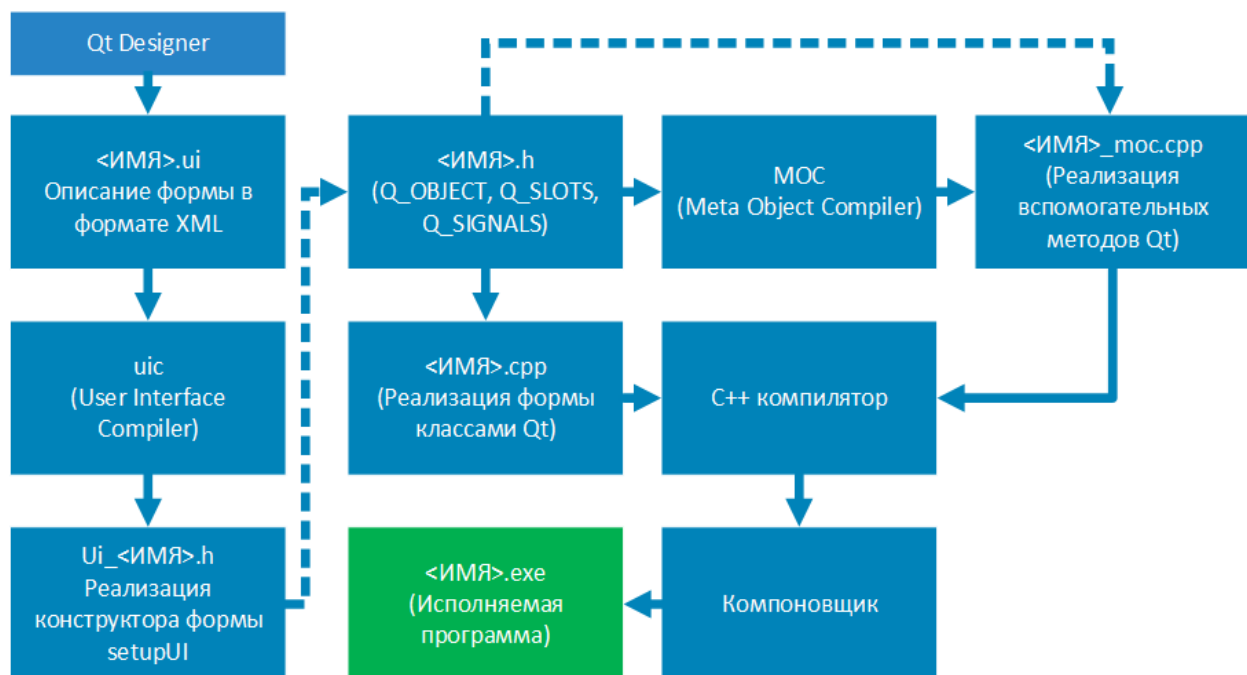


Рис. 22. Схема сборки приложения при использовании Qt Designer.

Компоненты графического интерфейса программы добавляются на форму простым перетаскиванием из панели «Панель виджетов». На панели «Инспектор объектов», отображается структура формы, а на панели «Редактор свойств» отображаются настраиваемые свойства выбранного виджета, такие как размер, стиль, название и многие другие.

Файлы графических форм можно создавать отдельно от процесса разработки самого приложения. Это позволяет работать в команде, когда дизайнеры трудятся над обликом приложения, а программисты реализуют логику программы. После создания файла графического интерфейса, разработчики могут использовать компилятор `uic` для генерации вспомогательного кода на языке `C++` и использовать этот код в программе.

Главное в использовании редактора Qt Designer – возможность широкой настройки компонентов графического интерфейса. Qt следует всем современным тенденциям разработки графических приложений – от возможности задания стилей компонентам формы до разработки собственных виджетов (рис. 23).

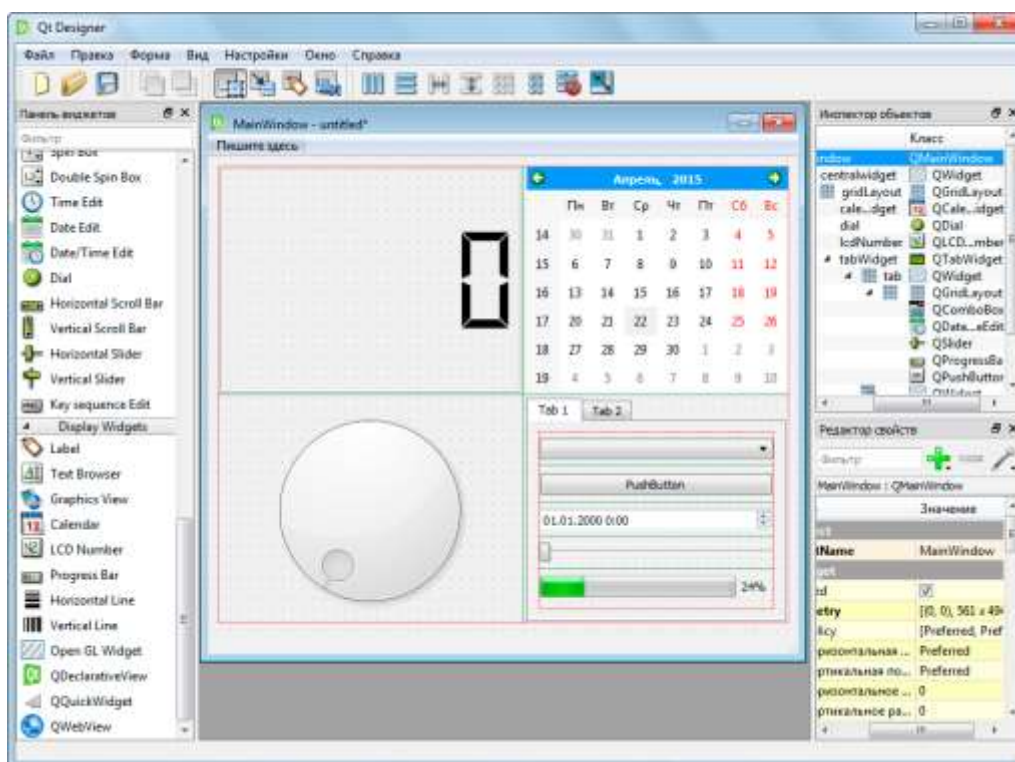


Рис. 23. Пример графического интерфейса Qt Designer.

Использование Qt совместно с Qt Designer даёт большое преимущество на всех этапах создания приложений. Также стоит отметить, что с каждой новой версией в Qt Designer появляется всё больше новых возможностей. Это способствует привлечению новых пользователей Qt и появлению качественных программ.

Практическая часть

1. Создание проекта приложения с использованием Qt Designer практически не требует дополнительных операций. Для того, чтобы создать новый проект, нужно повторить некоторые шаги из предыдущего урока (рис. 24).

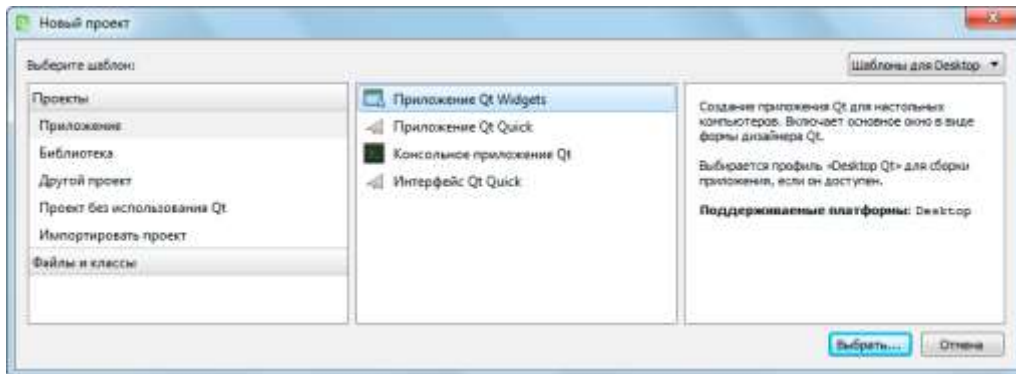


Рис. 24. Окно выбора типа проекта.

2. Следующий этап создания проекта с использованием Qt Designer также ничем не отличается от такового из предыдущего урока (рис. 25). Имя проекта – Stopwatch.

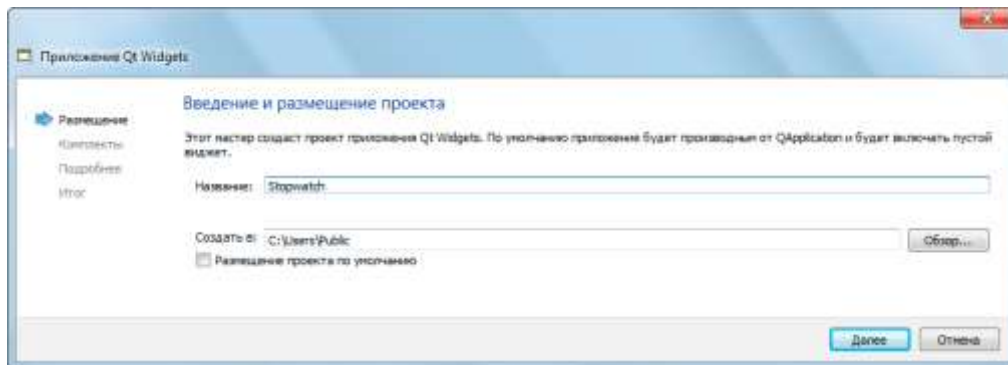


Рис. 25. Выбор имени и размещения проекта.

3. В следующем окне мастера создания проекта следует оставить параметры по умолчанию (рис. 26). Как уже было замечено, в проекте Qt Designer может быть использована любая система сборки.

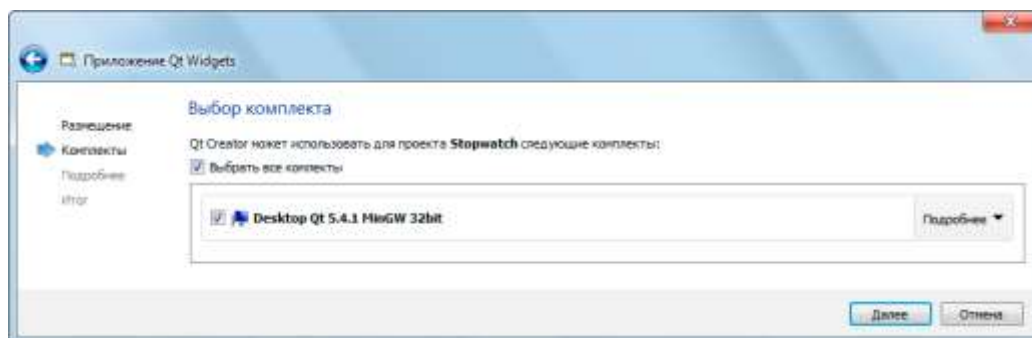


Рис. 26. Выбор комплекта сборки приложения Qt.

4. В следующем окне необходимо оставить выбранной опцию «Создать форму». Настройки, необходимые для выполнения данного урока, приведены на рис. 27.

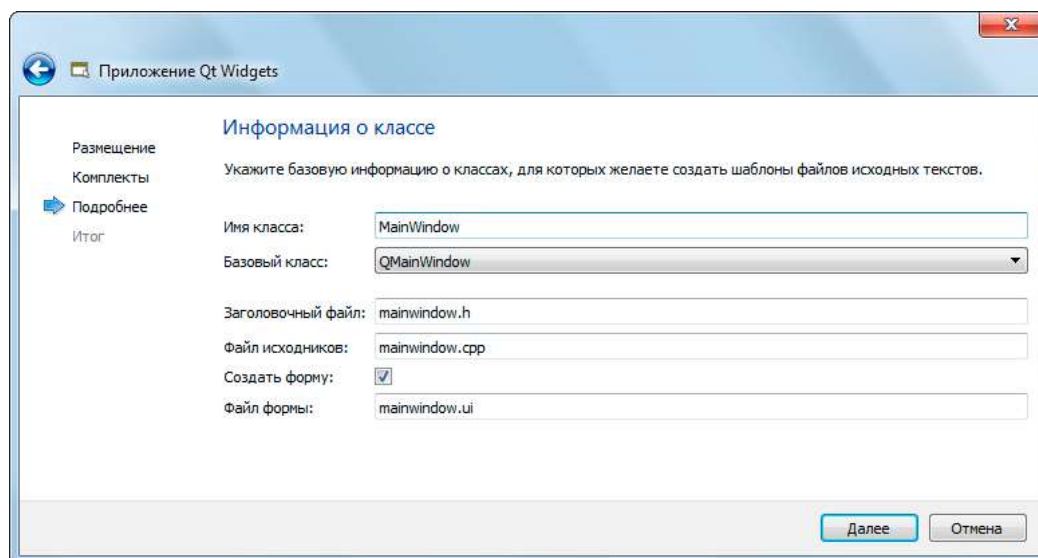


Рис. 27. Указание информации о классе формы приложения.

5. В завершающем окне мастера предлагается добавить проект в систему контроля версиями, а также приведён список файлов, которые будут созданы на данном этапе (рис. 28). В этом окне нужно нажать на кнопку «Завершить».

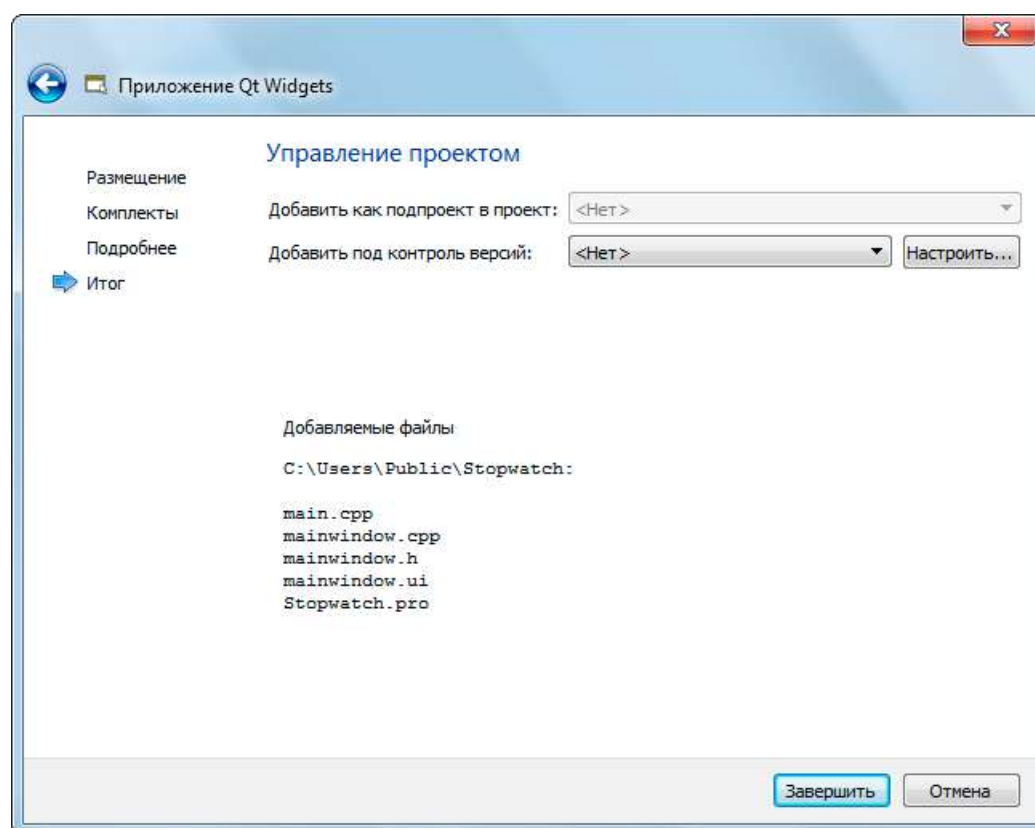


Рис. 28. Завершающий этап создания нового проекта Qt Widgets.

6. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода, файл проекта, а также файл графической формы mainwindow.ui, который управляется редактором Qt Designer (рис. 29).

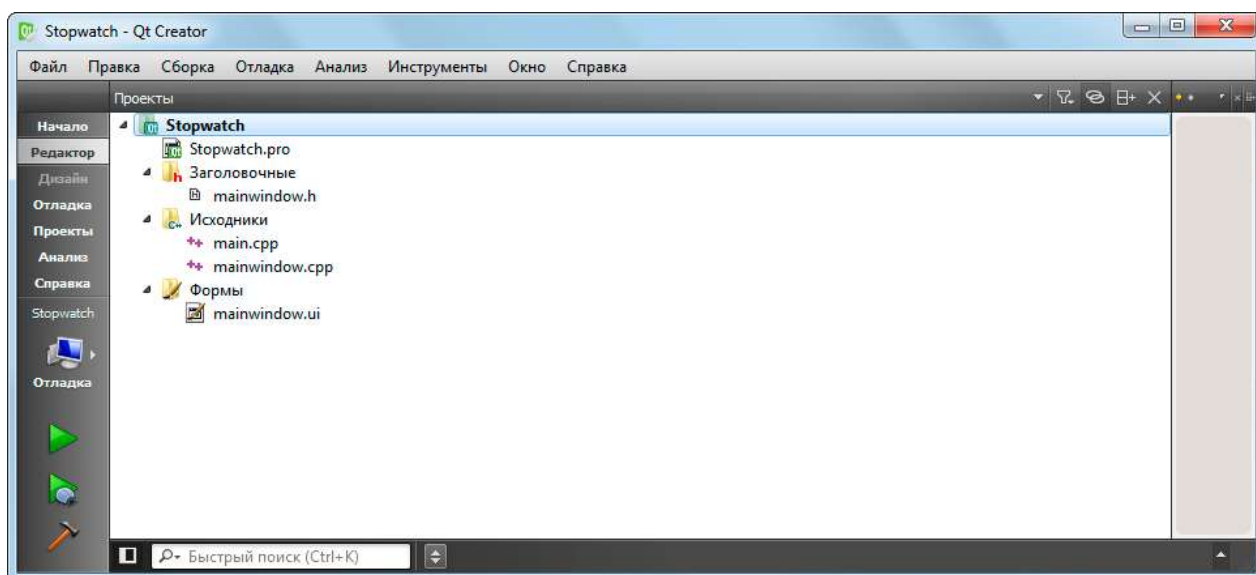


Рис. 29. Секция окна Qt Creator со структурой проекта.

7. На рис. 30 приведено содержимое файла mainwindow.ui, открытое в среде разработки интерфейсов Qt Designer.

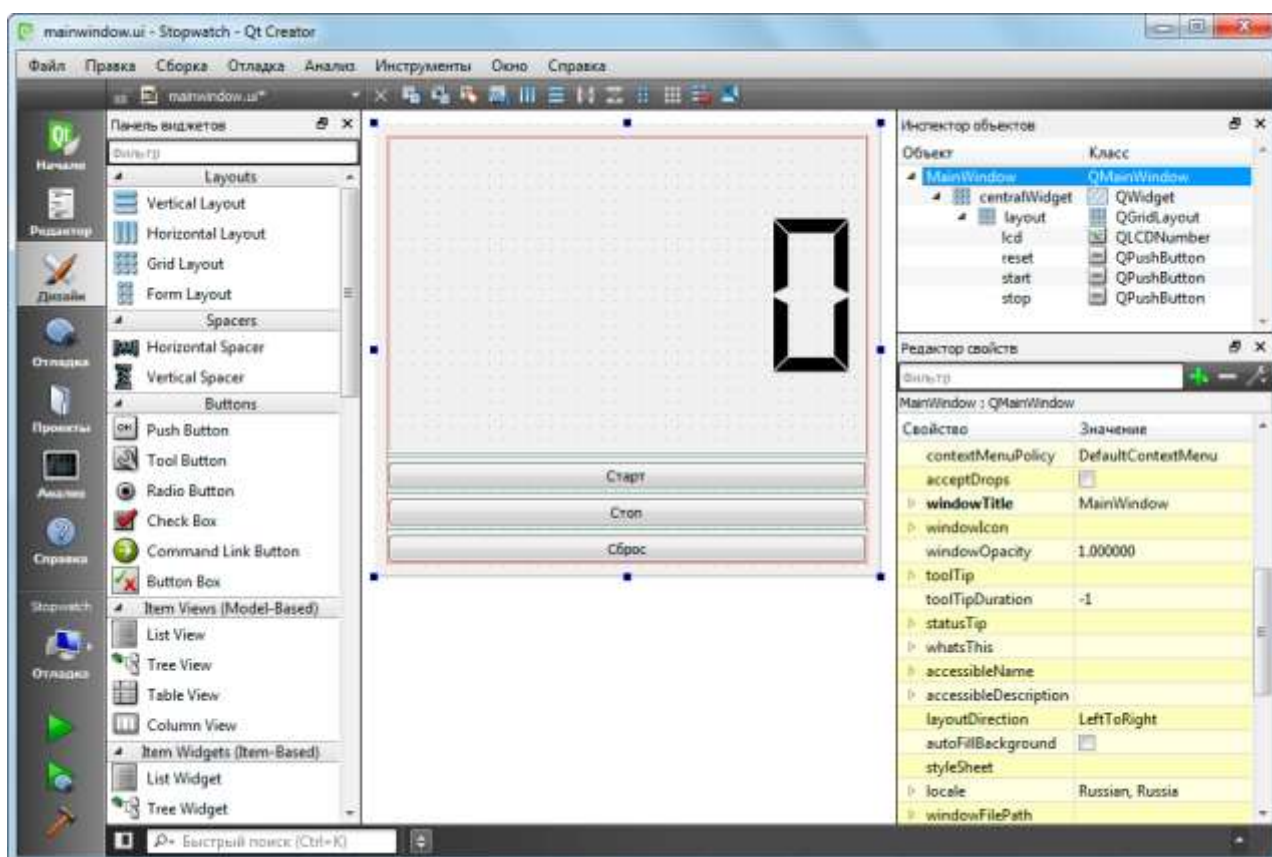


Рис. 30. Графическая форма приложения.

8. На рис. 31 приведено объявление класса `MainWindow` из файла `mainwindow.h`. Помимо объявления класса формы, этот заголовочный файл содержит объявление класса `Ui::MainWindow` – код этого класса будет сгенерирован компилятором `uic`. Также в описании класса формы используются переменные типов `QTime`, `QTimer` и `QString` – счётчик времени, объект таймера и формат времени соответственно.

```
1.  #ifndef MAINWINDOW_H
2.  #define MAINWINDOW_H
3.
4.  #include <QMainWindow>
5.  #include <QTime>
6.  #include <QTimer>
7.
8.  namespace Ui
9.  {
10.     class MainWindow;
11. }
12.
13. class MainWindow : public QMainWindow
14. {
15.     Q_OBJECT
16.
17. public:
18.     /*-----
19.         Объявление конструктора и деструктора
20.     -----*/
21.     MainWindow(QWidget* = 0);
22.     ~MainWindow();
23.
24. public slots:
25.     void Start();           // Обработчик кнопки "Старт"
26.     void Stop();           // Обработчик кнопки "Стоп"
27.     void Reset();          // Обработчик кнопки "Сброс"
28.     void Time();           // Событие таймера
29.
30. private:
31.     Ui::MainWindow *ui;    // Доступ к компонентам формы
32.     QTime time;           // Объект времени
33.     QTimer timer;         // Объект таймера
34.     QString format;       // Формат вывода времени
35. };
36.
37. #endif
```

Рис. 31. Объявление класса формы и вспомогательных объектов.

9. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы (рис. 32). Эта функция выводит на экран созданную форму.

```
1.  #include "mainwindow.h"
2.  #include <QApplication>
3.
4.  int main(int argc, char *argv[])
5.  {
6.     QApplication a(argc, argv);           // Создание объекта приложения
7.     MainWindow w;                         // Создание объекта класса MainWindow
8.     w.show();                             // Вывод формы на экран
9.     return a.exec();                     // Запуск приложения
10. }
```

Рис. 32. Главная функция программы.

10. Файл исходного кода `mainwindow.cpp` содержит определение класса `MainWindow` и составляет основную часть программы (рис. 33). В данном файле содержатся обработчики событий нажатия на кнопки «Старт», «Стоп» и «Сброс». При нажатии на кнопку «Старт» происходит запуск таймера, который каждую миллисекунду генерирует вызов функции `MainWindow::Time`. При нажатии на кнопку «Стоп» таймер останавливается. Обработчик события кнопки «Сброс» сбрасывает счётчик времени и настраивает формат вывода прошедшего времени.

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3.
4. MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent),
5.     ui(new Ui::MainWindow)
6. {
7.     ui->setUpUi(this);           // Инициализация формы
8.     Reset();                     // Сброс счётчика
9.     /*-----
10.         Установка связей между сигналами и слотами
11.     -----*/
12.     connect(ui->start, SIGNAL(released()), SLOT(Start()));
13.     connect(ui->stop,  SIGNAL(released()), SLOT(Stop ()));
14.     connect(ui->reset, SIGNAL(released()), SLOT(Reset()));
15.     connect(&timer,   SIGNAL(timeout()),  SLOT(Time ()));
16. }
17.
18. MainWindow::~MainWindow()
19. {
20.     delete ui;                  // Очистка памяти
21. }
22.
23. void MainWindow::Start()
24. {
25.     timer.start();              // Запуск таймера
26. }
27.
28. void MainWindow::Stop()
29. {
30.     timer.stop();               // Остановка таймера
31. }
32.
33. void MainWindow::Reset()
34. {
35.     format = "hh:mm:ss:zzz";    // Установка формата
36.     time = QTime(0, 0);         // Сброс счётчика
37.     QString text;               // Строковая переменная
38.     text = time.toString(format); // Форматирование счётчика
39.     ui->lcd->display(text);       // Вывод на экран
40. }
41.
42. void MainWindow::Time()
43. {
44.     /*-----
45.         Обновление счётчика времени
46.     -----*/
47.     time = time.addMSecs(1);     // Обновление счётчика
48.     QString text;               // Строковая переменная
49.     text = time.toString(format); // Форматирование счётчика
50.     ui->lcd->display(text);       // Вывод на экран
51. }
```

Рис. 33. Определение класса формы.

11. Чтобы запустить программу на выполнение, необходимо нажать на кнопку «Запустить» слева, на панели выбора режимов, или использовать сочетание клавиш «*Ctrl + R*». Исходные файлы проекта будут скомпилированы, после чего система сборки Qt автоматически создаст исполняемый файл приложения. Через некоторое время окно программы появится на экране (рис. 34).

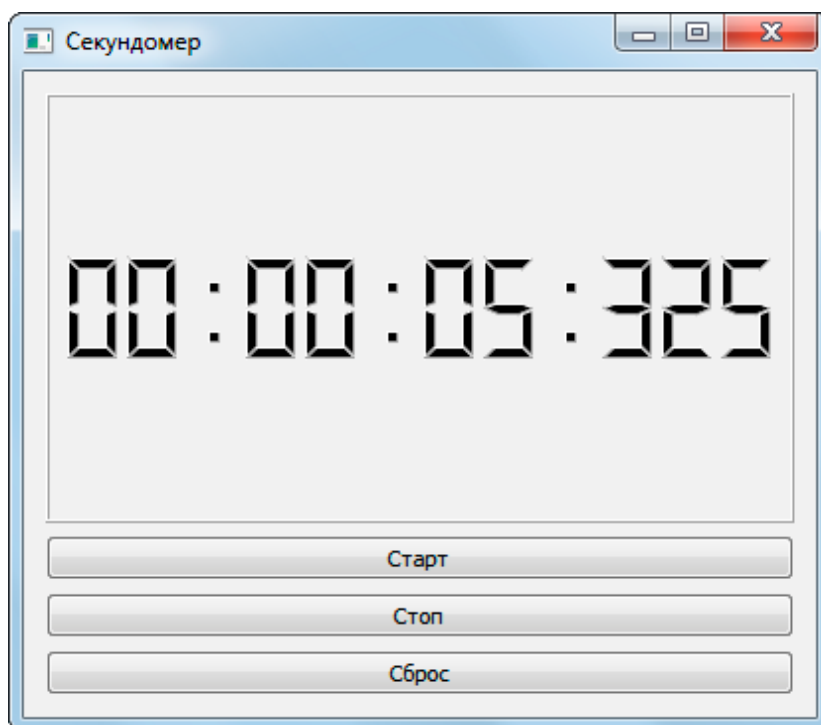


Рис. 34. Результат работы программы.

Вопросы

1. Какие основные преимущества даёт использование Qt Designer?
2. Какими возможностями обладает среда Qt Designer?
3. Как происходит сборка приложения при использовании Qt Designer?

Урок № 4

Работа с OpenGL.

Цель урока: Изучить принципы построения трёхмерных изображений в Qt, познакомиться с возможностями создания трёхмерной графики OpenGL, научиться создавать графические приложения с использованием OpenGL в среде Qt.

Теоретическая часть

Трёхмерная графика является одной из самых интересных тем в программировании, которая существует с ранних стадий развития компьютерной техники. На протяжении довольно продолжительного времени она применялась исключительно в рамках правительственных или военных проектов, таких как симуляторы полёта, моделирование и проектирование дорогостоящих двигателей, виртуальная разработка архитектуры будущих зданий, исследование земной поверхности в географических целях. С недавнего времени трёхмерная графика вышла за рамки научной деятельности и превратилась в целую индустрию, включающую в себя такие сферы, как анимация (от спецэффектов до компьютерных игр), кино, виртуальная реальность, медицина и многое другое. С каждым годом всё больше людей задействованы в этой области.

OpenGL (Open Graphics Library – открытая графическая библиотека) – это мощный программный интерфейс, применяемый для получения высококачественных, программно генерированных изображений и интерактивных приложений, использующих двухмерные и трёхмерные объекты, а также растровые изображения.

OpenGL – это графический стандарт в области компьютерной графики. Впервые этот стандарт был введён компанией Silicon Graphics в 1992 году. Разработчики OpenGL – это крупнейшие фирмы, производители как программного, так и аппаратного обеспечения: Silicon Graphics, Microsoft, IBM, DEC, Intel и другие.

OpenGL поддерживается многими популярными платформами, например Linux, Windows, Mac OS X. Программы на OpenGL можно переносить между поддерживаемыми платформами, получая при этом одинаковый результат, будь это графическая станция или суперкомпьютер. OpenGL освобождает программиста от написания программ для конкретного оборудования. Если устройство поддерживает какую-либо функцию, то эта функция выполняется аппаратно, если нет, то библиотека выполняет её программно.

С точки зрения программиста OpenGL – это программный интерфейс графических устройств, таких как графические ускорители (видеокарты). Он включает в себя множество различных команд, с помощью которых программист может определять различные объекты и производить их рендеринг. Говоря более простым языком, разработчик приложения определяет трёхмерные объекты, задаёт их местоположение в трёхмерном пространстве и другие свойства, такие как цвет, текстуры, материал, положение наблюдателя трёхмерной сцены и многое другое, а библиотека OpenGL отображает всё это на экране. OpenGL поддерживает только команды рисования и программистам приходится самостоятельно реализовывать поддержку мыши, клавиатуры или джойстика, заботиться о воспроизведении звука и видео, а также настраивать множество других подсистем приложения, без которых не обходятся современные игры.

OpenGL имеет хорошо продуманную внутреннюю структуру и довольно простой процедурный интерфейс. Несмотря на это, с помощью OpenGL можно создавать сложные и мощные программные комплексы, затрачивая при этом минимальное время по сравнению с другими библиотеками. В настоящее время существует большое количество программных систем для разработки игр, в основе которых лежит именно OpenGL. На рис. 35 показано окно приложения, в котором изображена трёхмерная сцена, выведенная на экран средствами OpenGL.

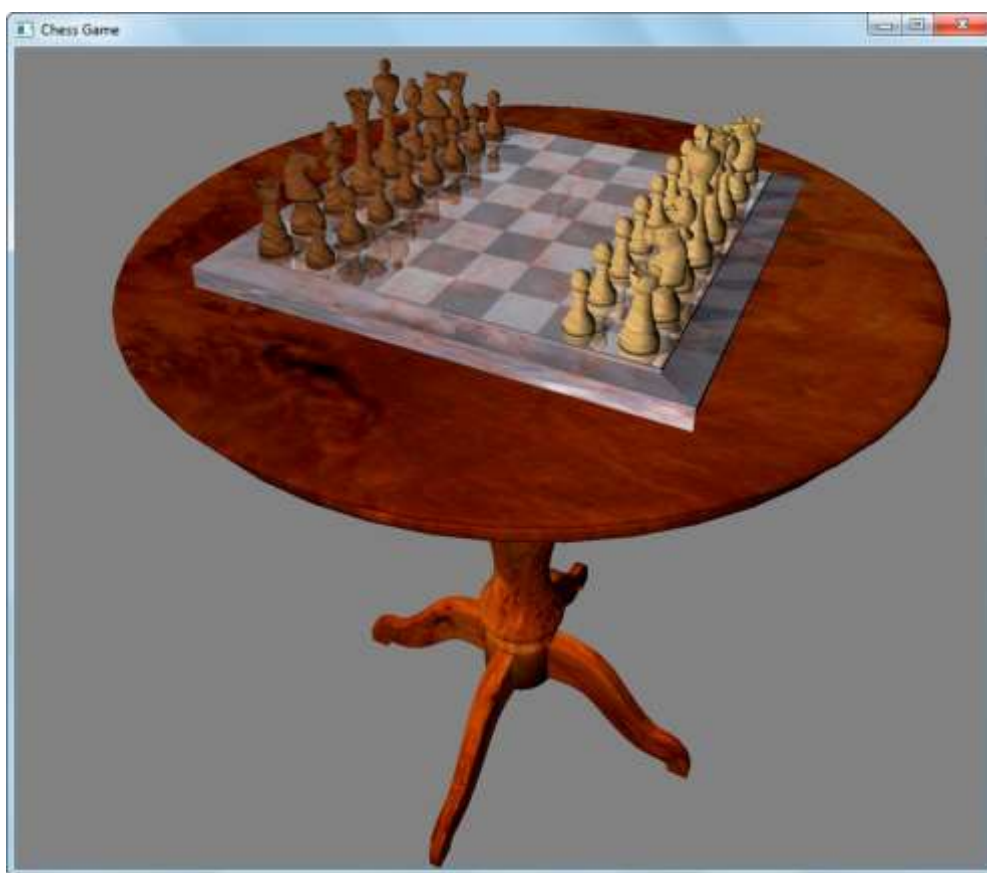


Рис. 35. Графическое приложение, созданное при помощи OpenGL.

Библиотека OpenGL не является объектно-ориентированной. При работе с библиотекой разработчик имеет дело только с функциями, переменными и константами. Имена всех функций OpenGL начинаются с букв `gl`, а констант – с `GL_`. В имена функций входят суффиксы, говорящие о количестве и типах передаваемых параметров. Например, прототип функции `glColor3f` говорит о том, что в неё должны передаваться три значения с плавающей точкой. В табл. 1 указаны суффиксы и типы, используемые в OpenGL.

Суффикс	Тип OpenGL	C++ эквивалент	Описание
b	GLbyte	char	Байт
s	GLshort	short	Короткое целое
i	GLint	int	Целое
f	GLfloat	float	С плавающей точкой
d	GLdouble	double	С плавающей точкой двойной точности
ub	GLubyte	unsigned char	Байт без знака
us	GLushort	unsigned short	Короткое целое без знака
ui	GLuint	unsigned int	Целое без знака
v			Массив

Табл. 1. Суффиксы и типы, используемые в OpenGL.

В табл. 2 перечислены основные классы Qt, которые используются для доступа к возможностям OpenGL.

Класс	Описание
QGLWidget	Специальный виджет Qt для рисования графики OpenGL
QGLBuffer	Класс для создания и управления объектами буферов OpenGL
QGLContext	Класс, инкапсулирующий контекст рендеринга OpenGL
QGLFormat	Настройки формата вывода контекста рендеринга
QGLFramebufferObject	Класс, инкапсулирующий кадровый буфер OpenGL
QGLShader	Класс для компиляции шейдеров OpenGL
QGLShaderProgram	Класс для компоновки шейдеров и использования их в программе
QGLColormap	Класс для установки собственных цветовых карт
QGLPixelBuffer	Класс, инкапсулирующий буфер пикселей OpenGL

Табл. 2. Описание основных классов OpenGL в Qt.

Практическая часть

1. Создание проекта приложения с использованием OpenGL практически не потребует никаких дополнительных шагов со стороны разработчика. Первым шагом на пути к созданию программы в Qt является выбор проекта «*Приложение Qt Widgets*» (рис. 36).

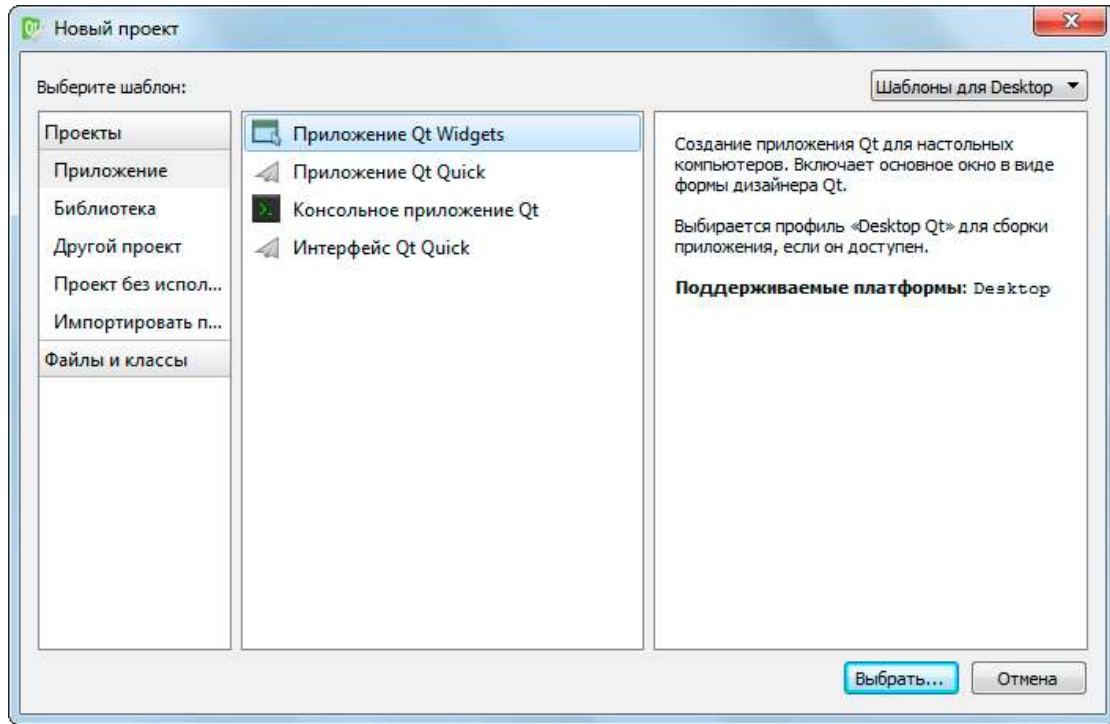


Рис. 36. Окно выбора типа проекта.

2. В следующем окне нужно указать имя (OpenGL) проекта, а также папку на жёстком диске компьютера для размещения проекта (рис. 37).

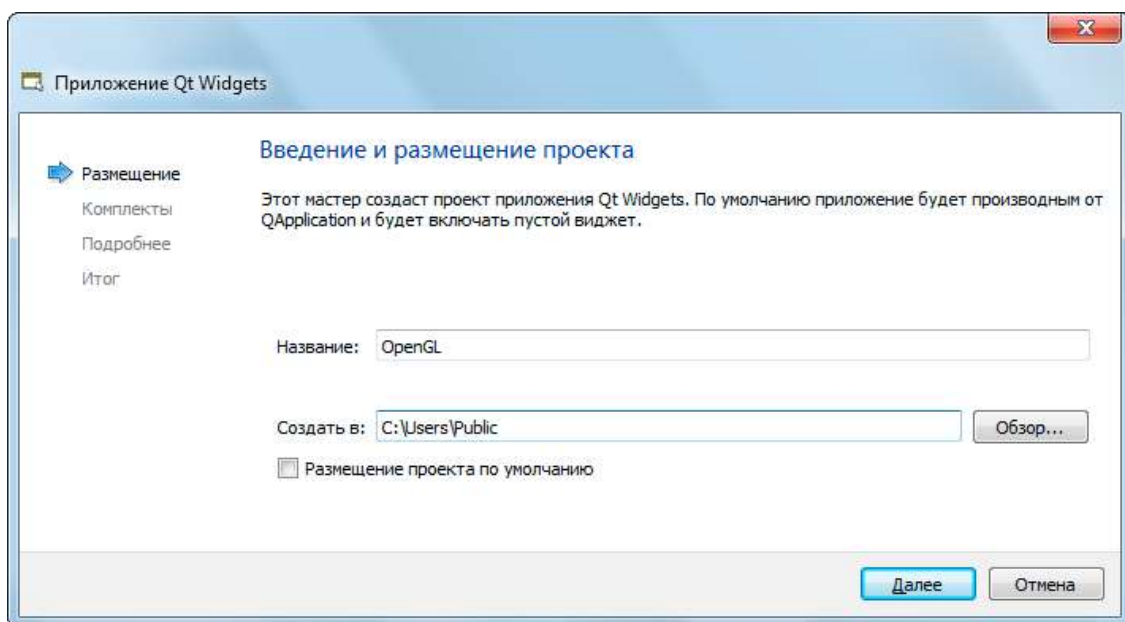


Рис. 37. Выбор имени и размещения проекта.

3. В следующем окне мастера создания проекта следует оставить параметры по умолчанию (рис. 38).

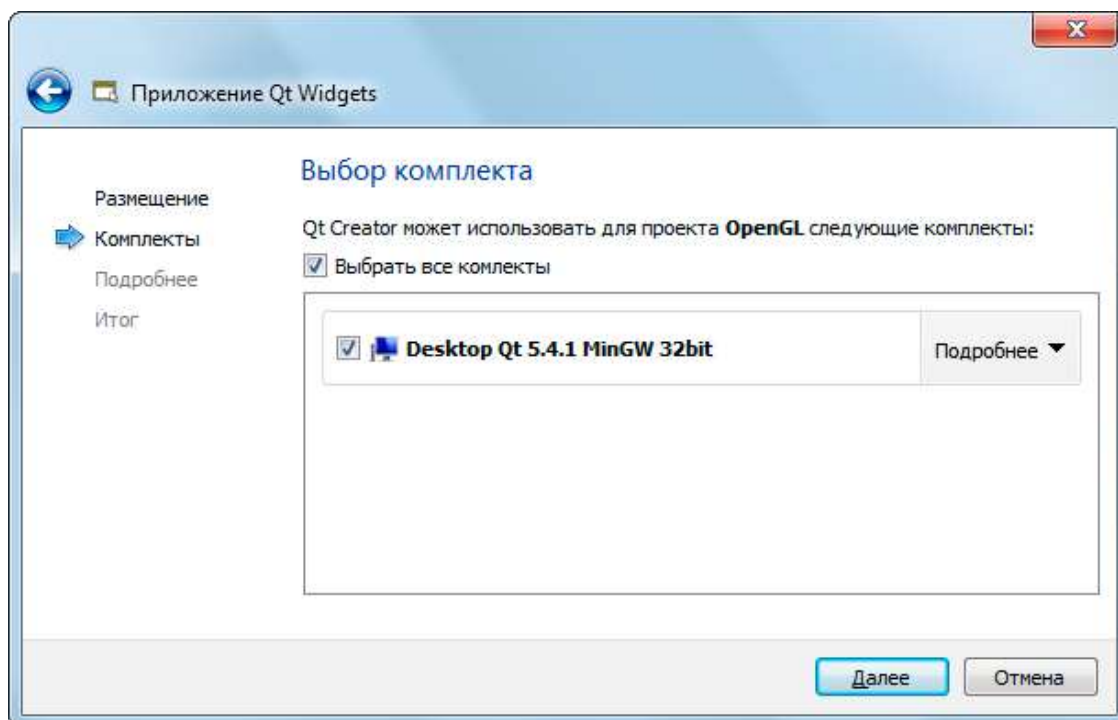


Рис. 38. Выбор комплекта сборки приложения Qt.

4. В следующем окне необходимо выбрать имя класса виджета OpenGL. Настройки, необходимые для выполнения данного урока, приведены на рис. 39.

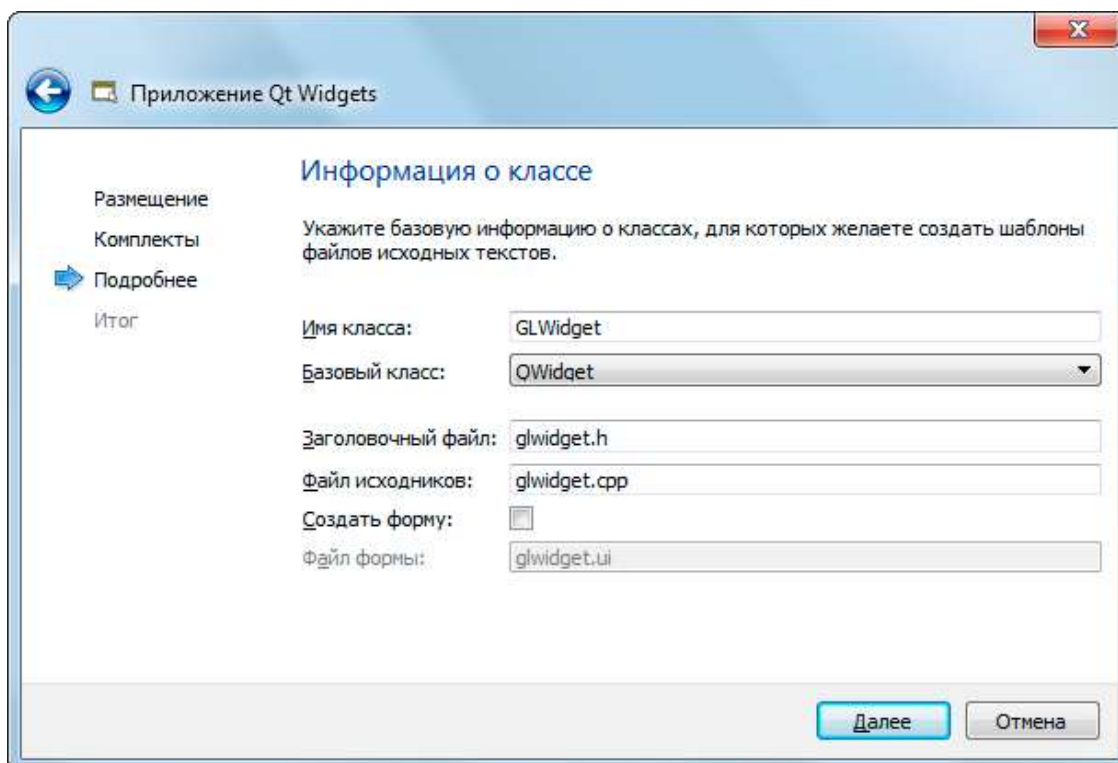


Рис. 39. Указание информации о классе виджета OpenGL.

5. В завершающем окне мастера приведён список файлов, которые будут созданы на данном этапе (рис. 40).

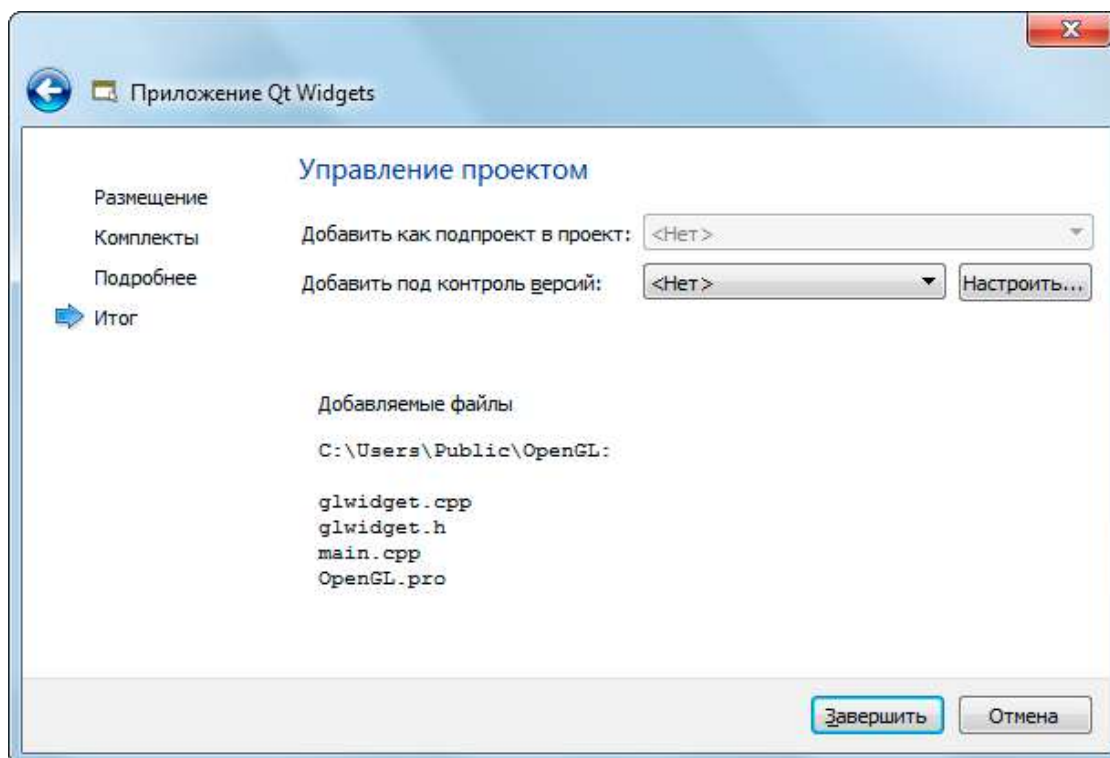


Рис. 40. Завершающий этап создания нового проекта OpenGL.

6. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода и файл проекта, который придётся изменить, чтобы задействовать OpenGL (рис. 41).

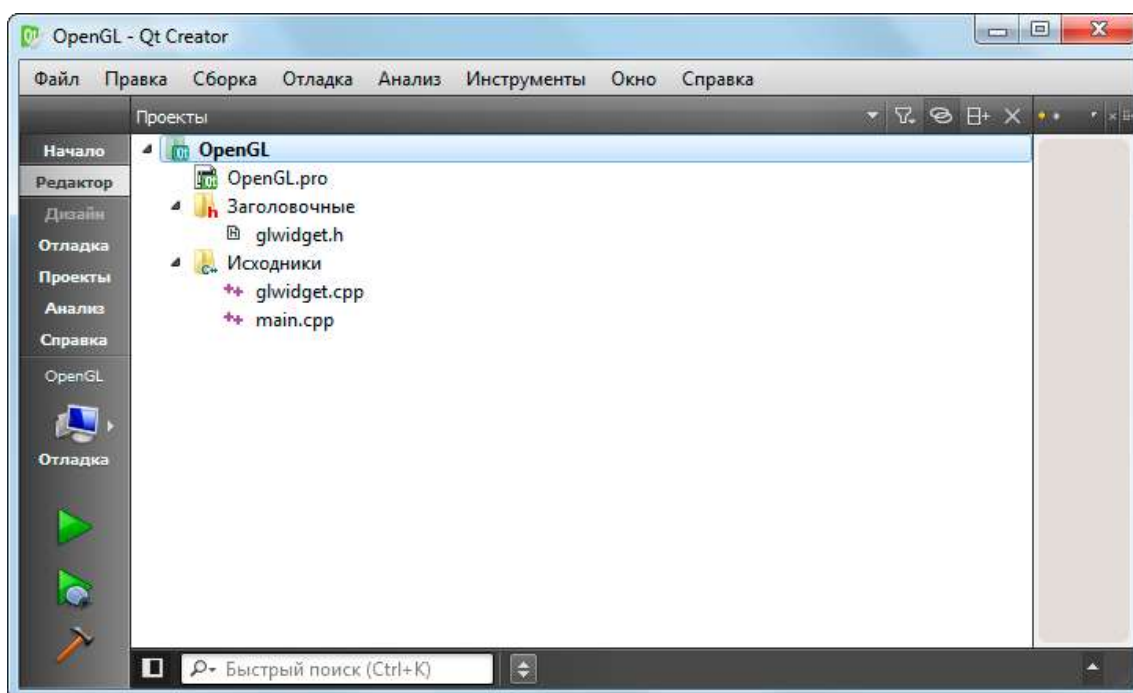


Рис. 41. Секция окна Qt Creator со структурой проекта.

7. Файл исходного кода `glwidget.cpp` содержит реализации методов класса `GLWidget` (рис. 42). Методы `initializeGL`, `resizeGL` и `paintGL` являются виртуальными методами, которые нужно определить в каждом потомке `QGLWidget`.

```
1. #include "glwidget.h"
2.
3. void GLWidget::initializeGL()
4. {
5.     glEnable(GL_DEPTH_TEST);           // Включение буфера глубины
6. }
7.
8. void GLWidget::resizeGL(int w, int h)
9. {
10.    glViewport(0, 0, w, h);             // Настройка порта просмотра
11.    glMatrixMode(GL_PROJECTION);        // Настройка проекционной матрицы
12.    glLoadIdentity();                  // Матрица по умолчанию
13.    glFrustum(-1, 1, -1, 1, 5, 30);    // Настройка плоскостей отсечения
14. }
15.
16. void GLWidget::paintGL() {
17.
18.    glClear(GL_COLOR_BUFFER_BIT);       // Очистка буфера цвета
19.    glClear(GL_DEPTH_BUFFER_BIT);      // Очистка буфера глубины
20.    glMatrixMode(GL_MODELVIEW);        // Настройка мировой матрицы
21.    glLoadIdentity();                  // Загрузка начальных значений
22.    glTranslatef(0.0f, 0.0f, -20.0f);   // Смещение куба по оси Z
23.    glRotatef(30.0, 1.0, 1.0, 0.0);    // Вращение по оси X
24.    glBegin(GL_QUADS);                 // Построение трёхмерного куба
25.    /*----- Передняя грань -----*/
26.    glColor3f(1.0, 0.0, 0.0);
27.    glVertex3f(1.0, 1.0, 1.0);
28.    glVertex3f(-1.0, 1.0, 1.0);
29.    glVertex3f(-1.0, -1.0, 1.0);
30.    glVertex3f(1.0, -1.0, 1.0);
31.    /*----- Задняя грань -----*/
32.    glColor3f(0.0, 1.0, 0.0);
33.    glVertex3f(1.0, 1.0, -1.0);
34.    glVertex3f(-1.0, 1.0, -1.0);
35.    glVertex3f(-1.0, -1.0, -1.0);
36.    glVertex3f(1.0, -1.0, -1.0);
37.    /*----- Верхняя грань -----*/
38.    glColor3f(0.0, 0.0, 1.0);
39.    glVertex3f(-1.0, 1.0, 1.0);
40.    glVertex3f(1.0, 1.0, 1.0);
41.    glVertex3f(1.0, 1.0, -1.0);
42.    glVertex3f(-1.0, 1.0, -1.0);
43.    /*----- Нижняя грань -----*/
44.    glColor3f(0.0, 1.0, 1.0);
45.    glVertex3f(1.0, -1.0, 1.0);
46.    glVertex3f(1.0, -1.0, -1.0);
47.    glVertex3f(-1.0, -1.0, -1.0);
48.    glVertex3f(-1.0, -1.0, 1.0);
49.    /*----- Правая грань -----*/
50.    glColor3f(1.0, 0.0, 1.0);
51.    glVertex3f(1.0, 1.0, 1.0);
52.    glVertex3f(1.0, -1.0, 1.0);
53.    glVertex3f(1.0, -1.0, -1.0);
54.    glVertex3f(1.0, 1.0, -1.0);
55.    /*----- Левая грань -----*/
56.    glColor3f(1.0, 1.0, 0.0);
57.    glVertex3f(-1.0, 1.0, 1.0);
58.    glVertex3f(-1.0, -1.0, 1.0);
59.    glVertex3f(-1.0, -1.0, -1.0);
60.    glVertex3f(-1.0, 1.0, -1.0);
61.    glEnd();                           // Завершение рендеринга куба
62. }
```

Рис. 42. Определение класса виджета OpenGL.

8. Для того, чтобы включить поддержку OpenGL в приложении Qt, необходимо добавить соответствующий модуль в проект. В первой строке файла проекта нужно приписать название модуля `opengl` в директиве `QT += ...`. На рис. 43 приведено содержимое файла проекта `OpenGL.pro`.

```
1. QT      += core gui opengl widgets
2. TARGET  = OpenGL
3. TEMPLATE = app
4. SOURCES += main.cpp glwidget.cpp
5. HEADERS += glwidget.h
```

Рис. 43. Содержимое файла проекта Qt.

9. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы. (рис. 44). Эта функция выводит на экран окно виджета OpenGL и устанавливает его размеры.

```
1. #include "glwidget.h"
2. #include <QApplication>
3.
4. int main(int argc, char *argv[])
5. {
6.     QApplication a(argc, argv);           // Создание объекта приложения
7.     GLWidget w;                           // Создание объекта класса GLWidget
8.     w.resize(640, 480);                   // Установка размеров виджета
9.     w.show();                             // Вывод окна на экран
10.    return a.exec();                       // Запуск приложения
11. }
```

Рис. 44. Главная функция программы.

10. На рис. 45 приведено объявление класса `GLWidget`. Как видно из исходного кода, класс виджета является наследником суперкласса `QGLWidget`, следовательно, поддерживает управление OpenGL и способен выводить трёхмерную графику.

```
1. #ifndef MYGLWIDGET_H
2. #define MYGLWIDGET_H
3.
4. #include <QGLWidget>
5.
6. class GLWidget : public QGLWidget
7. {
8.     Q_OBJECT
9.
10. protected:
11.     void initializeGL();                 // Функция инициализации OpenGL
12.     void paintGL();                     // Функция рисования кадра
13.     void resizeGL (int, int);           // Функция изменения размеров окна
14. };
15.
16. #endif
```

Рис. 45. Объявление класса виджета OpenGL.

11. На рис. 46 показан результат работы приложения. Главное окно программы залито цветом по умолчанию – чёрным, в центре находится трёхмерный куб. В более сложных случаях трёхмерные объекты моделируются в специальных редакторах, а затем загружаются в программу прямо из файла. Но, так или иначе, для рендеринга используется графический программный интерфейс OpenGL.

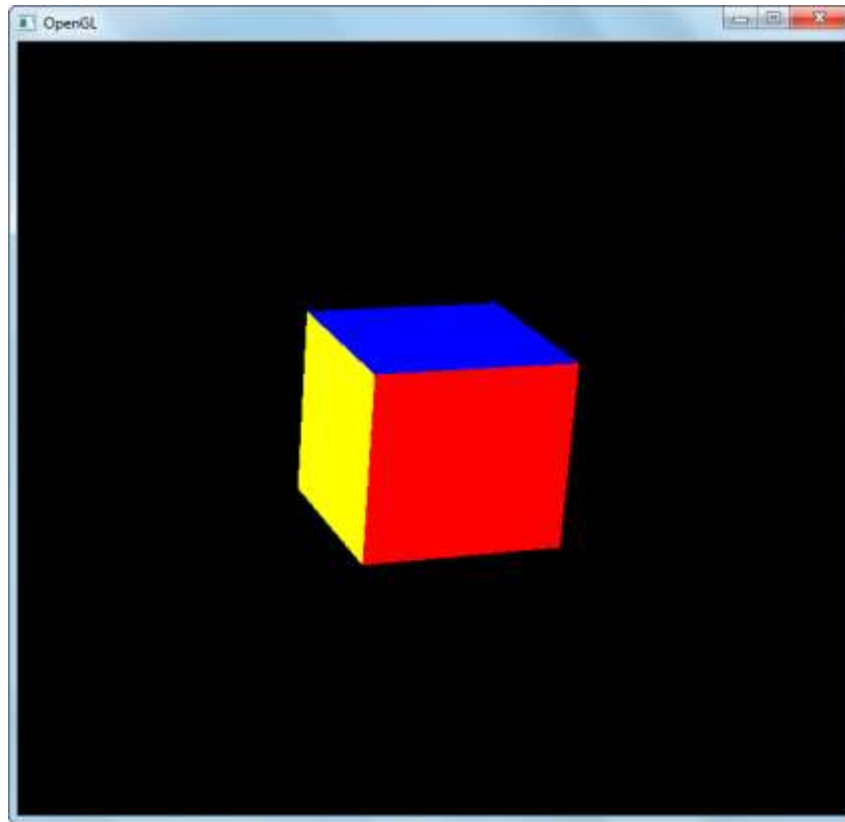


Рис. 46. Результат работы программы.

Вопросы

1. Что такое OpenGL?
2. Какими возможностями обладает OpenGL?
3. Какие классы программирования OpenGL имеются в Qt?

Урок № 5

Работа с мультимедиа.

Цель урока: Изучить основные принципы работы с мультимедиа в Qt, познакомиться с возможностями воспроизведения звука и видео, научиться создавать графические приложения для работы со звуковыми и видеоданными в среде Qt.

Теоретическая часть

Мультимедиа – это интерактивные системы, обеспечивающие одновременную работу со звуком, анимированной компьютерной графикой, видео, статическими изображениями и текстами.

Мультимедиа – сравнительно молодая отрасль новых информационных технологий. Под этим термином понимается одновременное воздействие на пользователя по нескольким информационным каналам. При этом пользователю, как правило, отводится активная роль.

Компьютерные системы мультимедиа находят широкое применение в образовании, искусстве, рекламе, науке, торговле, и других областях человеческой деятельности. Причём в каждой из этих областей применение мультимедиа открывает новые возможности, которые были недоступны при использовании старых технологий.

Главным примером мультимедийных компьютерных программ являются игры. В таких приложениях сочетаются разнообразные формы подачи информации с интерактивным управлением. Красочное оформление, стереофоническое звуковое сопровождение, движущиеся персонажи – всё это создаёт иллюзию реальности происходящих на экране событий. Кроме того, с помощью мыши, клавиатуры или джойстика играющий может использовать игровые объекты, управлять персонажами и многое другое. Именно игры являются двигателем прогресса мультимедийных технологий, поскольку в игровой индустрии постоянно испытываются новые методы воздействия информации на пользователя.

Современные компьютерные обучающие программы, как правило, создаются в технологии мультимедиа. Используя одновременно зрительный и звуковой информационные каналы ученика, такие программы помогают ему лучше понять и запомнить учебный материал. Кроме того, интерактивный режим работы позволяет ученику самому влиять на темп обучения, проверять степень усвоения материала, возвращаться к повторению непонятых фрагментов урока.

Всё большей популярностью пользуются электронные справочники, энциклопедии, художественные и музыкальные альбомы, созданные в технологии мультимедиа. Они содержат невиданные ранее объёмы информации, с цветными иллюстрациями, анимационными фильмами, видеороликами и музыкальным сопровождением. Например, мультимедийная музыкальная энциклопедия даёт возможность прослушать музыкальные произведения и одновременно увидеть выдающихся дирижёров и исполнителей.

Представление результатов компьютерного моделирования в мультимедийной форме также даёт очень сильный эффект. Например, осуществив на компьютере астрономические расчёты, получив траекторию движения небесного тела через определённое количество лет, можно воспроизвести на экране его перемещение в космосе в виде анимационного ролика со звуковыми эффектами.

Также мультимедиа активно используются в торговой рекламе и в сфере услуг. Всё чаще можно увидеть в торговых залах и витринах магазинов компьютеры, на экранах которых демонстрируется реклама продаваемых товаров. Мультимедийную рекламу также постоянно показывают по телевидению.

Разработчикам приложений на Qt как правило доступны те мультимедийные возможности, которые предоставляет платформа, на которой происходит запуск программы. Мультимедиа в среде Qt в основном представлена разнообразными средствами воспроизведения аудио и видео. Кроме того, есть ряд классов, обеспечивающих доступ к камере компьютера и другим периферийным устройствам (например, средству передачи данных Bluetooth).

Ниже перечислены некоторые примеры того, что может быть сделано с помощью модуля мультимедиа Qt:

- доступ к аудиоустройствам для ввода и вывода информации;
- проигрывание звуковых эффектов с низкой задержкой;
- проигрывание мультимедийных файлов из списков воспроизведения;
- запись и сжатие аудио;
- декодирование аудиофайлов в памяти для обработки;
- настройка и прослушивание радиостанций.

Следует заметить, что мультимедийные возможности библиотеки Qt постоянно расширяются, по мере того как появляются всё новые возможности влияния на пользователей. Например, уже сейчас с помощью Qt можно создавать современные и качественные игры.

Практическая часть

1. Создание проекта приложения с использованием мультимедийных возможностей Qt неизменно начинается с выбора типа проекта «*Приложение Qt Widgets*» в начальном окне мастера (рис. 47).

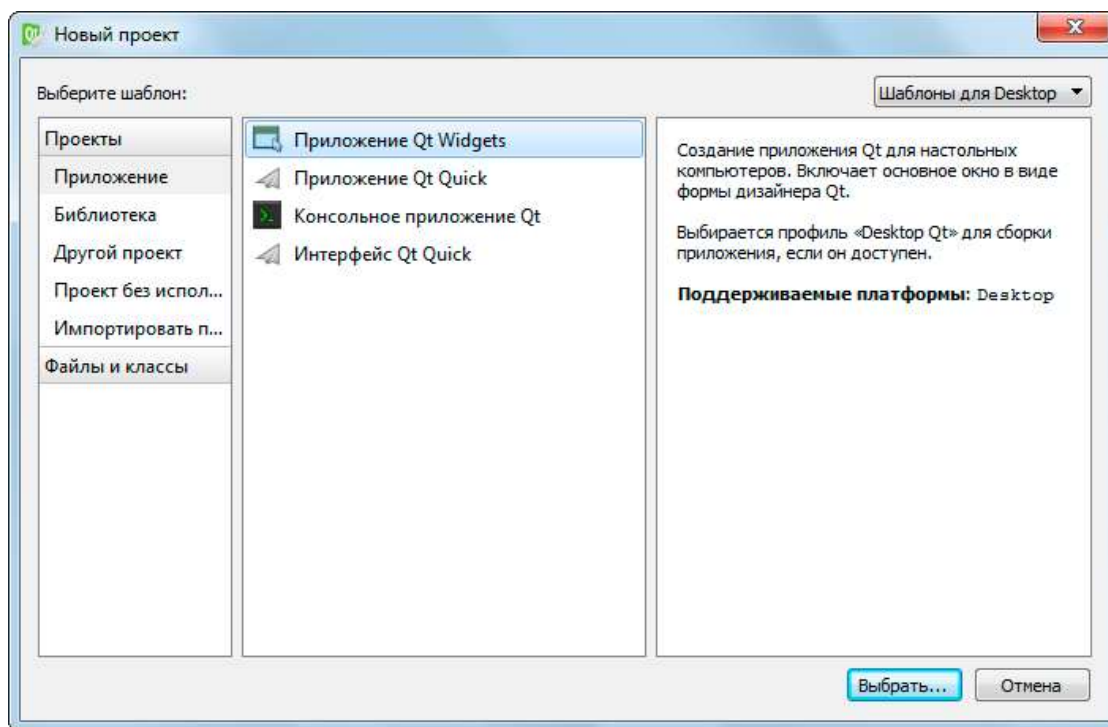


Рис. 47. Окно выбора типа проекта.

2. В следующем окне мастера создания проекта нужно указать имя проекта (в данной работе Multimedia), а также папку на жёстком диске компьютера для размещения проекта (рис. 48).

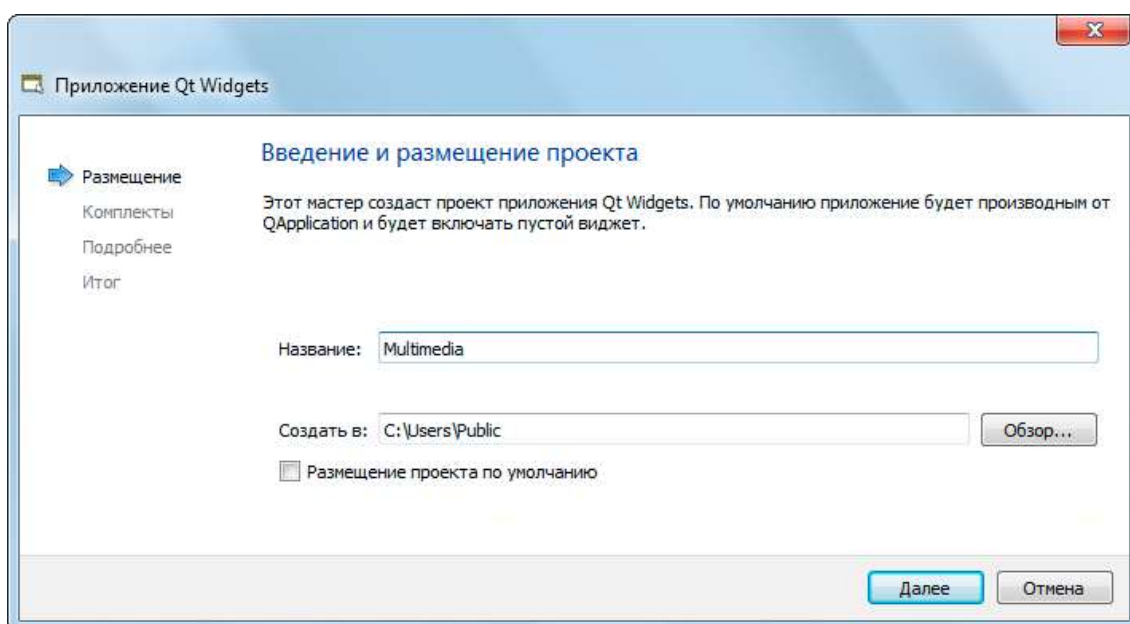


Рис. 48. Выбор имени и размещения проекта.

3. В следующем окне мастера создания проекта следует оставить параметры по умолчанию (рис. 49).

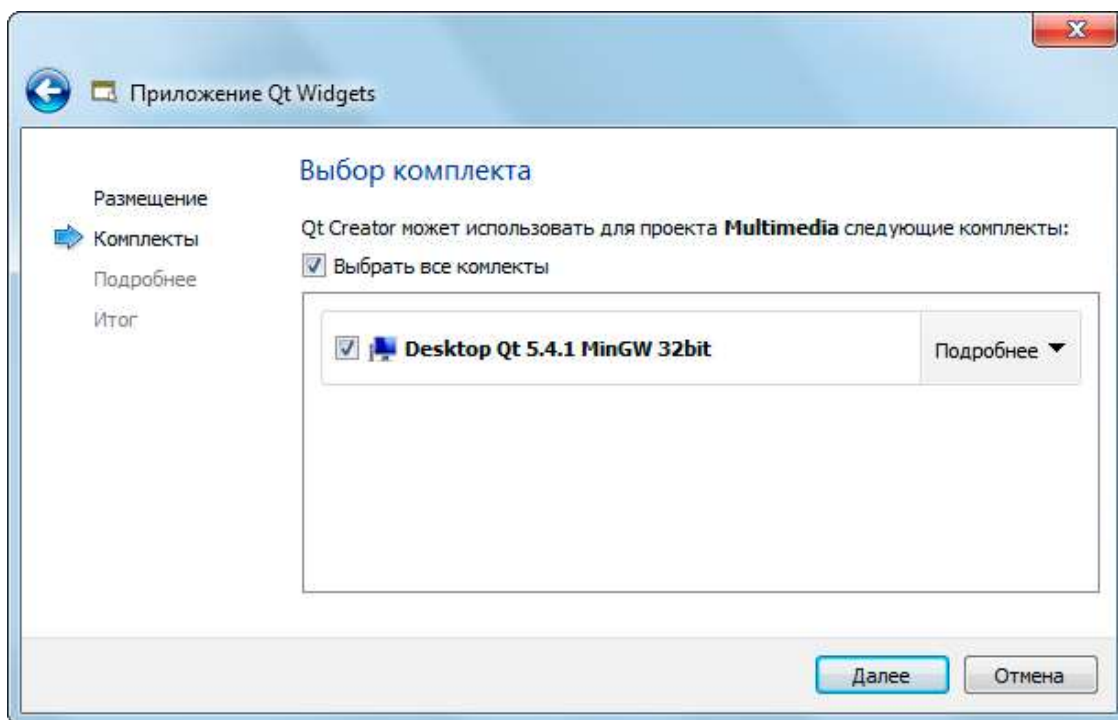


Рис. 49. Выбор комплекта сборки приложения Qt.

4. В следующем окне необходимо выбрать имя класса виджета мультимедийного плеера. Настройки, необходимые для выполнения данного урока, приведены на рис. 50.

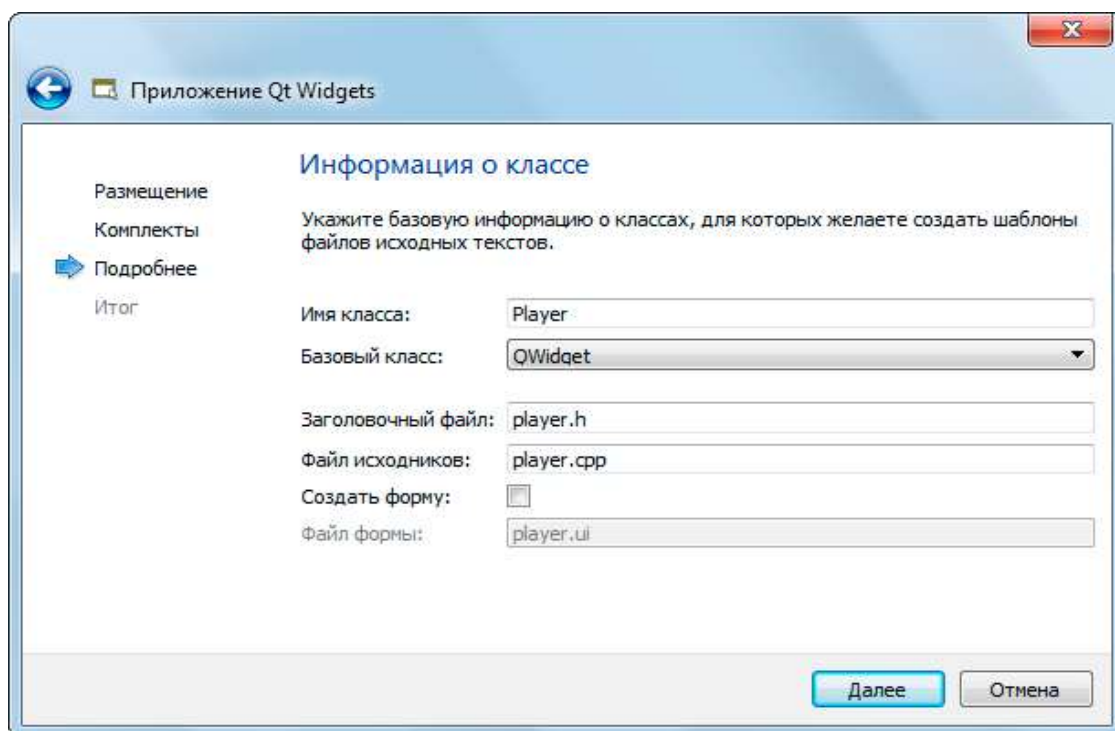


Рис. 50. Указание информации о классе виджета плеера.

5. В завершающем окне мастера приведён список файлов, которые будут созданы на данном этапе (рис. 51).

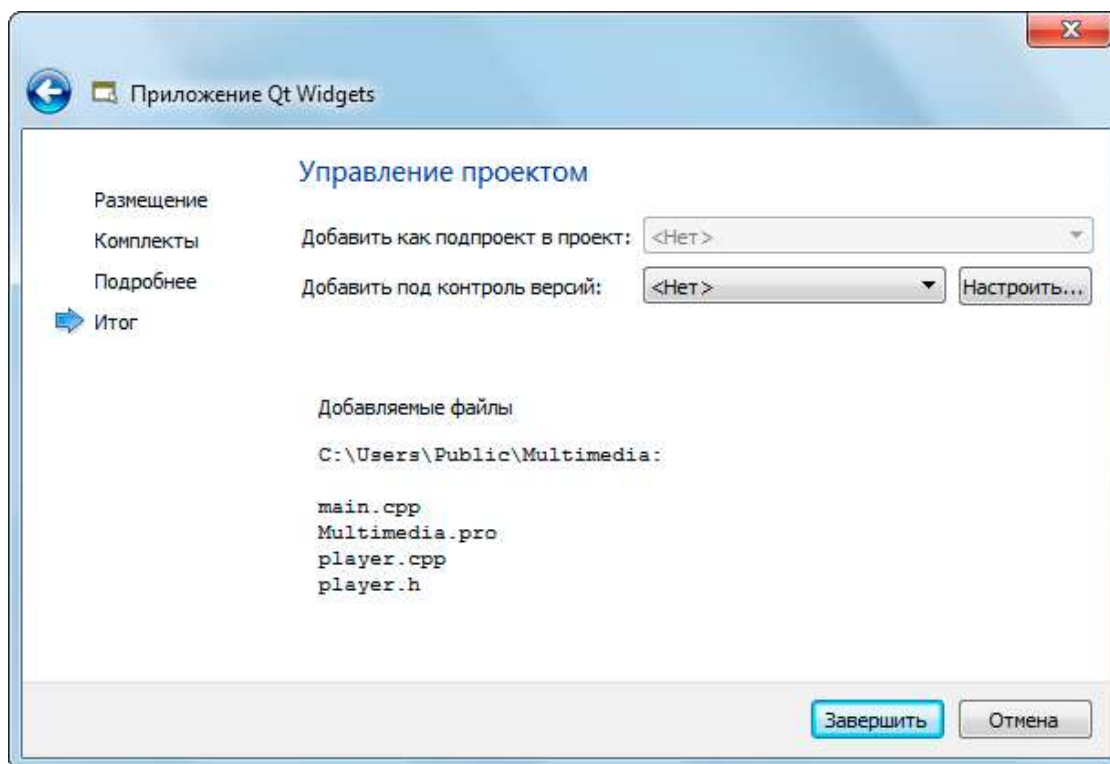


Рис. 51. Завершающий этап создания нового проекта.

6. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода и файл проекта (рис. 52).

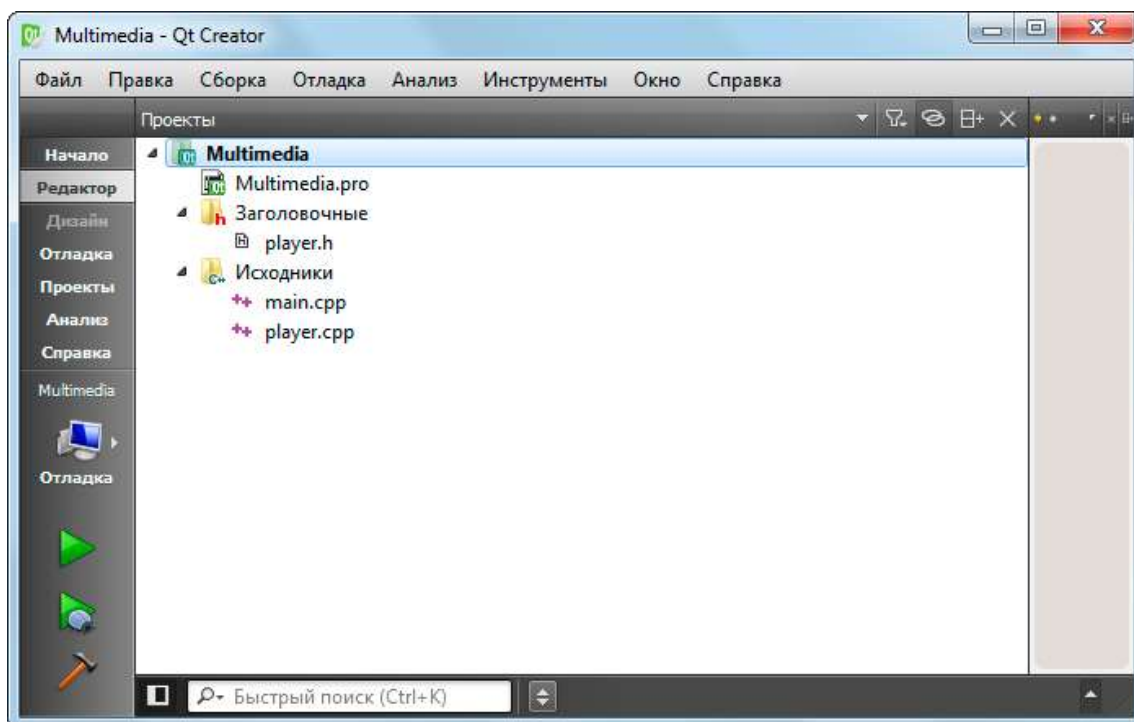


Рис. 52. Секция окна Qt Creator со структурой проекта.

7. Файл исходного кода `player.cpp` содержит реализации методов класса `Player`, который представляет в приложении мультимедийный плеер (рис. 53).

```
1.  #include "player.h"
2.
3.  Player::Player() : player(0, QMediaPlayer::VideoSurface) // Инициализация плеера
4.  {
5.      i1 = style()->standardIcon(QStyle::SP_MediaPlay);      // Значок "Старт"
6.      i2 = style()->standardIcon(QStyle::SP_MediaPause);    // Значок "Пауза"
7.      open.setText("Открыть файл");                          // Имя кнопки выбора файла
8.      play.setIcon(i1);                                       // Значок кнопки запуска
9.      slider.setOrientation(Qt::Horizontal);                 // Ориентация ползунка
10.     controls.addWidget(&open);                             // Компоновка виджетов
11.     controls.addWidget(&play);
12.     controls.addWidget(&slider);
13.     common.addWidget(&video);
14.     common.addLayout(&controls);
15.     player.setVideoOutput(&video);                          // Установка виджета видео
16.     setLayout(&common);                                       // Компоновка окна
17.     /*-----
18.         Установка связей между сигналами и слотами
19.     -----*/
20.     connect(&open, SIGNAL(released()), SLOT(OpenFile()));
21.     connect(&play, SIGNAL(released()), SLOT(Play()));
22.     connect(&slider, SIGNAL(sliderMoved(int)), SLOT(Position(int)));
23.     connect(&player, SIGNAL(stateChanged(QMediaPlayer::State)),
24.            SLOT(StateChanged(QMediaPlayer::State)));
25.     connect(&player, SIGNAL(positionChanged(qint64)),
26.            SLOT(PositionChanged(qint64)));
27.     connect(&player, SIGNAL(durationChanged(qint64)),
28.            SLOT(DurationChanged(qint64)));
29. }
30.
31. void Player::OpenFile()
32. {
33.     QString fileName = QFileDialog::getOpenFileName();      // Окно выбора файла
34.     player.setMedia(QUrl::fromLocalFile(fileName));         // Установка файла
35. }
36.
37. void Player::Play()
38. {
39.     if(player.state() == QMediaPlayer::PlayingState)      // Проверка состояния плеера
40.         player.pause();                                    // Остановка плеера
41.     else player.play();                                     // Иначе запуск
42. }
43.
44. void Player::StateChanged(QMediaPlayer::State state)
45. {
46.     QIcon ico = state ==                                  // Выбор значка
47.         QMediaPlayer::PlayingState ? i2 : i1;
48.     play.setIcon(ico);                                     // Установка значка
49. }
50.
51. void Player::PositionChanged(qint64 position)
52. {
53.     slider.setValue(position);                             // Положение ползунка
54. }
55.
56. void Player::DurationChanged(qint64 duration)
57. {
58.     slider.setRange(0, duration);                          // Диапазон ползунка
59. }
60.
61. void Player::Position(int position)
62. {
63.     player.setPosition(position);                           // Перемотка файла
64. }
```

Рис. 53. Определение класса мультимедийного плеера.

8. На рис. 54 приведено объявление класса `Player`. Данный класс использует объект `QMediaPlayer` для управления воспроизведением медиафайлов, а также специальный виджет `QVideoWidget` для вывода видео на экран. Этот класс также поддерживает выбор произвольных аудио или видеофайлов.

```
1. #ifndef PLAYER_H
2. #define PLAYER_H
3.
4. #include <QtGui>
5. #include <QtWidgets>
6. #include <QtMultimedia/QMediaPlayer>
7. #include <QtMultimediaWidgets/QVideoWidget>
8.
9. class Player : public QWidget
10. {
11.     Q_OBJECT
12.
13. public:
14.     Player();
15.
16. private slots:
17.     void OpenFile();           // Открытие медиа файла
18.     void Play();               // Запуск проигрывателя
19.     void StateChanged(QMediaPlayer::State); // Уведомление о состоянии плеера
20.     void PositionChanged(qint64); // Изменение положения ползунка
21.     void DurationChanged(qint64); // Изменение диапазона ползунка
22.     void Position(int);        // Прокрутка медиа файла
23.
24. private:
25.     QMediaPlayer player;      // Объект медиа плеера
26.     QSlider slider;           // Ползунок прокрутки
27.     QPushButton play;        // Кнопка "Старт"/"Пауза"
28.     QPushButton open;        // Кнопка выбора файла
29.     QVideoWidget video;      // Виджет вывода видео
30.     QHBoxLayout controls;    // Компоновка кнопок и ползунка
31.     QVBoxLayout common;      // Главная компоновка окна
32.     QIcon i1;                // Значок "Старт"
33.     QIcon i2;                // Значок "Пауза"
34. };
35.
36. #endif
```

Рис. 54. Объявление класса мультимедийного плеера.

9. Для того, чтобы включить поддержку мультимедиа в приложении Qt, необходимо добавить соответствующие модули в проект. В первой строке файла проекта нужно добавить название модулей `multimedia` и `multimediawidgets` в директиве `QT += ...`. На рис. 55 приведено содержимое файла проекта `Multimedia.pro`.

```
1. QT       += multimedia multimediawidgets
2. TARGET   = Multimedia
3. TEMPLATE = app
4. SOURCES  += main.cpp player.cpp
5. HEADERS  += player.h
```

Рис. 55. Содержимое файла проекта Qt.

10. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы. (рис. 56). Эта функция выводит на экран окно плеера и устанавливает его размеры.

```
1. #include "player.h"
2. #include <QApplication>
3.
4. int main(int argc, char *argv[])
5. {
6.     QApplication a(argc, argv);           // Создание объекта приложения
7.     Player w;                             // Создание объекта класса Player
8.     w.resize(640, 480);                   // Установка размеров плеера
9.     w.show();                             // Вывод окна плеера на экран
10.    return a.exec();                       // Запуск приложения
11. }
```

Рис. 56. Главная функция программы.

11. На рис. 57 показан результат работы приложения. В данном примере работы приложения открыт видеофайл формата `*.mp4`. Кнопка «Открыть файл» отображает диалоговое окно выбора файла, которое позволяет выбрать файл с любым расширением. Однако, необходимо заметить, что не все медиафайлы поддерживаются Qt. Для воспроизведения многих видеофайлов потребуется установить *кодеки*.



Рис. 57. Результат работы программы.

Вопросы

1. Что такое мультимедиа?
2. В каких областях используются мультимедиа?
3. Какие возможности мультимедиа имеются в Qt?

Урок № 6

Работа с сетью.

Цель урока: Изучить основные принципы работы с сетью в Qt, познакомиться с особенностями передачи данных по сети, научиться работать с сетевыми механизмами в среде Qt.

Теоретическая часть

Компьютерная сеть – это совокупность компьютеров, связанных каналами передачи информации. Основное назначение компьютерной вычислительной сети состоит в совместном использовании ресурсов и осуществлении быстрой связи.

Ресурсы – программы, файлы, а также принтеры и другие совместно используемые устройства. Ресурсом в компьютерной сети может быть также вычислительная мощность, которую делят между собой клиенты.

Клиент – компьютер, подключённый к вычислительной сети и использующий её ресурсы. В компьютерной сети может быть множество клиентов. Такие компьютеры, как правило, не предоставляют никаких данных для использования и работают с информацией, которую хранят серверы.

Сервер – компьютер, который предоставляет свои ресурсы клиентам сети. Различают следующие виды серверов:

- *файловый сервер* предназначен для хранения и предоставления файлов;
- *сервер баз данных* обеспечивает доступ клиентам к общим базам данных;
- *сервер приложений* служит для предоставления прикладных программ;
- *сервер печати* обеспечивает печать на общем устройстве с рабочих мест;
- *веб-сервер* обеспечивает предоставление информации через сеть Интернет;
- *почтовый сервер* обеспечивает управление электронной почтой.

Клиент-сервер – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между серверами и клиентами. Физически клиент и сервер – это программное обеспечение. Они взаимодействуют через компьютерную сеть посредством сетевых протоколов и могут находиться как на разных вычислительных машинах, так и на одной. Сервера ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или сервисных функций. Клиенты, как правило, предоставляют полученную информацию пользователю.

На рис. 58 представлена архитектура компьютерной сети клиент-сервер.

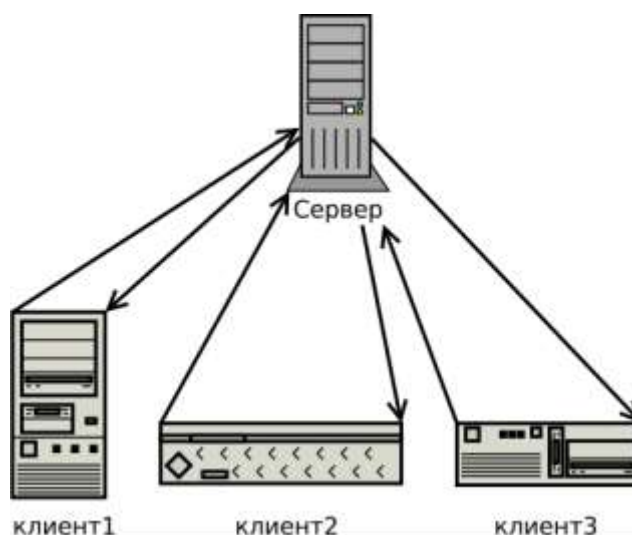


Рис. 58. Архитектура компьютерной сети клиент-сервер.

Сокет (англ. *socket* – разъём) – название программного интерфейса для пересылки данных между процессами (в частности, клиентами и серверами). Процессы могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных сетью.

Сокеты разделяют на *дейтаграммные* (datagram) и *поточные* (stream). Дейтаграммные сокеты осуществляют обмен пакетами данных, в то время как поточные сокеты устанавливают связь и используют потоковый обмен данными. Поточные сокеты могут работать в обоих направлениях, то есть клиент, записывая информацию в сокет для сервера, может получить ответ по тому же самому сокету. Поточные сокеты являются более надёжными по сравнению с дейтаграммными, так как предоставляют механизмы, предотвращающие искажение и потерю данных.

В библиотеке Qt для дейтаграммных сокетов предоставляется класс `QUdpSocket`, а для поточных – класс `QTcpSocket`. Класс `QTcpSocket` содержит набор методов для работы с *TCP* (Transmission Control Protocol) – сетевым протоколом низкого уровня, одним из основных протоколов в Интернете, и используется там, где необходима высокая надёжность передачи данных. Класс `QUdpSocket` содержит набор методов для работы с *UDP* (User Datagram Protocol). Этот сокет используется в приложениях, не требующих надёжной передачи данных между отправителем и получателем. Иногда недостатки этого протокола являются его преимуществами. Например, при передаче видеоданных лучше выбросить несколько кадров, чем отстать по времени.

Библиотека Qt предоставляет удобные средства сетевого взаимодействия на основе сигналов и слотов. На сегодняшний день сетевые механизмы Qt являются одними из самых богатых по возможностям среди аналогов.

Практическая часть

1. Создание проекта приложения с использованием средств сетевого взаимодействия Qt начинается с выбора типа проекта «*Приложение Qt Widgets*» в начальном окне мастера (рис. 59).

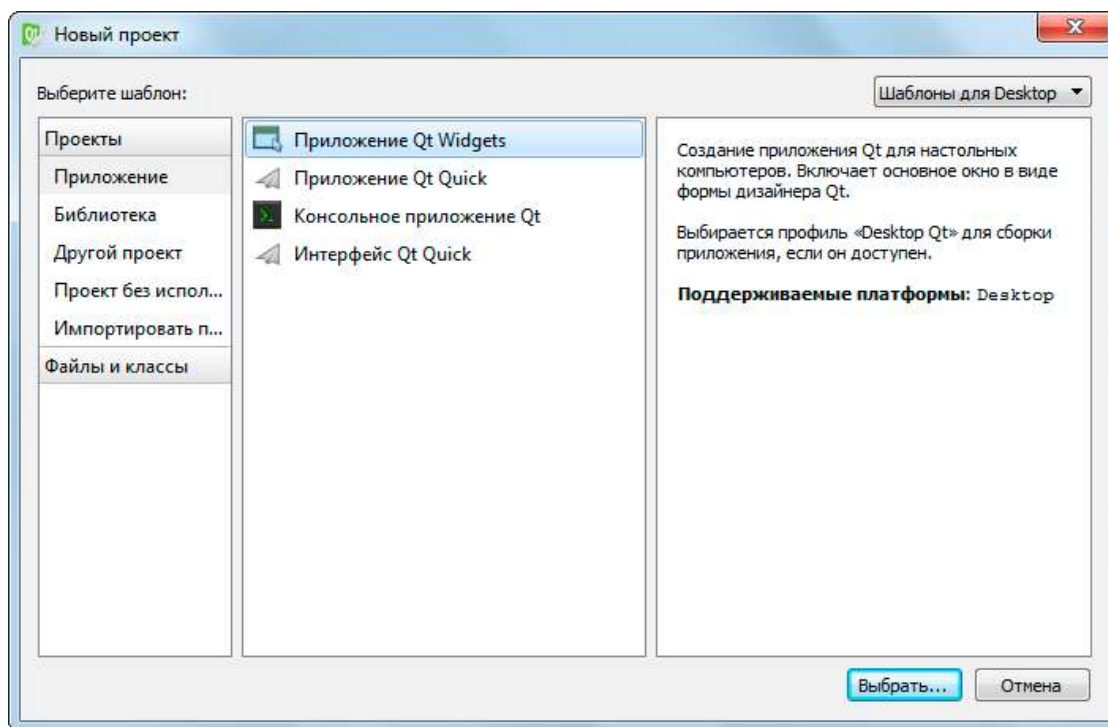


Рис. 59. Окно выбора типа проекта.

2. В следующем окне мастера нужно указать имя проекта (Network), а также папку на жёстком диске компьютера для размещения проекта (рис. 60).

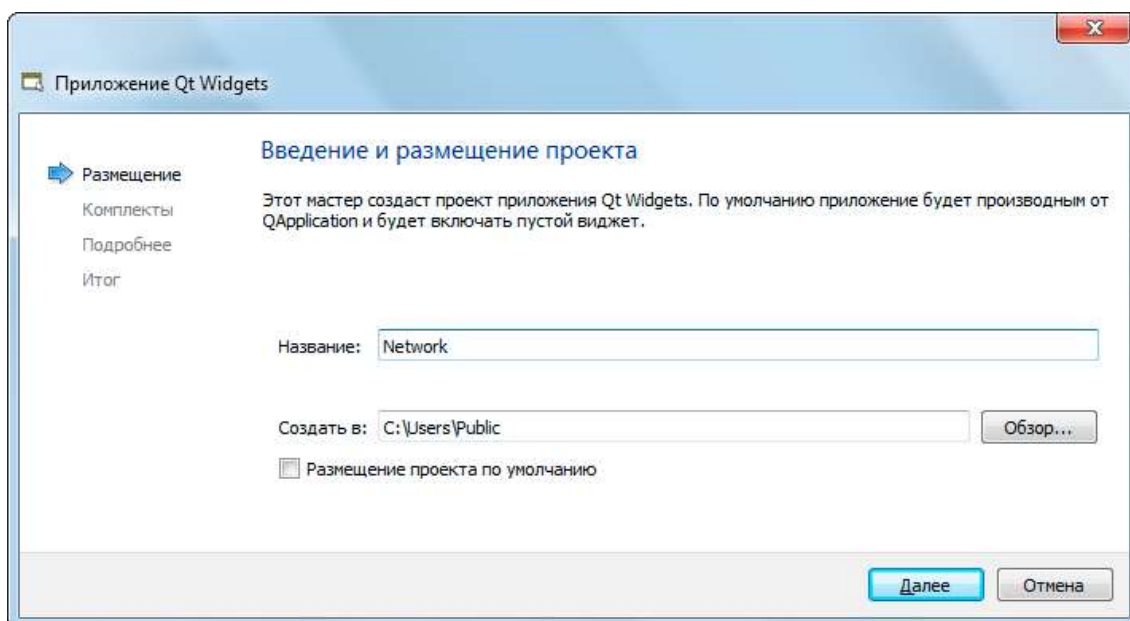


Рис. 60. Выбор имени и размещения проекта.

3. В следующем окне мастера создания проекта следует оставить параметры по умолчанию (рис. 61).

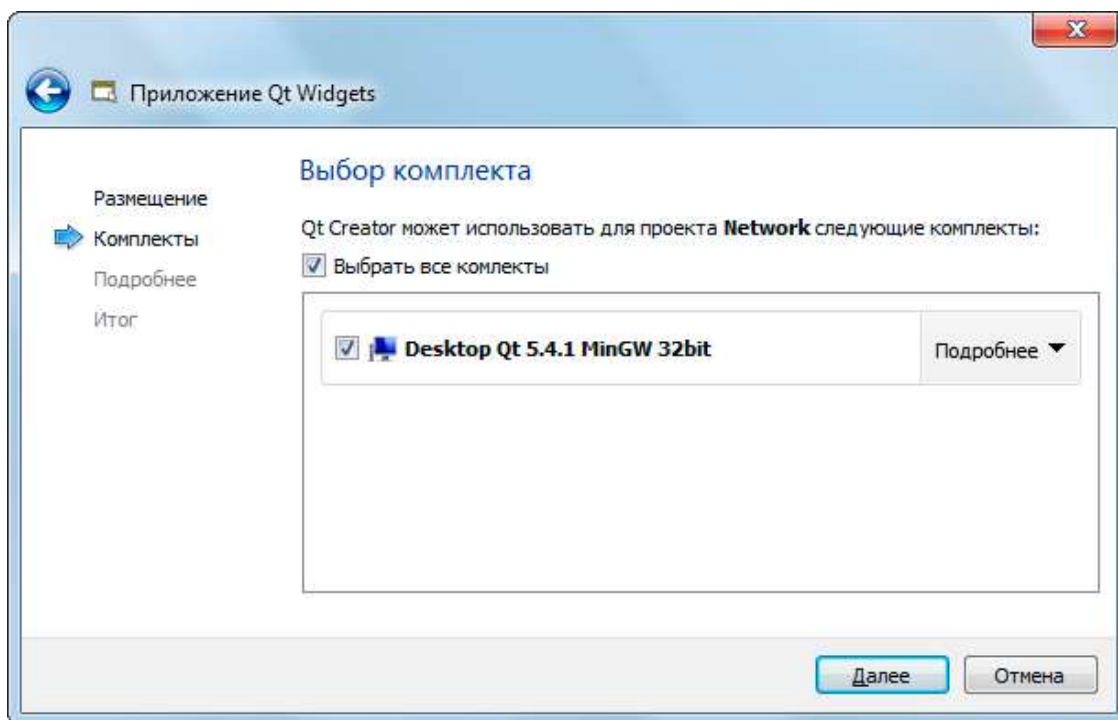


Рис. 61. Выбор комплекта сборки приложения Qt.

4. В следующем окне необходимо выбрать имя класса главного окна программы. Настройки, необходимые для выполнения данного урока, приведены на рис. 62.

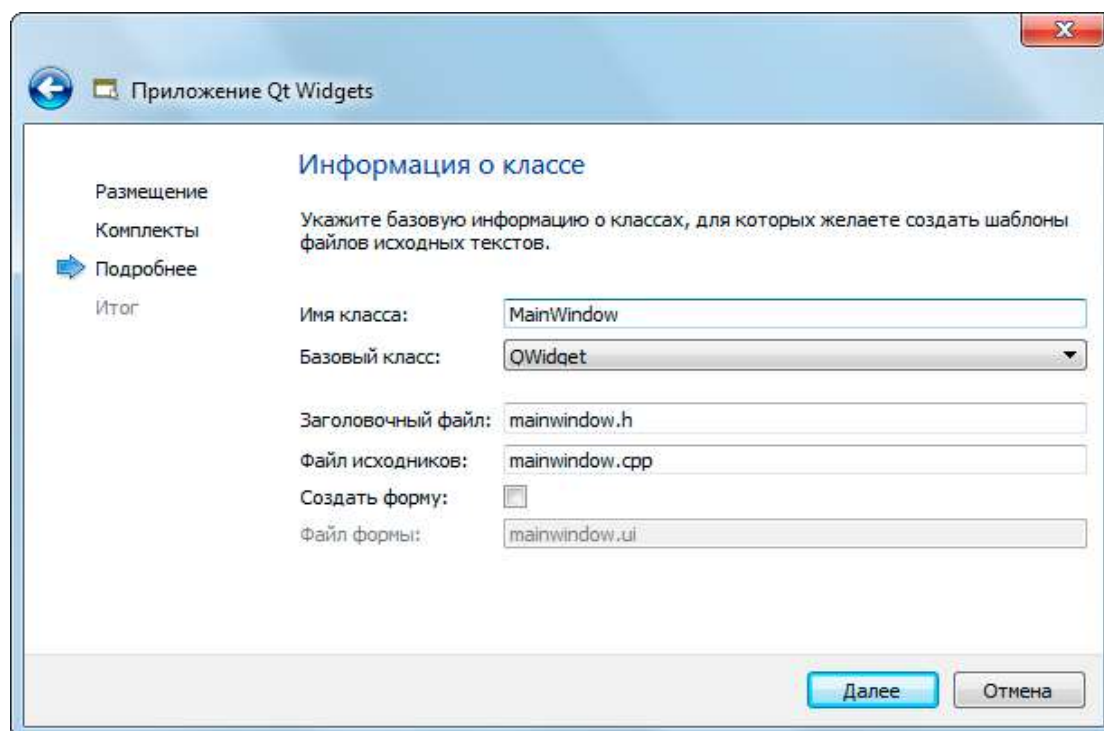


Рис. 62. Указание информации о классе окна приложения.

5. В завершающем окне мастера приведён список файлов, которые будут созданы на данном этапе (рис. 63).

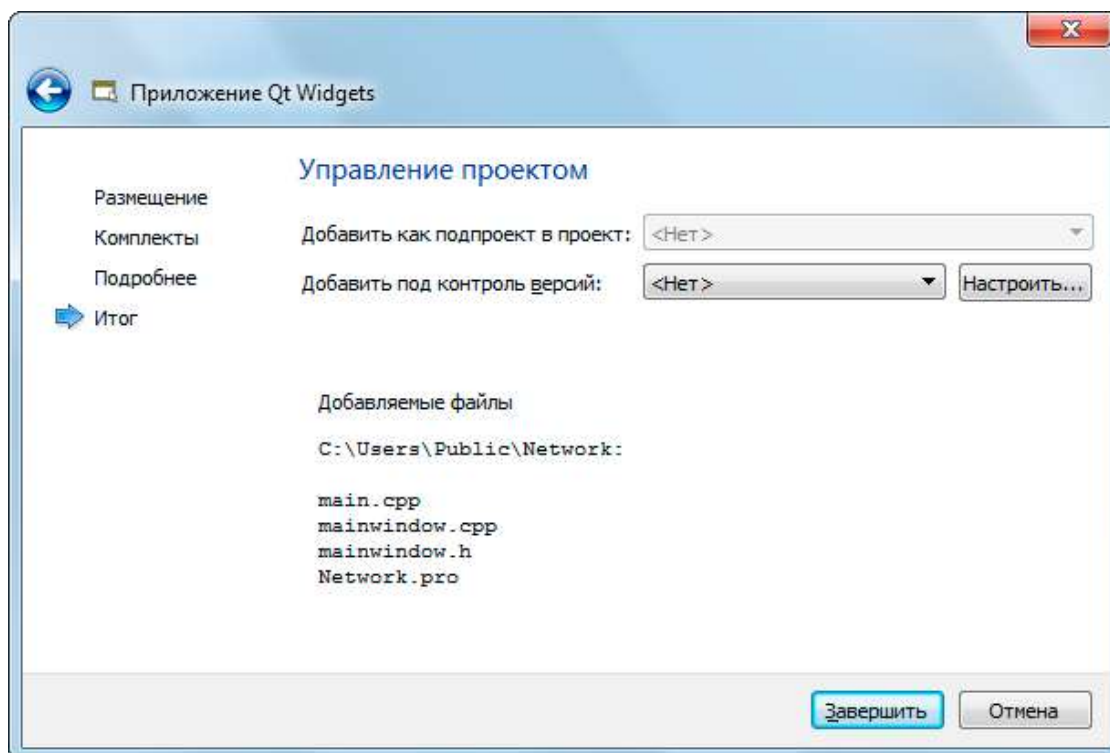


Рис. 63. Завершающий этап создания нового проекта.

6. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода и файл проекта (рис. 64).

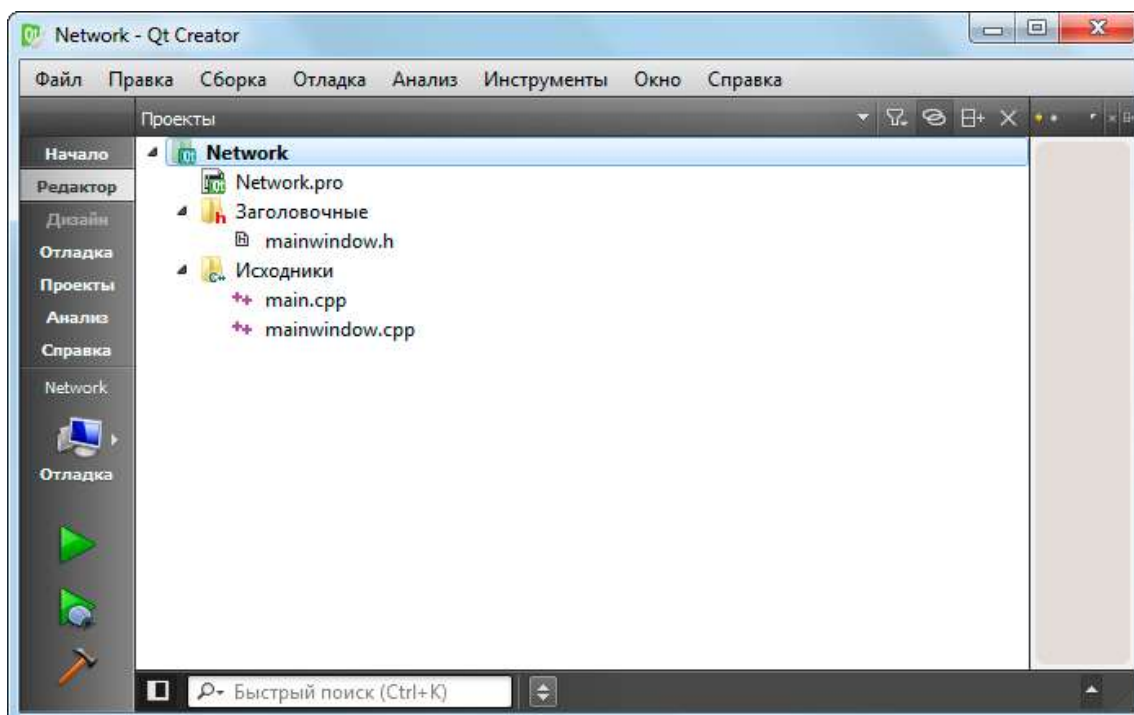


Рис. 64. Секция окна Qt Creator со структурой проекта.

7. Чтобы включить поддержку сетевых механизмов, в проект необходимо добавить модуль `network`. На рис. 65 приведено содержимое файла проекта `Network.pro`.

```
1. QT      += core gui widgets network
2. TARGET  = Network
3. TEMPLATE = app
4. SOURCES += main.cpp mainwindow.cpp
5. HEADERS += mainwindow.h
```

Рис. 65. Содержимое файла проекта Qt.

8. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы. (рис. 66). Эта функция выводит на экран главное окно программы.

```
1. #include "mainwindow.h"
2. #include <QApplication>
3.
4. int main(int argc, char *argv[])
5. {
6.     QApplication a(argc, argv);           // Создание объекта приложения
7.     MainWindow w;                         // Создание окна приложения
8.     w.show();                             // Вывод окна программы на экран
9.     return a.exec();                     // Запуск приложения
10. }
```

Рис. 66. Главная функция программы.

9. На рис. 67 приведено объявление класса `MainWindow`. Данный класс представляет главное окно программы обмена сообщениями между клиентом и сервером.

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3.
4. #include <QtWidgets>
5. #include <QtNetwork/QTcpServer>
6. #include <QtNetwork/QTcpSocket>
7.
8. class MainWindow : public QWidget
9. {
10.     Q_OBJECT
11. public:
12.     MainWindow();
13.
14. private slots:
15.     void Connection();           // Обработка подключения к серверу
16.     void SendToServer();         // Отправка сообщения серверу
17.     void SendToClient();         // Ответ сервера клиенту
18.     void ClientRead();           // Чтение сообщения от сервера
19.
20. private:
21.     QVBoxLayout layout;          // Компоновщик окна приложения
22.     QListWidget list;            // Виджет для вывода сообщений
23.     QPushButton button;         // Кнопка отправки сообщений
24.     QTcpServer server;          // Объект сервера
25.     QTcpSocket client;          // Объект клиента (сокета)
26. };
27.
28. #endif
```

Рис. 67. Объявление класса окна сетевого обмена сообщениями.

10. Файл исходного кода `mainwindow.cpp` содержит реализации методов класса `MainWindow`, который представляет окно обмена сообщениями между клиентом и сервером (рис. 68). В данной программе в конструкторе класса объект сервера настраивается для прослушивания входящих сообщений по любому возможному адресу сети. Связь с сервером возможна по конкретному порту. Затем клиент с помощью сокета TCP соединяется с сервером, используя указанный порт и IP-адрес локального компьютера. При нажатии на кнопку «Новое сообщение» на форме происходит отправка сообщения серверу. Сервер получает объект сокета, через который было отправлено сообщение, выводит на экран содержимое, и, используя тот же сокет, отправляет ответ клиенту. Клиент, со своей стороны, принимает сообщение сервера и также отображает его на экране.

```

1.  #include "mainwindow.h"
2.
3.  MainWindow::MainWindow() : QWidget()
4.  {
5.      quint16 port = 12002;                // Номер порта подключения
6.      button.setText("Новое сообщение");    // Обозначение кнопки
7.      server.listen(QHostAddress::Any, port); // Запуск сервера
8.      client.connectToHost(QHostAddress::LocalHost, port); // Подключение клиента
9.      layout.addWidget(&button);           // Добавление кнопки
10.     layout.addWidget(&list);              // Добавление списка
11.     setLayout(&layout);                  // Установка компоновщика
12.     resize(640, 480);                   // Установка размера окна
13.     /*-----
14.         Установка связей между сигналами и слотами
15.     -----*/
16.     connect(&button, SIGNAL(released()), SLOT(SendToServer()));
17.     connect(&server, SIGNAL(newConnection()), SLOT(Connection()));
18.     connect(&client, SIGNAL(readyRead()), SLOT(ClientRead()));
19. }
20.
21. void MainWindow::Connection()
22. {
23.     QTcpSocket* s = server.nextPendingConnection(); // Получение объекта клиента
24.     connect(s, SIGNAL(readyRead()), SLOT(SendToClient())); // Ответ сервера на сообщение
25. }
26.
27. void MainWindow::SendToServer()
28. {
29.     client.write(QString("Привет, сервер!").toUtf8()); // Отправка сообщения серверу
30. }
31.
32. void MainWindow::SendToClient()
33. {
34.     QTcpSocket* s = (QTcpSocket*)sender();          // Получение объекта клиента
35.     QString t = QTime::currentTime().toString();   // Получение текущего времени
36.     QString m = s->readAll();                        // Чтение сообщения клиента
37.     list.addItem(t + " Клиент: " + m);              // Отображение сообщения
38.     s->write(QString("Привет, клиент!").toUtf8());  // Ответное сообщение
39. }
40.
41. void MainWindow::ClientRead()
42. {
43.     QString t = QTime::currentTime().toString();   // Получение текущего времени
44.     list.addItem(t + " Сервер: " + client.readAll()); // Чтение сообщения сервера
45. }

```

Рис. 68. Определение класса главного окна приложения.

11. На рис. 69 показан результат работы приложения. В главном окне программы содержится виджет списка для вывода сообщений клиента и сервера. Каждое сообщение сопровождается временем его отправления для удобства отслеживания коммуникаций клиента и сервера. Данное приложение может быть легко расширено для поддержки обращений нескольких клиентов к одному серверу.

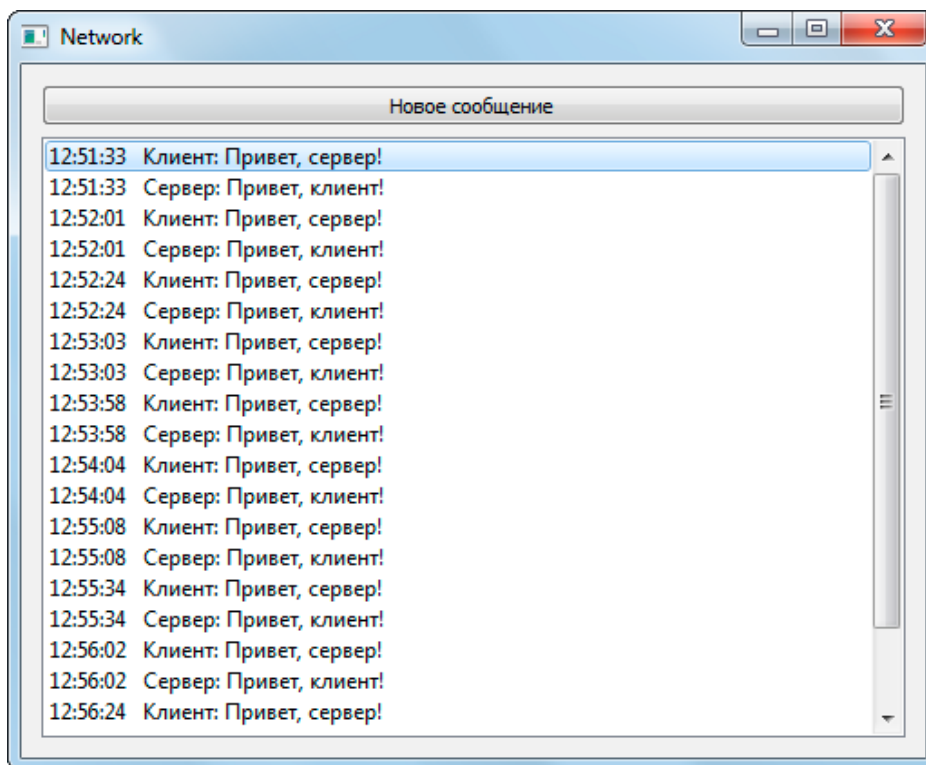


Рис. 69. Результат работы программы.

Вопросы

1. Что такое компьютерная сеть?
2. Для чего предназначены сетевые сокеты?
3. Какие сетевые механизмы имеются в Qt для реализации модели «клиент-сервер»?

Урок № 7

Работа с потоками.

Цель урока: Изучить основные принципы работы с потоками в Qt, познакомиться с особенностями многопоточной обработки данных, научиться создавать многопоточные программы в Qt.

Теоретическая часть

Многопоточность – свойство операционной системы или приложения, состоящее в том, что процесс, запущенный в системе, может состоять из нескольких потоков, выполняющихся одновременно. При выполнении некоторых задач такое разделение позволяет достичь более эффективного использования ресурсов вычислительной машины.

Многопоточность является многозадачностью на уровне одного процесса. Код многопоточной программы по-прежнему выполняется в рамках адресного пространства процесса. Различные потоки разделяют между собой общие ресурсы, например, открытые файлы. Главное преимущество многопоточности состоит в том, что потоки не обязаны быть синхронизированы во времени.

К достоинствам многопоточности в программировании можно отнести следующее:

- упрощение программы за счёт использования общего адресного пространства;
- меньшие относительно процесса временные затраты на создание потока;
- повышение производительности приложения за счёт распараллеливания процессорных вычислений.

Многопоточность является перспективным направлением в разработке программного обеспечения. В то время как постоянное увеличение частоты процессора не представляется возможным, принцип многопоточности предполагает увеличение количества вычислительных ядер с одинаковыми частотами. Такой подход к увеличению эффективности программ также подталкивает программистов к разработке более эффективных алгоритмов, которые поддерживают независимую обработку данных.

Очень часто многопоточность используется в программах не для повышения производительности, а для асинхронной обработки каких-либо задач. В таком случае количество потоков обычно невелико, и в глубоком анализе производительности приложение не нуждается.

В многопоточной среде часто возникают проблемы, связанные с использованием параллельно исполняемыми потоками одних и тех же данных или устройств. Для решения подобных проблем используются такие методы синхронизации потоков, как *мьютексы, семафоры и события*.

Ниже даны описания вышеперечисленных механизмов синхронизации:

- **Мьютекс** – это объект синхронизации, который устанавливается в особое сигнальное состояние, когда не занят ни одним потоком. Только один поток владеет этим объектом в любой момент времени. После всех необходимых действий мьютекс освобождается, предоставляя другим потокам доступ к общему ресурсу. Мьютекс поддерживает захват второй раз тем же потоком, увеличивая счётчик, не блокируя поток, и требуя потом многократного освобождения.
- **Семафоры** – это доступные ресурсы, которые могут быть приобретены несколькими потоками в одно и то же время, пока пул ресурсов не опустеет. В этом случае остальные потоки должны ждать, пока требуемое количество ресурсов не будет снова доступно.
- **Событие** – это объект, хранящий в себе логическое значение «просигнализован или нет», над которым определены операции «просигнализовать», «сбросить в непросигнализованное состояние» и «ожидать». Ожидание на просигнализованном событии есть отсутствие операции с немедленным продолжением исполнения потока. Ожидание на непросигнализованном событии приводит в остановке исполнения потока до тех пор, пока другой поток не просигнализует событие.

В библиотеке Qt существует основополагающий класс `QThread`, который представляет собой низкоуровневое средство запуска нового потока. Чтобы запустить длительную операцию на выполнение, необходимо создать подкласс `QThread`, а затем переопределить метод `QThread::run`. Внутри этого метода следует поместить код выполняемой задачи. В процессе работы поток может использовать механизм сигналов и слотов для того, чтобы оповещать о прогрессе выполнения или завершении задачи. Использование механизма сигналов и слотов весьма эффективно для реализации такого примитива синхронизации, как событие. За кулисами потоки используют очередь событий, эффективная реализация которой присутствует практически во всех операционных системах.

Библиотека Qt предоставляет удобные инструменты создания многопоточных программ. В настоящее время в Qt постоянно появляются новые механизмы распараллеливания приложений, что выгодно выделяет библиотеку среди аналогов.

Практическая часть

1. Создание проекта приложения с использованием классов запуска дополнительных потоков начинается с выбора типа проекта «*Приложение Qt Widgets*» в начальном окне мастера (рис. 70).

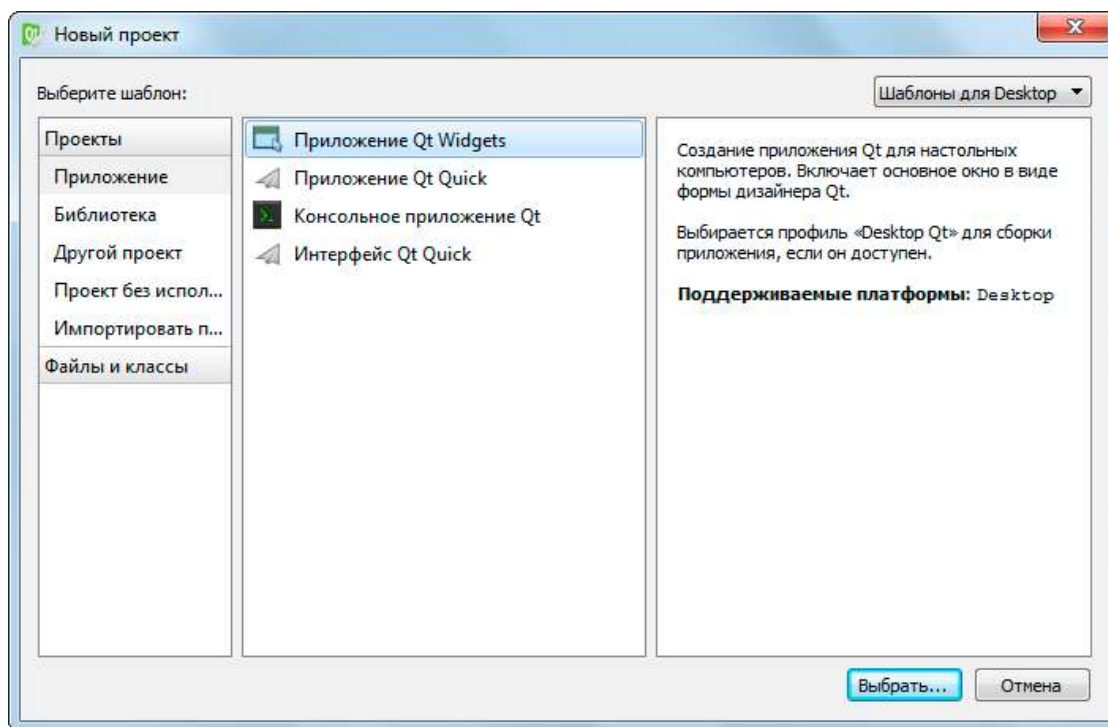


Рис. 70. Окно выбора типа проекта.

2. В следующем окне мастера нужно указать имя проекта (Threads), а также папку на жёстком диске компьютера для размещения проекта (рис. 71).

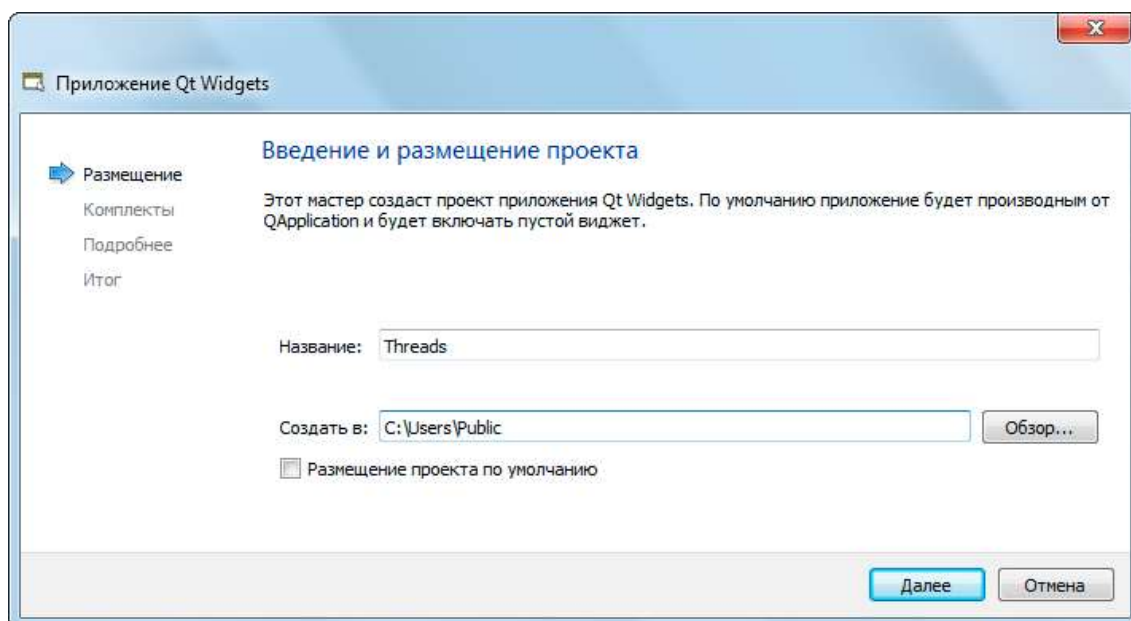


Рис. 71. Выбор имени и размещения проекта.

3. В следующем окне мастера создания проекта следует оставить параметры по умолчанию (рис. 72).

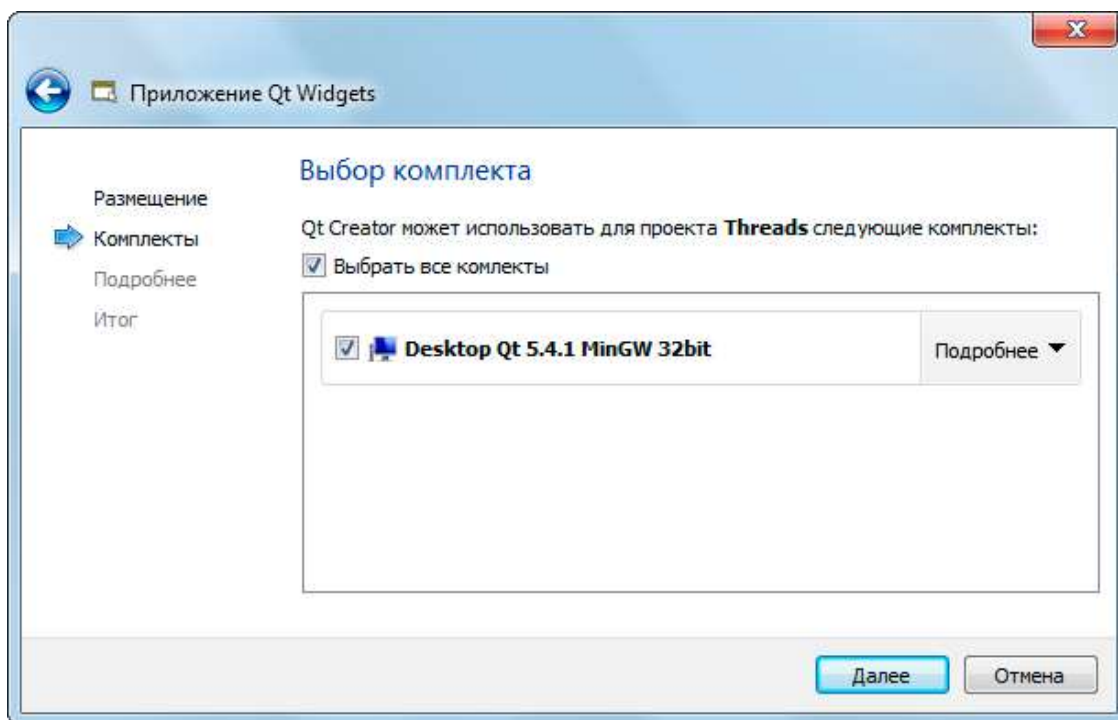


Рис. 72. Выбор комплекта сборки приложения Qt.

4. В следующем окне необходимо выбрать имя класса главного окна программы. Настройки, необходимые для выполнения данного урока, приведены на рис. 73.

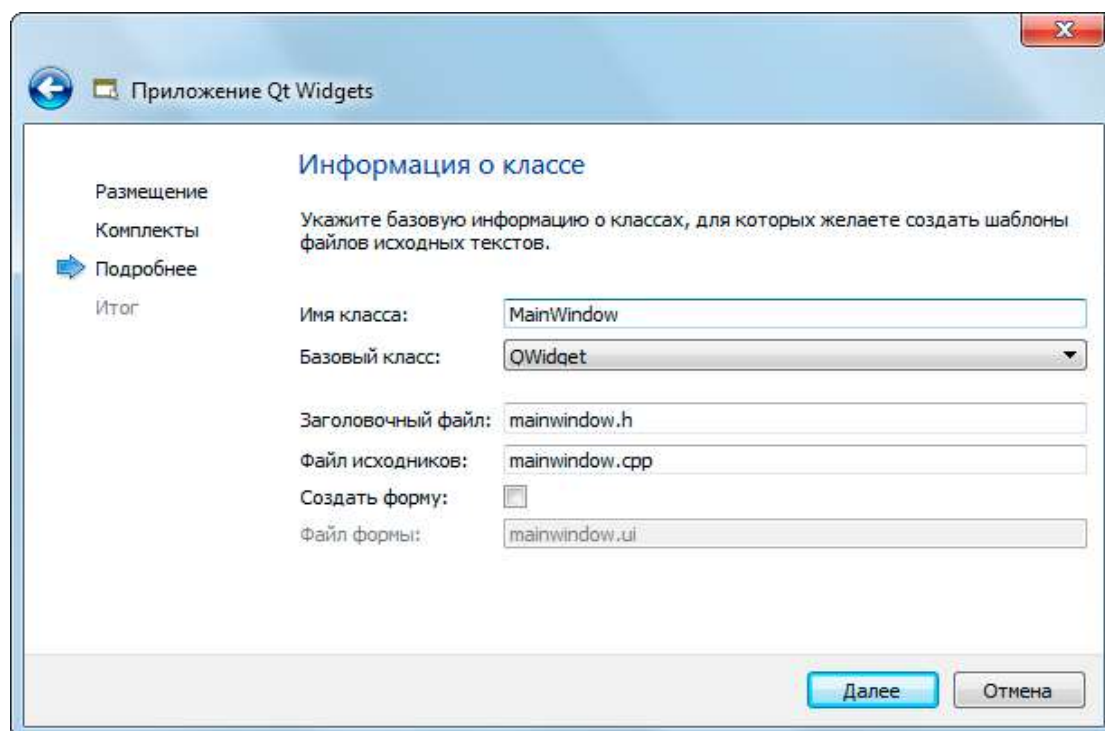


Рис. 73. Указание информации о классе формы приложения.

5. В завершающем окне мастера приведён список файлов, которые будут созданы на данном этапе (рис. 74).

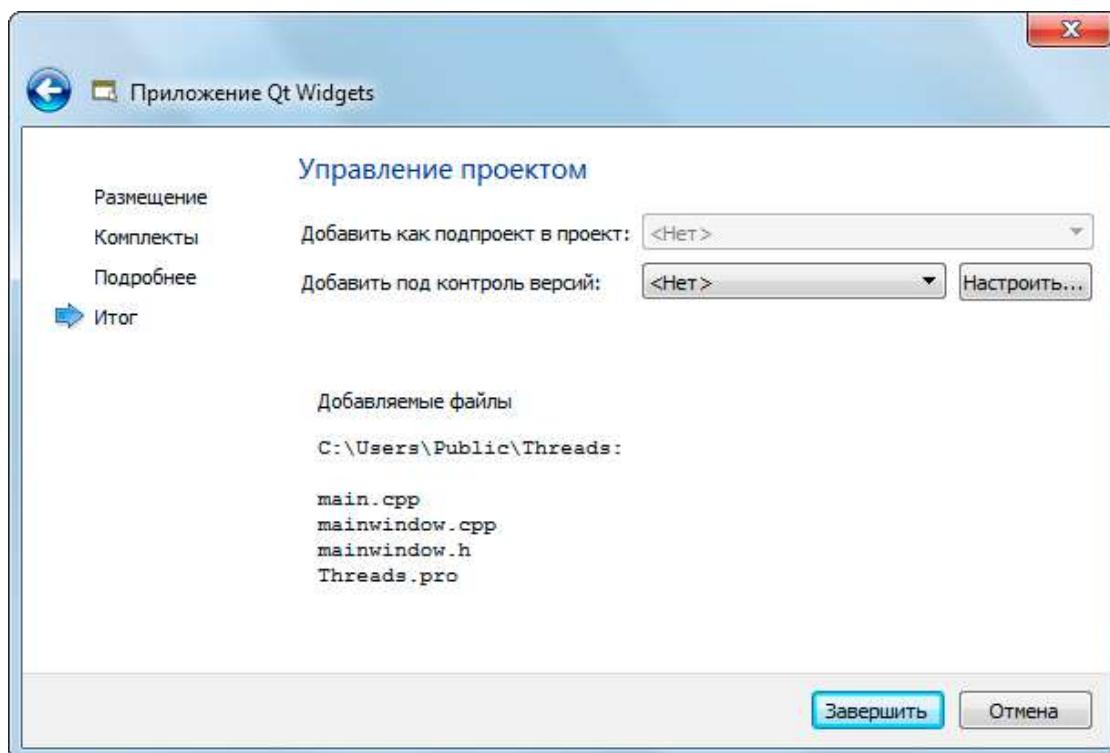


Рис. 74. Завершающий этап создания нового проекта.

6. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода и файл проекта (рис. 75).

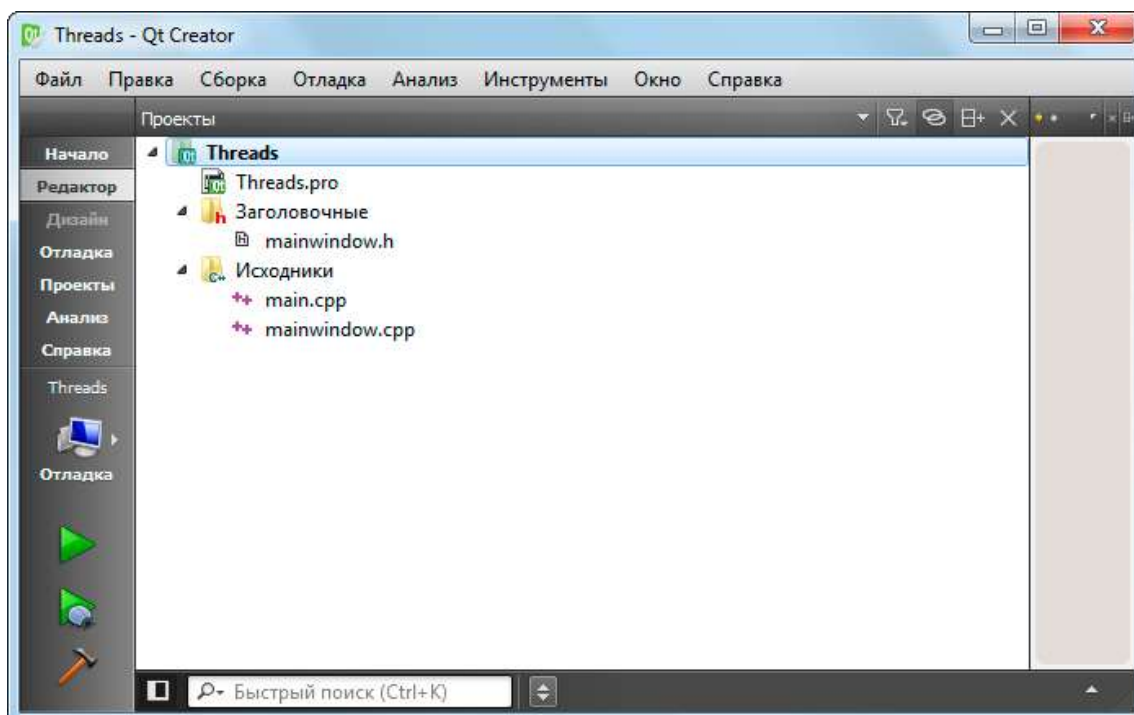


Рис. 75. Секция окна Qt Creator со структурой проекта.

7. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы. (рис. 76). Эта функция выводит на экран главное окно программы.

```
1.  #include "mainwindow.h"
2.
3.  int main(int argc, char *argv[])
4.  {
5.      QApplication a(argc, argv);          // Создание объекта приложения
6.      MainWindow w;                        // Создание окна приложения
7.      w.show();                           // Вывод окна программы на экран
8.      return a.exec();                    // Запуск приложения
9.  }
```

Рис. 76. Главная функция программы.

8. В файле `mainwindow.h` содержатся объявления сразу двух классов – класс рабочего потока и класс формы (рис. 77). Согласно синтаксису языка C++, объявление класса должно предшествовать его использованию. Таким образом, класс `Thread` предшествует классу `MainWindow`, одним из полей которого он является.

```
1.  #ifndef MAINWINDOW_H
2.  #define MAINWINDOW_H
3.
4.  #include <QGridLayout>
5.  #include <QApplication>
6.  #include <QLCDNumber>
7.  #include <QWidget>
8.  #include <QThread>
9.  #include <QTimer>
10.
11. class Thread : public QThread
12. {
13.     Q_OBJECT
14. signals:
15.     void Finished();                // Сигнал завершения приложения
16.     void CurrentValue(int);         // Сигнал таймера
17.
18. private slots:
19.     void run();                     // Функция задачи потока
20.     void NextValue();               // Функция изменения счётчика
21.
22. private:
23.     int value;                      // Переменная счётчика
24. };
25.
26. class MainWindow : public QWidget
27. {
28.     Q_OBJECT
29. public:
30.     MainWindow();                  // Конструктор формы программы
31.
32. private:
33.     QLCDNumber lcd;                // Виджет счётчика в стиле LCD
34.     QGridLayout layout;            // Сеточный компоновщик формы
35.     Thread thread;                 // Объект рабочего потока
36. };
37.
38. #endif
```

Рис. 77. Объявление класса главного окна приложения.

9. Файл исходного кода `mainwindow.cpp` содержит реализации методов классов `MainWindow` и `Thread` (рис. 78).

```
1.  #include "mainwindow.h"
2.
3.  void Thread::run()
4.  {
5.      value = 10;                                // Инициализация счётчика
6.      QTimer timer;                               // Объект таймера
7.      connect(&timer, SIGNAL(timeout()), SLOT(NextValue())); // Обработчик сигнала таймера
8.      timer.start(1000);                           // Сигнал каждую секунду
9.      exec();                                       // Запуск потока
10. }
11.
12. void Thread::NextValue()
13. {
14.     emit CurrentValue(--value);                  // Уменьшение счётчика
15.     if(!value) emit Finished();                  // Завершение приложения
16. }
17.
18. MainWindow::MainWindow() : QWidget()
19. {
20.     /*-----
21.         Установка связей между сигналами и слотами
22.     -----*/
23.     connect(&thread, SIGNAL(CurrentValue(int)), &lcd, SLOT(display(int)));
24.     connect(&thread, SIGNAL(Finished()), qApp, SLOT(quit()));
25.     lcd.display(10);                             // Инициализация счётчика
26.     lcd.setSegmentStyle(QLCDNumber::Filled);      // Стил счётчика
27.     layout.addWidget(&lcd);                       // Добавление счётчика на форму
28.     setLayout(&layout);                           // Установка компоновщика
29.     resize(640, 480);                             // Изменение размеров окна
30.     thread.start();                               // Запуск потока
31. }
```

Рис. 78. Определение классов рабочего потока и главного окна приложения.

10. На рис. 79 показан результат работы приложения. В главном окне программы содержится виджет счётчика для индикации оставшегося времени до завершения работы приложения. Каждую секунду таймер внутри запущенного потока генерирует сигнал и в этот момент значение счётчика уменьшается на единицу. Когда значение счётчика становится равным нулю, приложение завершается.

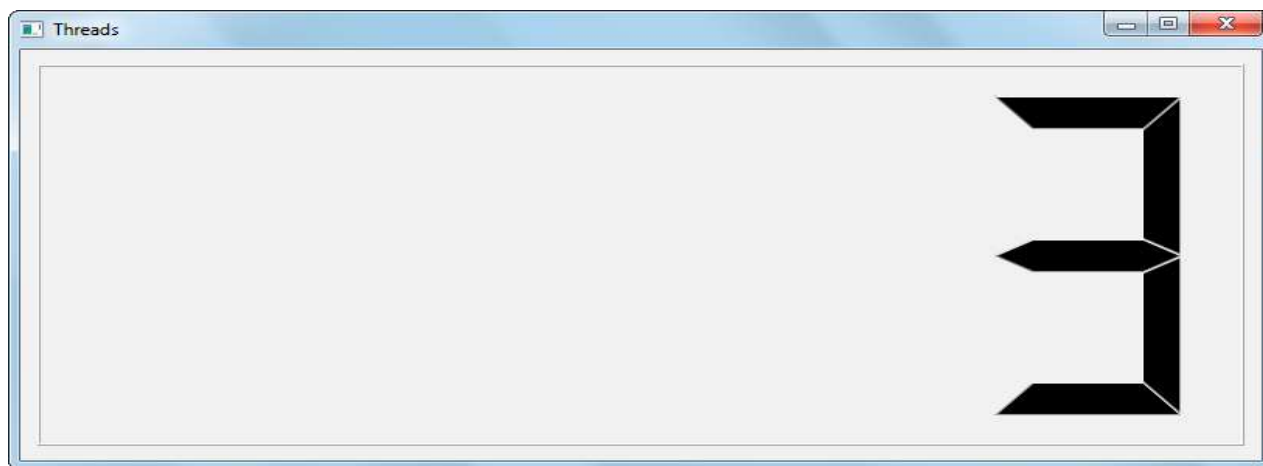


Рис. 79. Результат работы программы.

Вопросы

1. Что такое многопоточность?
2. Для чего служат механизмы синхронизации в многопоточных программах?
3. Какие механизмы предоставляет Qt для разработки многопоточных программ?

Урок № 8

Работа с базами данных.

Цель урока: Изучить основные принципы работы с базами данных в Qt, познакомиться с особенностями доступа к данным с помощью классов библиотеки, научиться выводить таблицы баз данных в приложении Qt для просмотра и редактирования.

Теоретическая часть

Задача длительного хранения и обработки информации появилась практически сразу с момента начала использования первых компьютеров. Базы данных представляют собой хранилища структурированной информации, исключающие ненужное дублирование данных и позволяющие эффективно получать доступ к ним. Различные технологии баз данных начали развиваться потому, что у файловых систем компьютеров было множество недостатков. Специально для управления базами данных были созданы специальные программные комплексы, которые в настоящее время используются во всех крупных организациях.

База данных – набор сведений, хранящихся некоторым упорядоченным способом. Сами по себе базы данных не представляли бы интереса, если бы не системы управления базами данных (СУБД). Базы данных бывают нескольких типов: иерархические, реляционные, объектно-ориентированные, гибридные. Реляционные базы данных получили наибольшее распространение, поэтому далее рассматривается только этот тип баз данных.

Система управления базами данных (СУБД) – это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, изменять и удалять, обеспечивает безопасность данных и многое другое. СУБД – это система, осуществляющая управление базами данных. Средством доступа к данным практически во всех СУБД является специальный язык SQL (англ. *structured query language* – «язык структурированных запросов»).

SQL – формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей СУБД. SQL обладает огромным спектром возможностей для управления базами данных. В настоящее время SQL является стандартизированным инструментом, не зависящим от конкретной СУБД. Со времени своего появления SQL претерпел множество изменений, в спецификацию этого языка было добавлено множество полезных конструкций. Фактически знание SQL позволяет работать с базами данных без использования других средств СУБД.

Реляционная база данных представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определённого вида. Каждая строка таблицы содержит данные об одном объекте (например, человеке, книге, предприятии), а столбцы таблицы содержат различные характеристики этих объектов – атрибуты (например, дата рождения человека, название книги, адрес предприятия).

Строки таблицы называют записями. Все записи таблицы имеют одинаковую структуру – они состоят из полей (элементов данных), в которых хранятся атрибуты объектов (рис. 80). Каждое поле содержит одну характеристику объекта и представляет собой заданный тип данных (текстовая строка, число, дата). Для идентификации записей используется первичный ключ. Первичным ключом называется набор полей таблицы, комбинация значений которых однозначно определяет каждую запись в таблице.

	Номер билета [PK] integer	ФИО character varying	Год рождения integer	Группа character varying
1	601228	Петров Сергей Анатольевич	1994	У-111
2	601332	Сидоров Андрей Петрович	1993	У-110
3	621201	Матвеев Олег Викторович	1995	У-112
4	972221	Иванов Дмитрий Валерьевич	1993	П-110
*				

Рис. 80. Таблица реляционной базы данных.

Библиотека Qt предоставляет средства управления базами данных, не зависящие от операционной системы и СУБД. С помощью этих средств можно просматривать отдельные таблицы баз данных, создавать запросы произвольной сложности, вызывать хранимые процедуры и многое другое. В поставку Qt входит специальный виджет `QTableView`, с помощью которого можно легко выводить данные для просмотра и редактирования. Программирование баз данных с использованием Qt является быстрым и лёгким занятием, не требующим особых усилий.

Практическая часть

1. Чтобы приступить к работе с базами данных в среде Qt, необходимо установить подходящую СУБД. Одной из самых популярных систем является PostgreSQL. Чтобы начать процедуру установки, нужно зайти на страницу загрузки СУБД enterprisedb.com/products-services-training/pgdownload и нажать на кнопку «Win x86-32» (рис. 81). Программа установки будет загружена на компьютер.

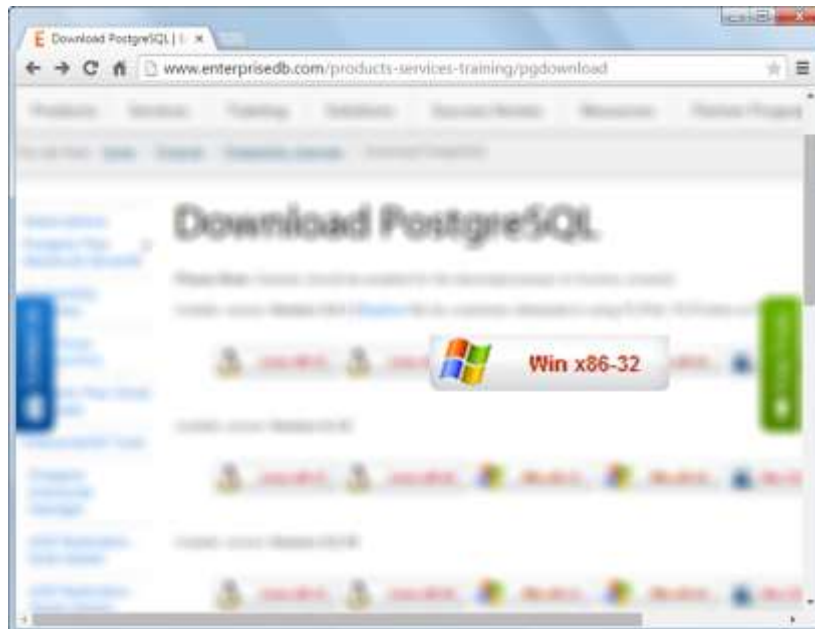


Рис. 81. Страница загрузки СУБД PostgreSQL.

2. Далее необходимо запустить загруженную программу на выполнение. Откроется мастер установки – в его первом окне нужно нажать кнопку «Next» (рис. 82).



Рис. 82. Окно приветствия мастера установки PostgreSQL.

3. В следующем окне мастера установки можно настроить путь установки СУБД (рис. 83). Этот параметр лучше не изменять, так как PostgreSQL может работать неправильно.

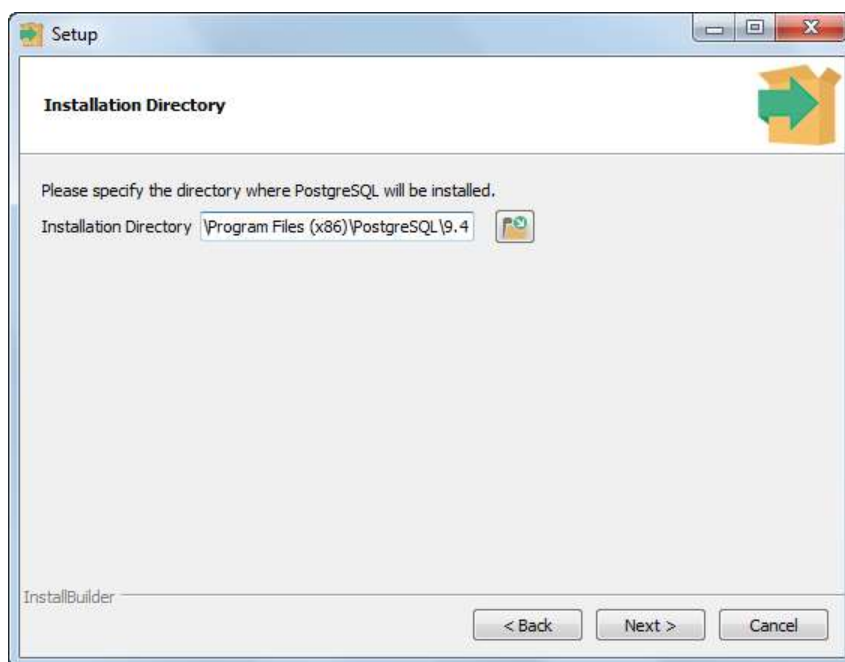


Рис. 83. Окно выбора каталога установки PostgreSQL.

4. В следующем окне мастера можно настроить каталог хранения файлов баз данных (рис. 84). Файлы в данном каталоге должны без проблем обнаруживаться СУБД и поэтому данный параметр лучше оставить без изменения. В противном случае сервер СУБД может работать неправильно.

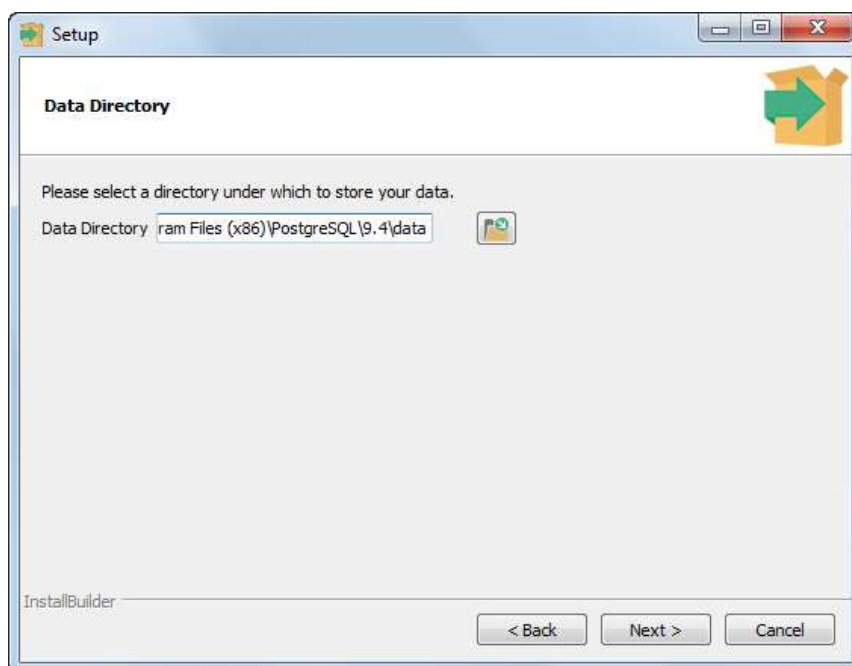


Рис. 84. Окно выбора каталога файлов баз данных.

5. В следующем окне мастера установки необходимо задать пароль для доступа к серверу баз данных (рис. 85). Сложный пароль абсолютно необходим в промышленных условиях эксплуатации систем управления базами данных, однако для данной работы подойдёт простой пароль из нескольких символов.

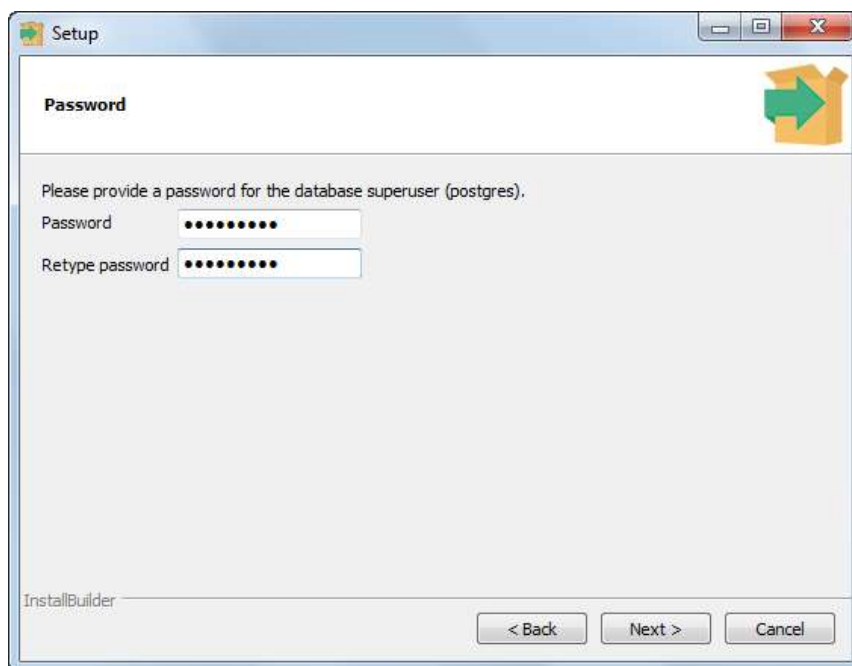


Рис. 85. Окно установки пароля СУБД.

6. В следующем окне мастера установки PostgreSQL можно задать сетевой порт, через который сервер СУБД будет взаимодействовать с клиентскими программами (рис. 86). Можно выбрать любое число в диапазоне от 1 до 65535.

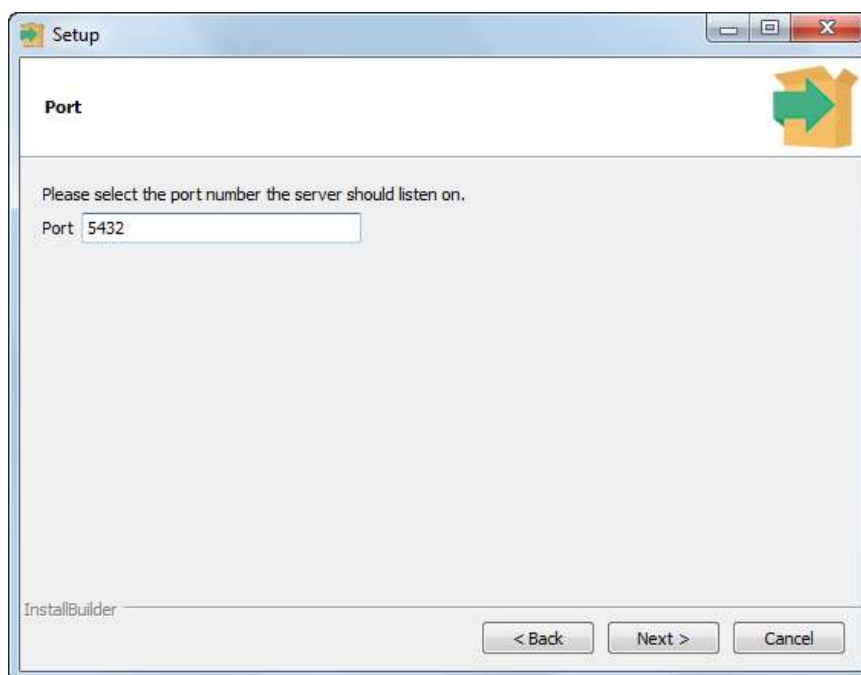


Рис. 86. Окно выбора порта сервера СУБД.

7. В следующем окне мастера установки можно выбрать локализацию нового кластера баз данных (рис. 87). В данном окне мастера следует оставить стандартные настройки.

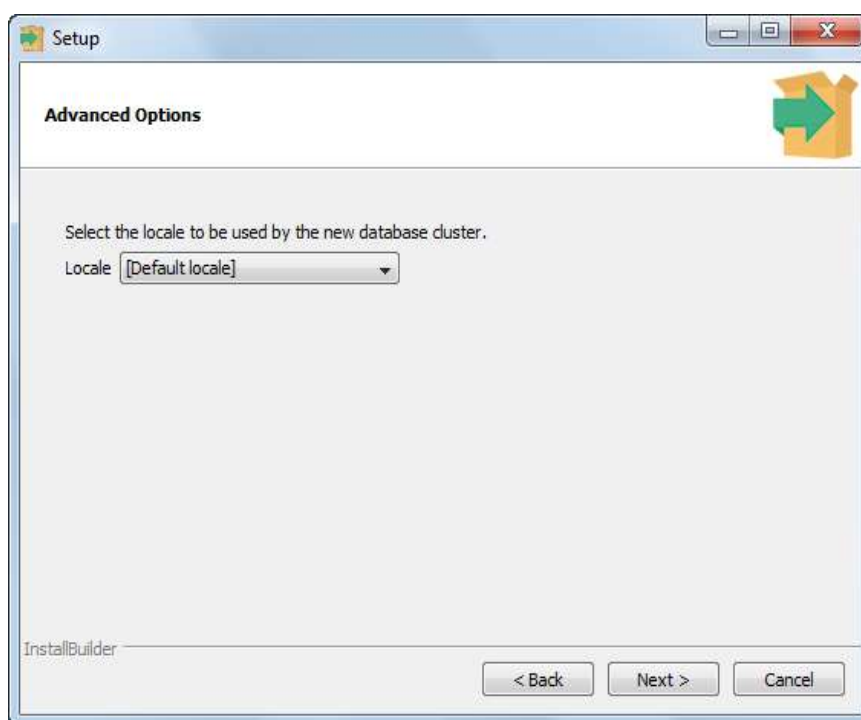


Рис. 87. Окно выбора локализации кластера баз данных.

8. В последнем окне мастера установки PostgreSQL необходимо нажать кнопку «Next» для распаковки файлов СУБД на локальный компьютер (рис. 88).

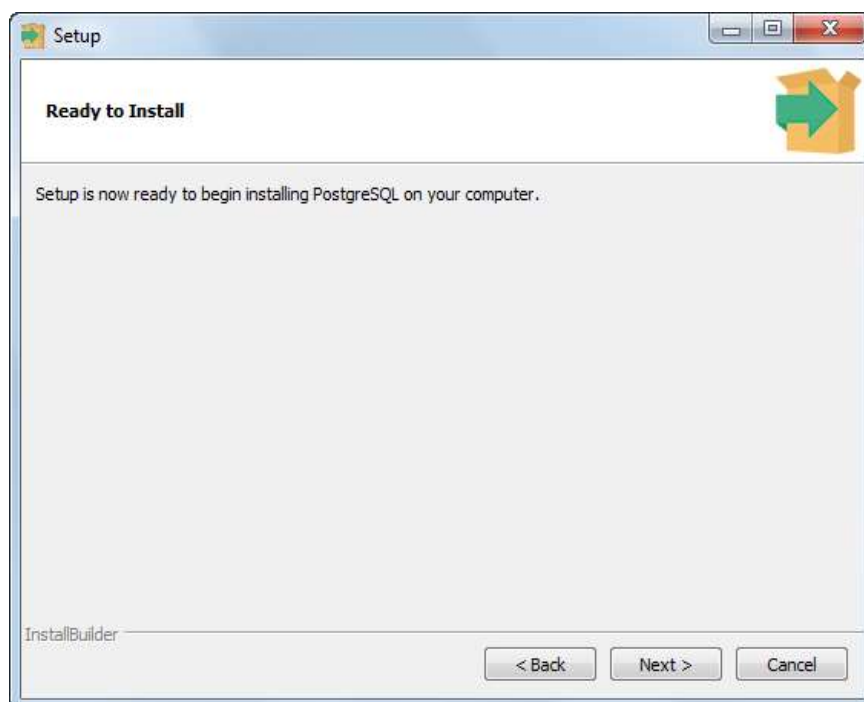


Рис. 88. Окно готовности к установке СУБД.

9. В зависимости от мощности компьютера установка СУБД PostgreSQL может занять некоторое время. На рис. 89 показано окно с индикатором прогресса установки СУБД на локальный компьютер.

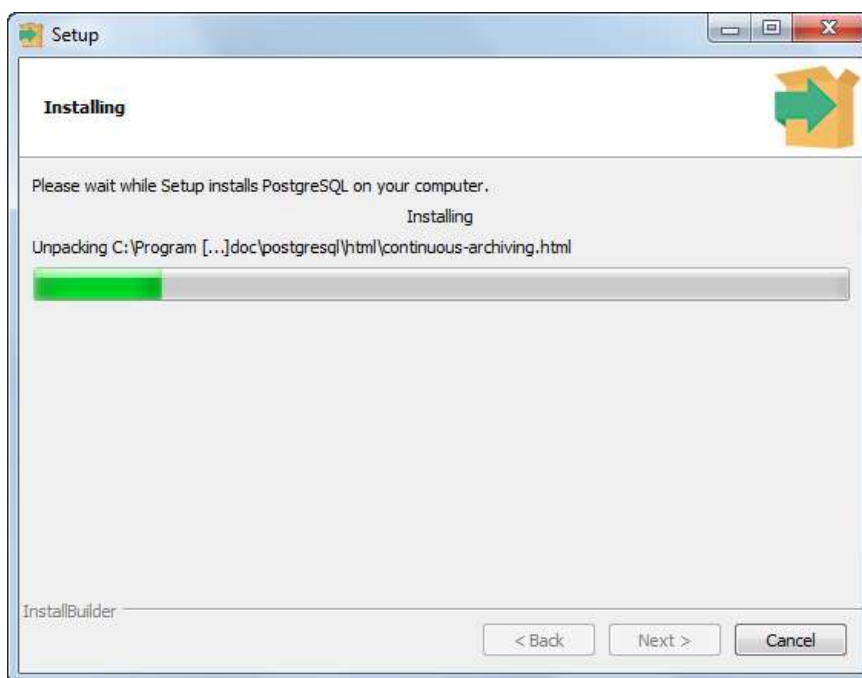


Рис. 89. Окно с индикатором установки СУБД PostgreSQL.

10. В завершающем окне мастера установки следует отказаться от дополнительных загрузок и установок (рис. 90). Мастер установки не предлагает запустить СУБД PostgreSQL после установки, поэтому начать пользоваться системой следует с помощью созданных ярлыков в меню «Пуск».

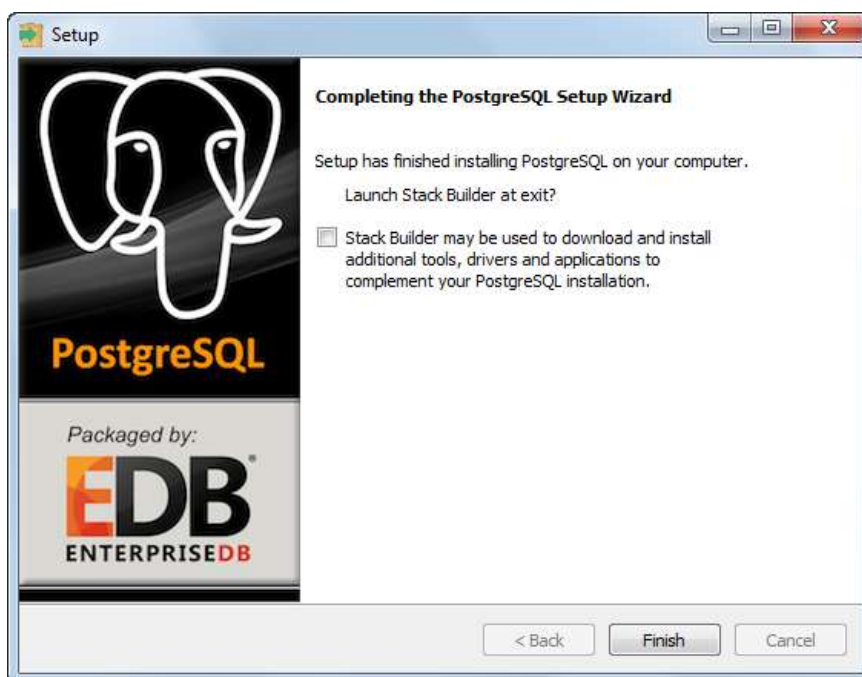


Рис. 90. Завершающее окно мастера установки.

11. На рис. 91 показано окно программы pgAdmin III, среды для работы со средствами СУБД PostgreSQL. Данное приложение позволяет управлять базами данных как в графическом режиме, так и с помощью SQL.

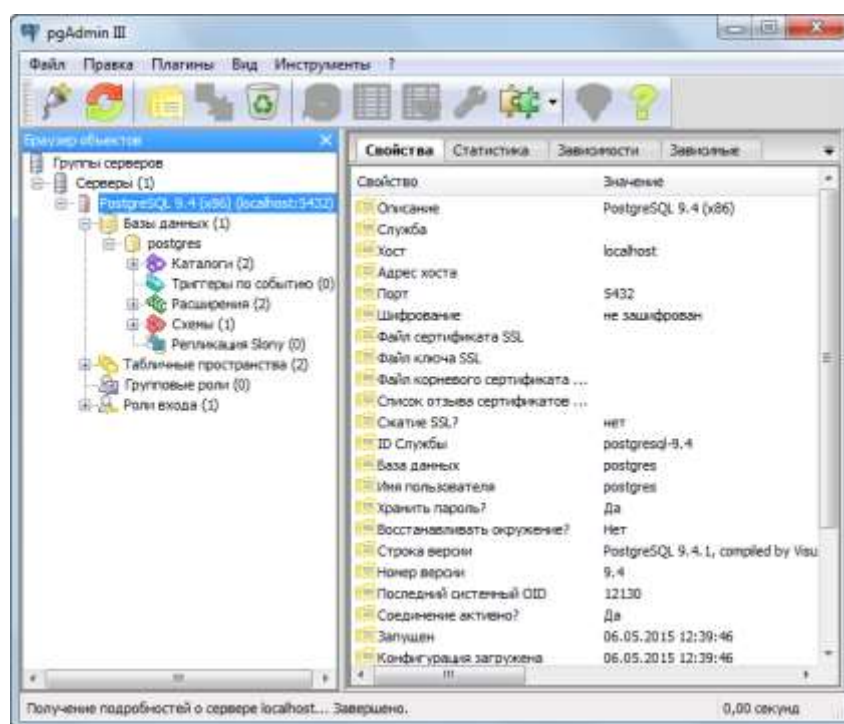


Рис. 91. Окно программы pgAdmin III.

12. Для того, чтобы среда Qt могла обращаться к СУБД для управления данными, необходимо, чтобы каталог установки PostgreSQL был расположен в известном для Qt месте. Лучший способ сделать это – добавить путь к каталогу PostgreSQL в переменную среды PATH (рис. 92).

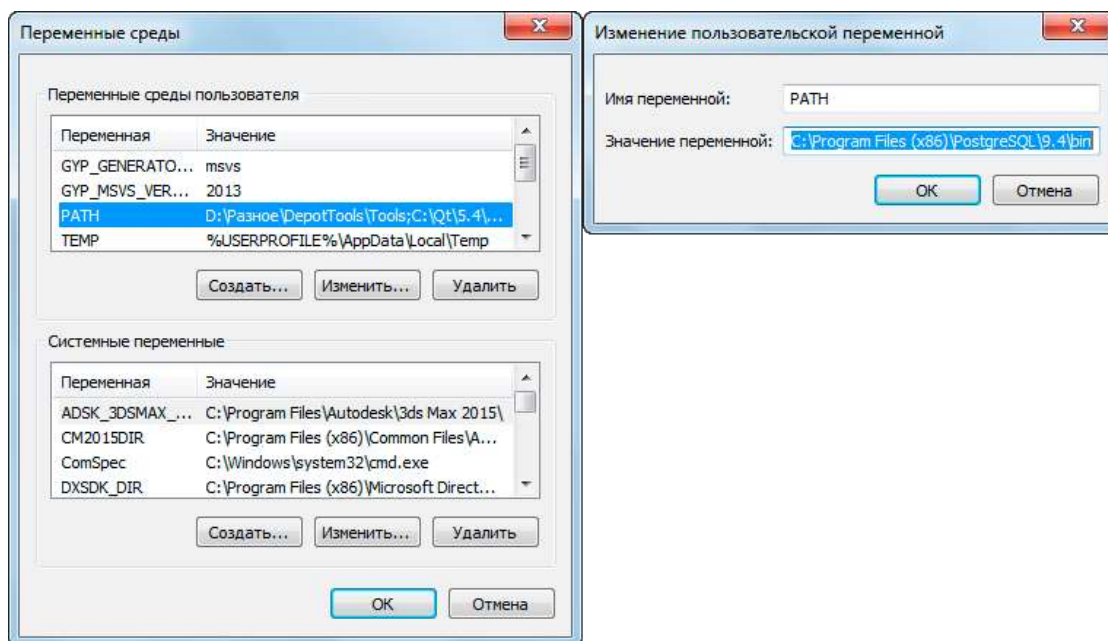


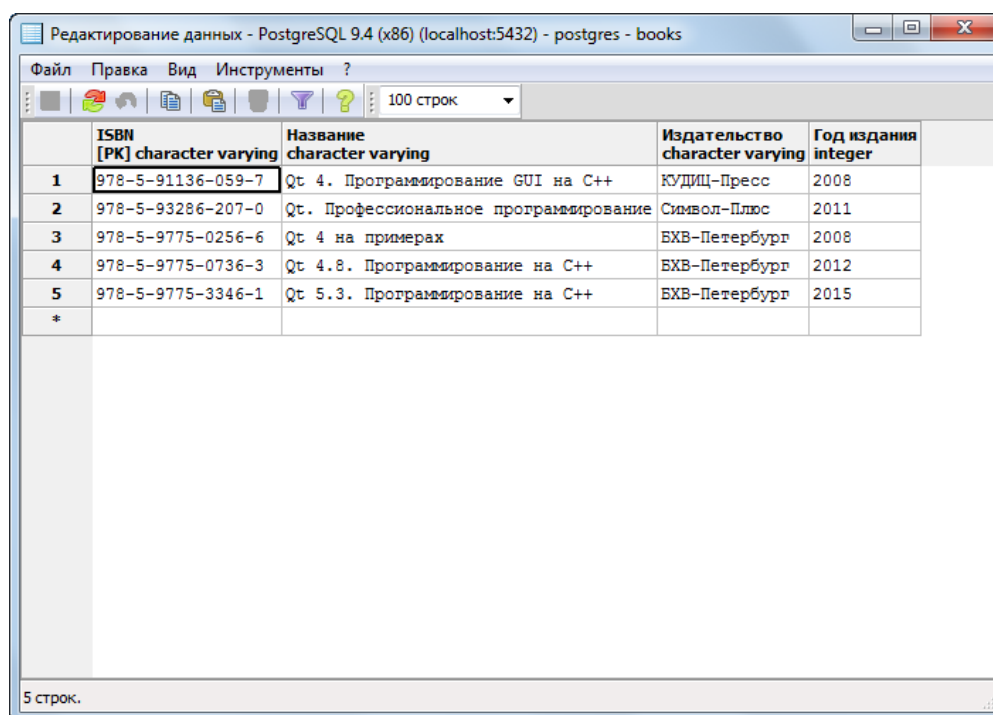
Рис. 92. Добавление каталога PostgreSQL в переменную PATH.

13. Для того, чтобы испытать работу с СУБД PostgreSQL в Qt, необходимо создать тестовую таблицу данных. Для этого нужно выполнить следующий сценарий в окне среды pgAdmin III (рис. 93).

```
1. CREATE TABLE books
2. (
3.     "ISBN" character varying NOT NULL,
4.     "Название" character varying NOT NULL,
5.     "Издательство" character varying NOT NULL,
6.     "Год издания" integer NOT NULL,
7.     CONSTRAINT "books_pkey" PRIMARY KEY ("ISBN")
8. )
9. WITH (OIDS=FALSE);
10. ALTER TABLE books OWNER TO postgres;
11. INSERT INTO books ("ISBN", "Название", "Издательство", "Год издания")
12. VALUES ('978-5-91136-059-7', 'Qt 4. Программирование GUI на C++', 'КУДИЦ-Пресс', 2008);
13. INSERT INTO books ("ISBN", "Название", "Издательство", "Год издания")
14. VALUES ('978-5-93286-207-0', 'Qt. Профессиональное программирование', 'Символ-Плюс', 2011);
15. INSERT INTO books ("ISBN", "Название", "Издательство", "Год издания")
16. VALUES ('978-5-9775-0256-6', 'Qt 4 на примерах', 'БХВ-Петербург', 2008);
17. INSERT INTO books ("ISBN", "Название", "Издательство", "Год издания")
18. VALUES ('978-5-9775-0736-3', 'Qt 4.8. Программирование на C++', 'БХВ-Петербург', 2012);
19. INSERT INTO books ("ISBN", "Название", "Издательство", "Год издания")
20. VALUES ('978-5-9775-3346-1', 'Qt 5.3. Программирование на C++', 'БХВ-Петербург', 2015);
```

Рис. 93. Сценарий SQL для создания таблицы данных.

14. В случае успешного выполнения сценария в единственной базе данных `postgres` появится новая таблица. На рис. 94 показано окно просмотра данной таблицы. При создании этой таблицы был задан первичный ключ, а также каждому полю был присвоен необязательный параметр `NOT NULL` для исключения пустых значений.



	ISBN [PK] character varying	Название character varying	Издательство character varying	Год издания integer
1	978-5-91136-059-7	Qt 4. Программирование GUI на C++	КУДИЦ-Пресс	2008
2	978-5-93286-207-0	Qt. Профессиональное программирование	Символ-Плюс	2011
3	978-5-9775-0256-6	Qt 4 на примерах	БХВ-Петербург	2008
4	978-5-9775-0736-3	Qt 4.8. Программирование на C++	БХВ-Петербург	2012
5	978-5-9775-3346-1	Qt 5.3. Программирование на C++	БХВ-Петербург	2015
*				

Рис. 94. Тестовая таблица в базе данных PostgreSQL.

15. Создание проекта приложения с использованием классов доступа к базам данных начинается с выбора типа проекта «*Приложение Qt Widgets*» в начальном окне мастера (рис. 95).

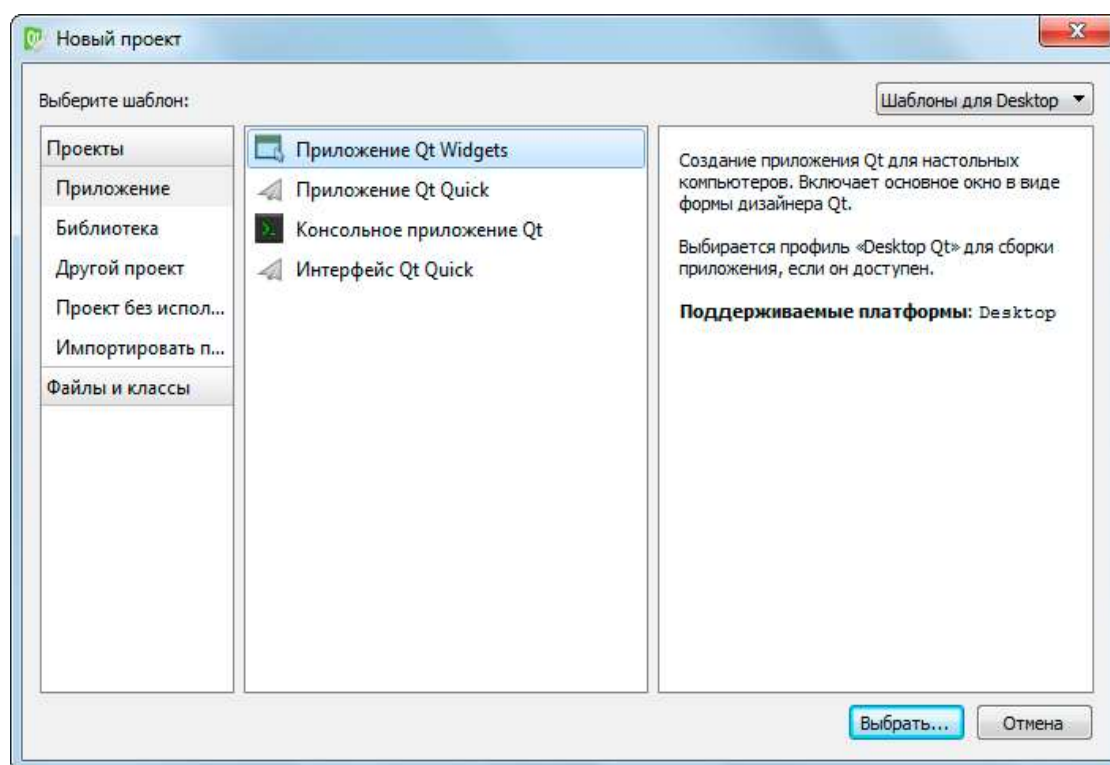


Рис. 95. Окно выбора типа проекта.

16. В следующем окне мастера нужно указать имя проекта (Database), а также папку на жёстком диске компьютера для размещения проекта (рис. 96).

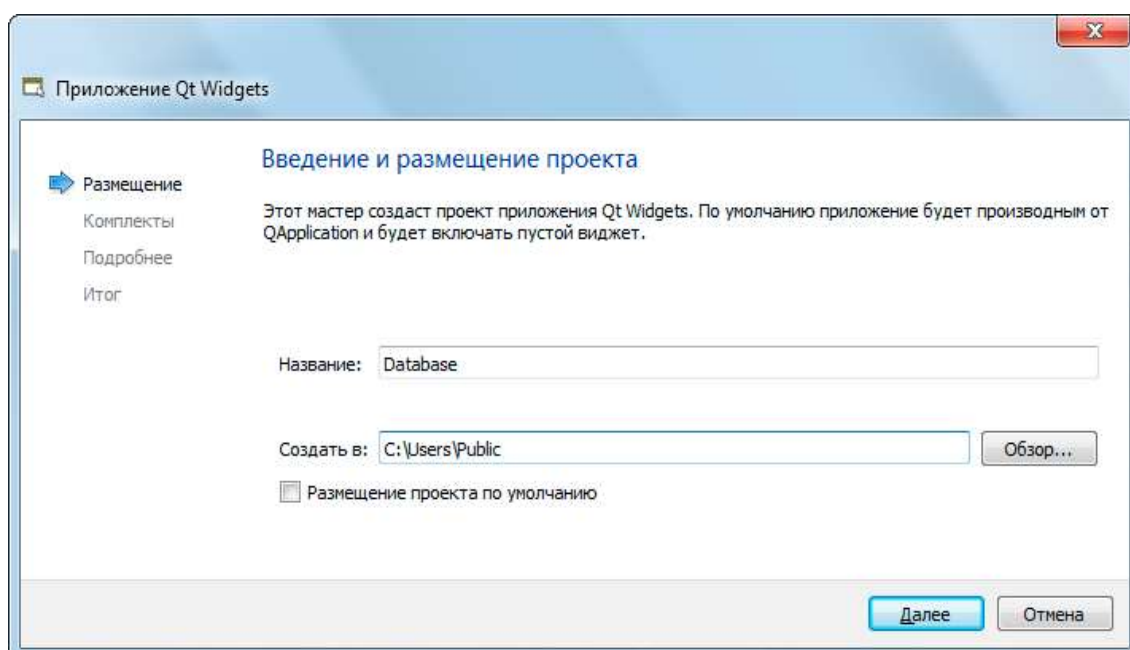


Рис. 96. Выбор имени и размещения проекта.

17. В следующем окне мастера создания проекта следует оставить параметры по умолчанию (рис. 97).

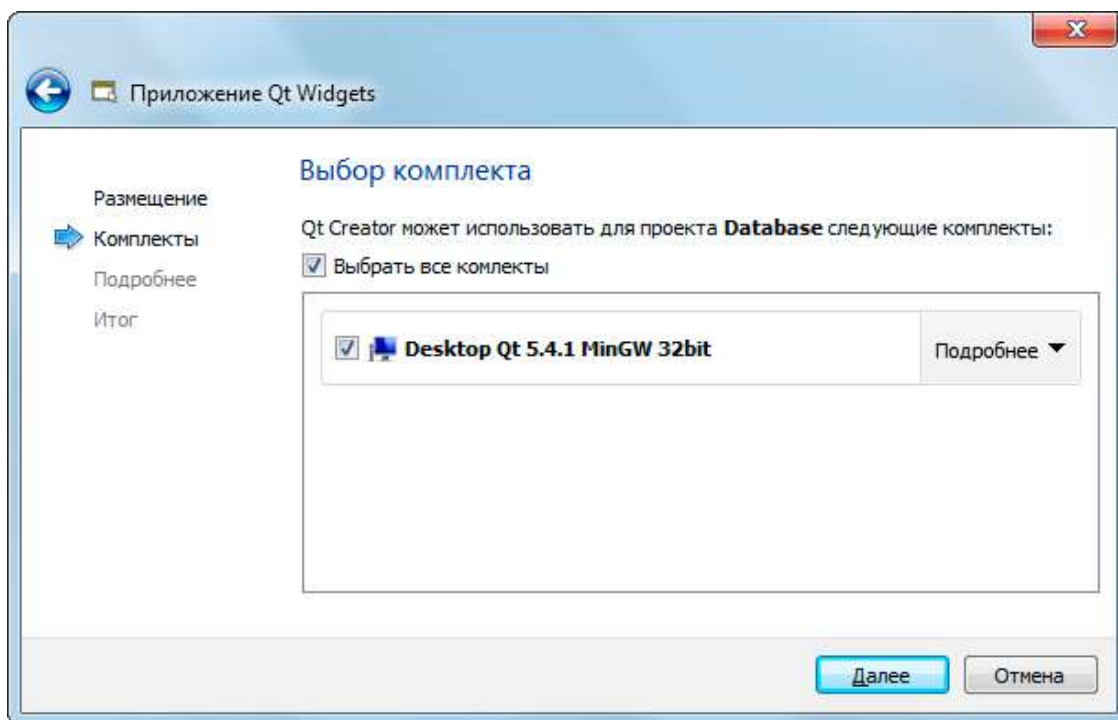


Рис. 97. Выбор комплекта сборки приложения Qt.

18. В следующем окне необходимо выбрать имя класса главного окна программы. Настройки, необходимые для выполнения данного урока, приведены на рис. 98.

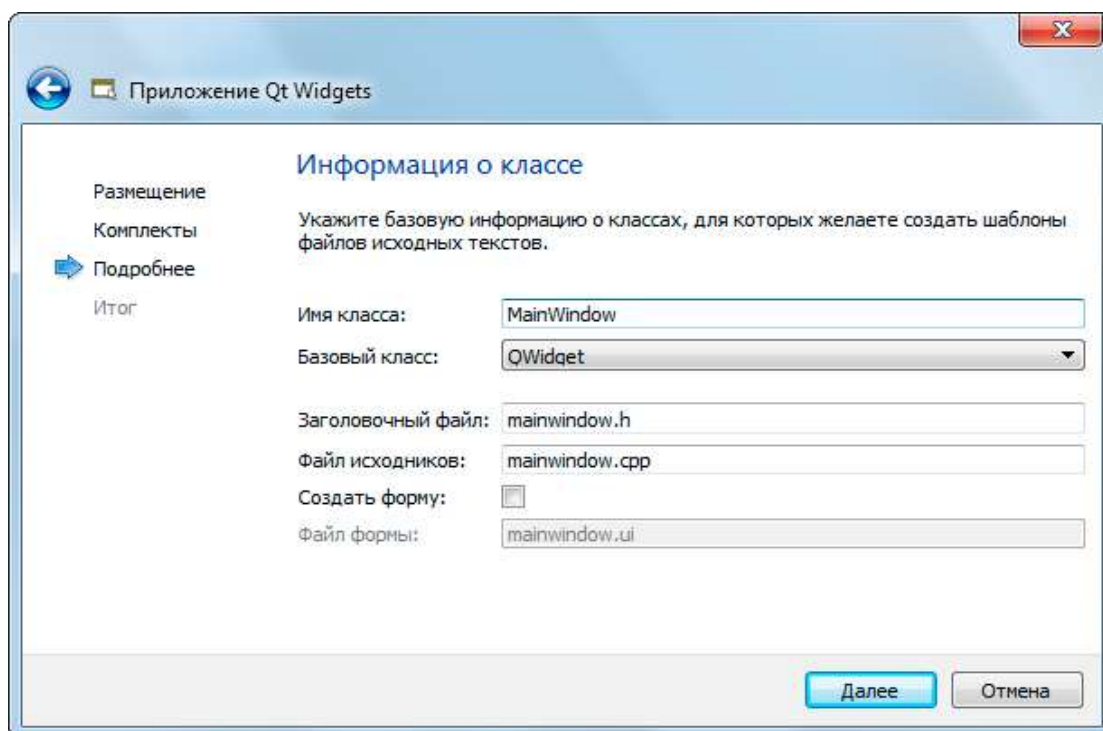


Рис. 98. Указание информации о классе формы приложения.

19. В завершающем окне мастера приведён список файлов, которые будут созданы на данном этапе (рис. 99).

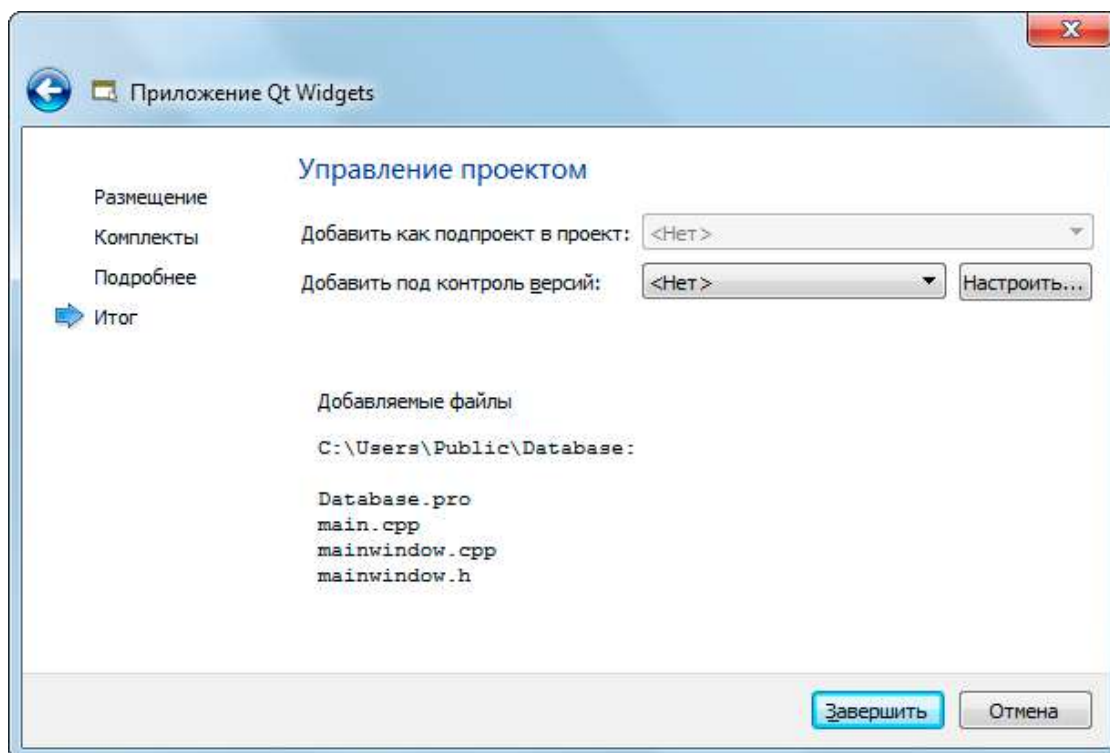


Рис. 99. Завершающий этап создания нового проекта.

20. Среда разработки Qt Creator создаст проект, содержащий заголовочные файлы, файлы исходного кода и файл проекта (рис. 100).

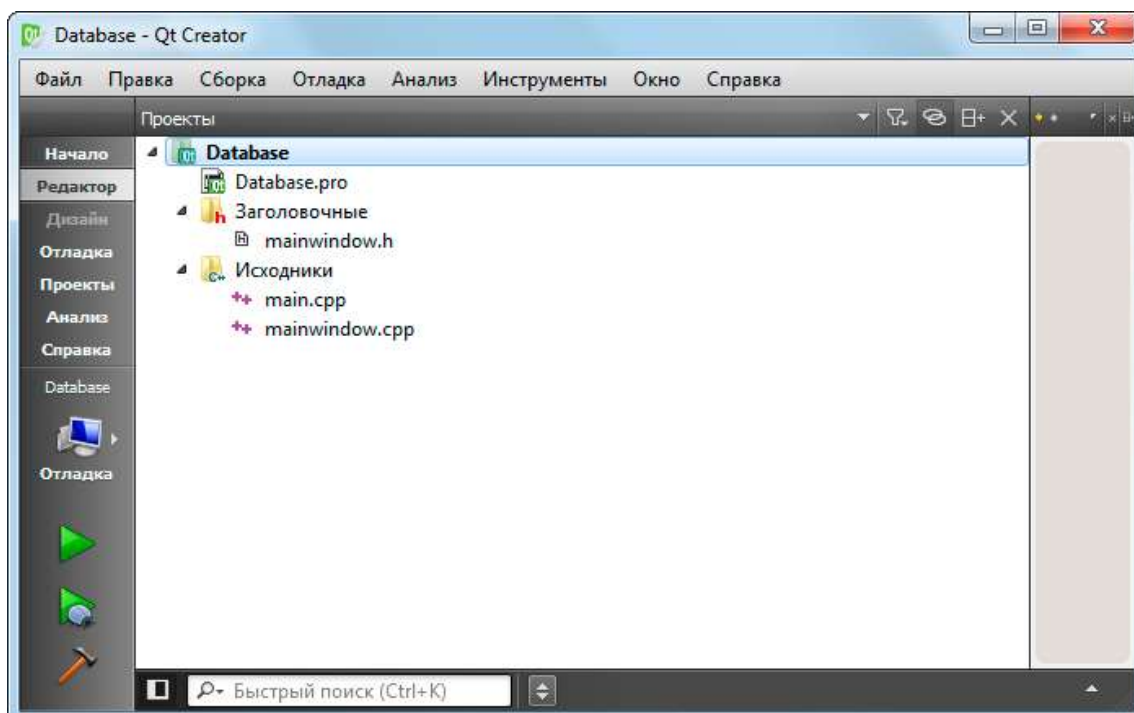


Рис. 100. Секция окна Qt Creator со структурой проекта.

21. Чтобы включить поддержку инструментов баз данных, в проект необходимо добавить модуль `sql`. На рис. 101 приведено содержимое файла `Database.pro`.

```
1. QT      += core gui widgets sql
2. TARGET  = Database
3. TEMPLATE = app
4. SOURCES += main.cpp mainwindow.cpp
5. HEADERS += mainwindow.h
```

Рис. 101. Содержимое файла проекта Qt.

22. Файл `main.cpp` содержит функцию `main` – начальную точку выполнения программы. (рис. 102). Эта функция выводит на экран главное окно программы.

```
1. #include "mainwindow.h"
2. #include <QApplication>
3.
4. int main(int argc, char *argv[])
5. {
6.     QApplication a(argc, argv);      // Создание объекта приложения
7.     MainWindow w;                    // Создание окна приложения
8.     w.show();                        // Вывод окна программы на экран
9.     return a.exec();                 // Запуск приложения
10. }
```

Рис. 102. Главная функция программы.

23. На рис. 103 приведено объявление класса `MainWindow`. Данный класс представляет главное окно программы с виджетом представления таблицы данных.

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3.
4. #include <QtWidgets>
5. #include <QtSql>
6.
7. class MainWindow : public QWidget
8. {
9.     Q_OBJECT
10. public:
11.     MainWindow();                    // Конструктор формы приложения
12.
13. private slots:
14.     void Connect();                 // Функция подключения к базе данных
15.
16. private:
17.     QVBoxLayout layout;              // Компоновщик формы
18.     QPushButton button;              // Кнопка подключения
19.     QLineEdit login;                 // Поле ввода логина
20.     QLineEdit password;              // Поле ввода пароля
21.     QTableView view;                 // Виджет отображения таблицы
22. };
23.
24. #endif
```

Рис. 103. Объявление класса окна приложения.

24. Файл исходного кода `mainwindow.cpp` содержит реализации методов класса `MainWindow` (рис. 104).

```

1.  #include "mainwindow.h"
2.
3.  MainWindow::MainWindow() : QWidget()
4.  {
5.      button.setText("Подключить");           // Название кнопки
6.      password.setEchoMode(QLineEdit::Password); // Символы-точки пароля
7.      password.setPlaceholderText("Введите пароль"); // Подсказка ввода пароля
8.      password.setAlignment(Qt::AlignCenter); // Выравнивание текста
9.      login.setPlaceholderText("Введите логин"); // Подсказка ввода логина
10.     login.setAlignment(Qt::AlignCenter); // Выравнивание текста
11.     setLayout(&layout); // Установка компоновщика
12.     resize(640, 480); // Изменение размера окна
13.     connect(&button, SIGNAL(released()), SLOT(Connect())); // Обработчик кнопки
14.     /*-----
15.             Добавление виджетов на форму приложения
16.     -----*/
17.     layout.addWidget(&login);
18.     layout.addWidget(&password);
19.     layout.addWidget(&button);
20.     layout.addWidget(&view);
21. }
22.
23. void MainWindow::Connect()
24. {
25.     QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL"); // Установка драйвера
26.     db.setHostName("localhost"); // Имя хоста сервера
27.     db.setDatabaseName("postgres"); // Имя базы данных
28.     db.setUserName(login.text()); // Логин
29.     db.setPassword(password.text()); // Пароль
30.     db.open(); // Открытие базы данных
31.     QSqlTableModel* m = new QSqlTableModel; // Объект таблицы
32.     m->setTable("books"); // Соединение с таблицей
33.     m->select(); // Извлечение данных
34.     view.setModel(m); // Отображение таблицы
35. }

```

Рис. 104. Определение класса главного окна приложения.

25. На рис. 105 показан результат работы приложения. Для того, чтобы подключиться к базе данных, нужно ввести логин `postgres` и пароль.

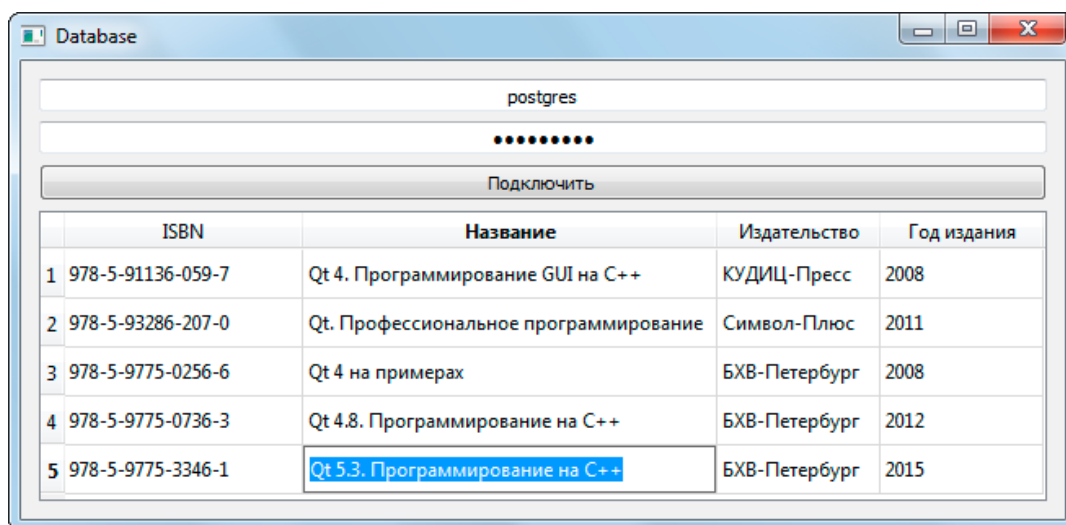


Рис. 105. Результат работы программы.

Вопросы

1. Что такое СУБД?
2. Для чего предназначен SQL?
3. Какие механизмы предоставляет Qt для управления базами данных?

Список литературы

1. Саммерфилд, М. Qt. Профессиональное программирование [Текст] / М. Саммерфилд. – СПб.: Символ-плюс, 2011. – 560 с.
2. Бланшет, Ж. Qt 4. Программирование GUI на C++ [Текст] / Ж. Бланшет, М. Саммерфилд. – СПб.: КУДИЦ-Пресс, 2008. – 718 с.
3. Шлее, М. Qt 4.8. Профессиональное программирование на C++ [Текст] / М. Шлее. – СПб.: БХВ-Петербург 2012. – 894 с.
4. Шлее, М. Qt 5.3. Профессиональное программирование на C++ [Текст] / М. Шлее. – СПб.: БХВ-Петербург 2015. – 928 с.
5. Земсков, Ю.В. Qt 4 на примерах [Текст] / Ю.В. Земсков. – СПб.: БХВ-Петербург 2008. – 608 с.