А.О. Семкин, С.Н. Шарангович

ИНФОРМАТИКА

Руководство к лабораторной работе «Создание GUI в Qt Creator. Механизм сигналов и слотов»

2015

Министерство образования и науки Российской федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

(ТУСУР)

Кафедра сверхвысокочастотной и квантовой радиотехники
(СВЧиКР)

А.О. Семкин, С.Н. Шарангович

ИНФОРМАТИКА

Руководство к лабораторной работе «Создание GUI в Qt Creator. Механизм сигналов и слотов»

2015

УДК 004.4+519.6

Рецензент:
профессор каф..СВЧиКР, А.Е. Мандель

А.О. Семкин, С.Н. Шарангович

Информатика: Руководство к лабораторной работе «Создание GUI в Qt Creator. Механизм сигналов и слотов» / А.О. Семкин, С.Н. Шарангович. – Томск: ТУСУР, 2015. – 16 с.

В данном руководстве изложены принципы работы в среде разработки *Qt Creator*. Приведено описание механизма сигналов и слотов. Представлены методические материалы компьютерной лабораторной работы, посвященной созданию графического пользовательского интерфейса (GUI) приложений.

Предназначено для студентов очной, заочной, и вечерней форм обучения по направлению подготовки бакалавриата «Инфокоммуникационные технологии и системы связи». – 11.03.02.

УДК 004.4+519.6

Оглавление

1. Цель работы

Целью данной работы является изучение принципов работы и возможностей среды *Qt Creator* для создания графического пользовательского интерфейса приложений, а также изучение механизма сигналов и слотов.

2. Введение

Qt представляет собой комплексную рабочую среду, предназначенную для разработки на C++ межплатформенных приложений с графическим пользовательским интерфейсом по принципу «написал программу – компилируй ее в любом месте». Qt позволяет программистам использовать дерево классов с одним источником в приложениях, которые будут работать в системах Windows, Mac OS X, Linux. Solaris, HP-UX и во многих других.

Графический пользовательский интерфейс (GUI — Graphical User Interface) это средства позволяющие пользователям взаимодействовать с аппаратными составляющими компьютера комфортным и удобным для себя образом.

В рамках данной лабораторной работы будут рассмотрены принципы программирования приложений с графическим пользовательским интерфейсом с использованием возможностей Qt.

3. Простая Qt-программа

Рассмотрим простую Qt-программу. Разберем каждую строку этой программы.

```
1 #include <QApplication>
2 #include <QLabel>
3 int main(int argc, char *argv[])
4 {
5     QApplication app(argc, argv);
6     QLabel *label = new QLabel("Hello Qt!");
7     label->show();
8     return app.exec();
9 }
```

В строках 1 и 2 в программу включаются определения классов QApplication и QLabel. Для каждого Qt-класса имеется заголовочный файл с тем же именем (с учетом регистра), содержащий определение этого класса.

В строке 5 создается объект QApplication для управления всеми ресурсами приложения. Для конструктора QApplication необходимо указывать параметры argc и argv, поскольку Qt сама обрабатывает некоторые из аргументов командной строки.

В строке 6 создается виджет текстовая метка QLabel, который выводит на экран сообщение "Hello Qt!" (здравствуй, Qt). По терминологии Qt и Unix виджетом (widget) называется любой визуальный элемент графического интерфейса пользователя. Этот термин происходит от "window gadget" и

соответствует элементу управления ("control") и контейнеру ("container") по терминологии Windows. Кнопки, меню, полосы прокрутки и фреймы являются примерами виджетов. Одни виджеты могут содержать в себе другие виджеты. Например, окно приложения обычно является виджетом, содержащим QMenuBar (панель меню), несколько QToolBar (панель инструментов), QStatusBar (строка состояния) и некоторые другие виджеты. Большинство приложений используют QMainWindow или QDialog в качестве окна приложения, однако Qt настолько гибка, что любой виджет может быть окном. В данном примере QLabel является окном приложения.

Строка 7 делает текстовую метку видимой. Виджеты всегда создаются сначала невидимыми, и поэтому до непосредственного вывода на экран вы можете настроить их и тем самым не допустить мерцания экрана.

Строка 8 обеспечивает передачу управления приложением Qt. В этом месте программа переходит в цикл обработки событий, т. е. в своего рода режим "простоя", ожидая со стороны пользователя таких действий, как щелчок мышки или нажатие клавиши на клавиатуре.

Для простоты мы не делаем вызов оператора delete для объекта QLabel в конце функции main(). Подобная утечка памяти в такой небольшой программе безвредна, поскольку после завершения программы эта память будет возвращена операционной системой.

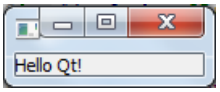


Рис. 1 – Вывод приветствия программы Hello в системе Windows 7

Заменим строку:

```
QLabel *label = new QLabel("Hello Qt!");
```

на

```
QLabel *label = new QLabel("<h2><i>Hello</i> "
                           "<font color=red>Qt!</font></h2>");
```

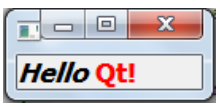


Рис. 2 – Текстовая метка с простым форматированием HTML

Как иллюстрирует этот пример, совсем не трудно выделять элементы пользовательского интерфейса Qt-приложения с использованием некоторых простых средств форматирования документов HTML.

4. Взаимодействие с пользователем

Рассмотрим возможности взаимодействия пользователя с программой. Приложение представляет собой кнопку, которую пользователь может нажать и тогда приложение закончит свою работу. Исходный код этой программы очень напоминает исходный код программы Hello, но здесь вместо QLabel используется QPushButton в качестве главного виджета и добавляется код, обеспечивающий реакцию программы на действие пользователя (нажатие кнопки).

```
 1 #include <QApplication>
 2 #include <QPushButton>
 3 int main(int argc, char *argv[])
 4 {
 5     QApplication app(argc, argv);
 6     QPushButton *button = new QPushButton("Quit");
 7     QObject::connect(button, SIGNAL(clicked()),
 8                      &app, SLOT(quit()));
 9     button->show();
10     return app.exec();
11 }
```

Виджеты Qt генерируют *сигналы* в ответ на выполнение пользователем какого-то действия или изменение состояния. Например, QPushButton генерируют сигнал clicked() при нажатии пользователем кнопки. Сигнал может быть связан с функцией (называемой слотом в данном контексте) для автоматического ее выполнения при получении данного сигнала. В данном примере мы связываем сигнал кнопки clicked() со слотом quit() объекта приложения QApplication. Макросы SIGNAL() и SLOT() являются частью синтаксиса; более подробно они объясняются в разделе 6.

Рис. 3 – Приложение Quit.

5. Компоновка виджетов

В данном разделе создается приложение, которое демонстрирует применение менеджеров компоновки для размещения виджетов в окне и использование сигналов и слотов для синхронизации работы двух виджетов. Приложение предлагает пользователю указать свой возраст, что можно сделать при помощи либо наборного счетчика (spin box), либо ползунка (slider).

Это приложение состоит из трех виджетов: QSpinBox, QSlider и QWidget. QWidget является главным окном приложения. Виджеты QSpinBox и QSlider помещены внутрь QWidget, и они являются дочерними виджетами по отношению к QWidget. С другой стороны, QWidget является родительским виджетом по отношению к QSpinBox и QSlider. Сам QWidget не имеет

родителя, потому что используется в качестве окна самого верхнего уровня. Конструкторы QWidget и все его подклассы принимают параметр QWidget *, задающий родительский виджет.

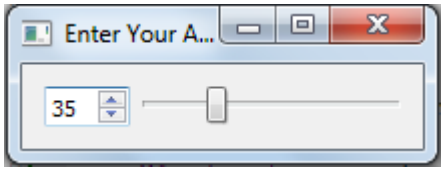


Рис. 4 – Приложение Age.

Ниже приводится исходный код:

```
 1 #include <QApplication>
 2 #include <QHBoxLayout>
 3 #include <QSlider>
 4 #include <QSpinBox>
 5 int main(int argc, char *argv[])
 6 {
 7     QApplication app(argc, argv);
 8     QWidget *window = new QWidget;
 9     window->setWindowTitle("Enter Your Age");
10     QSpinBox *spinBox = new QSpinBox;
11     QSlider *slider = new QSlider(Qt::Horizontal);
12     spinBox->setRange(0, 130);
13     slider->setRange(0, 130);
14     QObject::connect(spinBox, SIGNAL(valueChanged(int)),
15                      slider, SLOT(setValue(int)));
16     QObject::connect(slider, SIGNAL(valueChanged(int)),
17                      spinBox, SLOT(setValue(int)));
18     spinBox->setValue(35);
19     QHBoxLayout *layout = new QHBoxLayout;
20     layout->addWidget(spinBox);
21     layout->addWidget(slider);
22     window->setLayout(layout);
23     window->show();
24     return app.exec();
25 }
```

Строки 8 и 9 создают и настраивают виджет QWidget, который является главным окном приложения. Вызываются функции setWindowTitle() для вывода текстовой строки в заголовке окна.

Затем устанавливаем промежуток (в 6 пикселей) между дочерними виджетами и вокруг них.

Строки 10 и 11 создают виджеты QSpinBox и QSlider, а строки 12 и 13 устанавливают допустимый диапазон изменения их значений. Можем допустить, что возраст человека не будет превышать 130 лет. Мы могли бы передать window в конструкторах QSpinBox и QSlider, указывая на то, что window должен быть их родительским виджетом, но здесь это делать необязательно, поскольку система компоновки определит это самостоятельно и автоматически установит родительский виджет для наборного счетчика и ползунка, как мы это увидим вскоре.

Два вызова функции QObject::connect(), выполненные в строках с 14 по 17, обеспечивают синхронизацию работы наборного счетчика и ползунка,

заставляя их всегда показывать одинаковое значение. Если один из виджетов изменяет значение, то генерируется сигнал valueChanged(int) и вызывается слот setValue(int) другого виджета с новым значением возраста.

В строке 18 наборный счетчик устанавливается в значение 35. В результате виджет QSpinBox генерирует сигнал valueChanged(int) с целочисленным аргументом 35. Этот аргумент передастся слоту setValue(int) виджета QSlider, и в результате ползунок устанавливается в значение 35. Ползунок затем также генерирует сигнал valueChanged(int), поскольку его значение изменилось, и вызывает слот setValue(int) наборного счетчика. Но на этот раз функция setValue(int) не будет генерировать сигнал, поскольку наборный счетчик уже имеет значение 35. Это не позволяет повторять эти действия бесконечно. Описанная ситуация продемонстрирована на рис. 5.

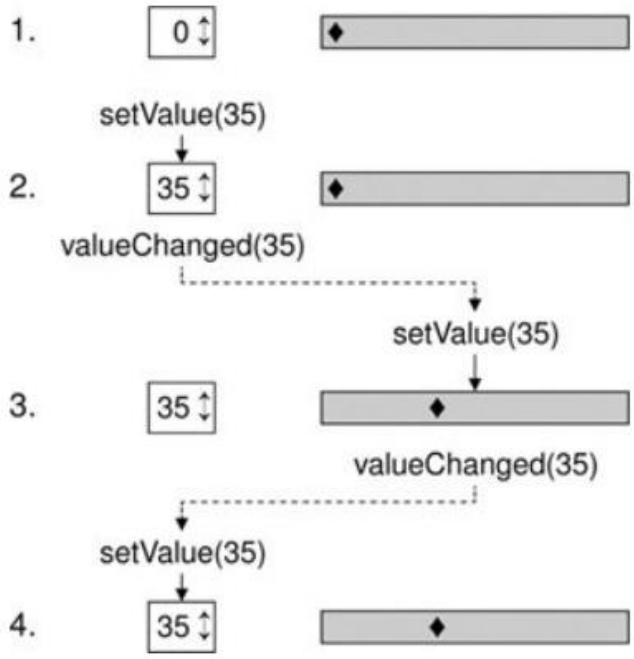


Рис. 5 – Изменение значения в одном из виджетов приводит к изменению значения в другом виджете.

В строках с 19 по 22 мы размешаем виджеты наборного счетчика и ползунка, используя менеджер компоновки. Менеджер компоновки - это объект, который устанавливает размер и положение виджетов, которые располагаются в зоне его действия. Qt имеет три основных класса менеджеров компоновки:

- `QHBoxLayout` размещает виджеты по горизонтали слева направо (или справа налево);
- `QVBoxLayout` размещает виджеты по вертикали сверху вниз;
- `QGridLayout` размещает виджеты в ячейках сетки.

Выполненный в строке 22 вызов QWidget::setLayout() устанавливает менеджер компоновки для окна. За кулисами создаются дочерние связи QSpinBox и QSlider с виджетом, для которого установлен менеджер

компоновки, и по этой причине нам не требуется в явной форме задавать родительский виджет при конструировании виджета, размещаемого в зоне действия менеджера компоновки.

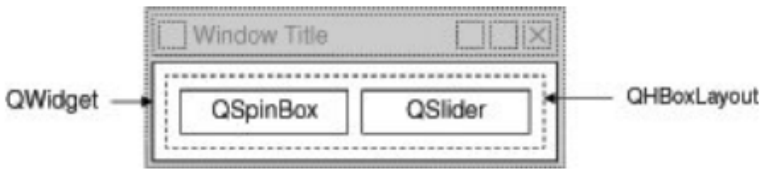


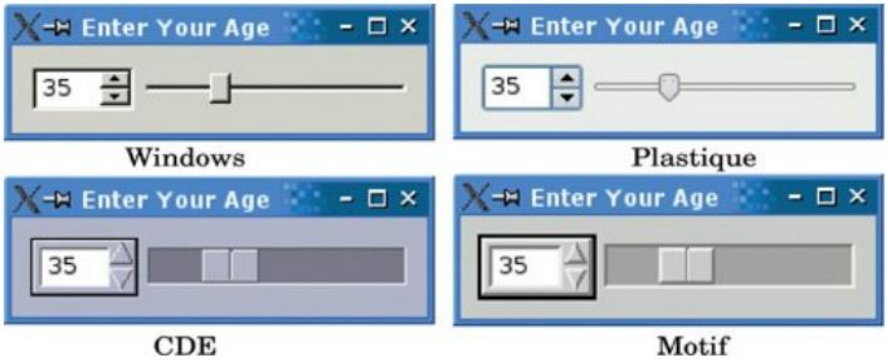
Рис. 6 – Виджеты приложения Age.

Несмотря на то что мы не задавали в явной форме положение и размер ни одного из виджетов, QSpinBox и QSlider аккуратно расположились в ряд. Это объясняется тем, что QHBox-Layout автоматически определяет разумные размеры и положение виджетов, попадающих в зону его действия, в зависимости от потребностей этих виджетов. Менеджеры компоновки освобождают нас от кодирования размещения виджетов нашего приложения на экране и гарантируют плавное изменение размеров окон.

Используемый средствами разработки Qt подход к построению графического пользовательского интерфейса легко понятен и очень гибок. Среди работающих в Qt программистов наиболее распространен подход, когда сначала создаются все необходимые графические элементы, а затем соответствующим образом настраиваются их свойства. Программисты добавляют виджеты к компоновщикам графических элементов, которые автоматически устанавливают для них нужные размер и положение. Управление работой графического интерфейса осуществляется через взаимодействие виджетов друг с другом посредством применения механизма сигналов и слотов Qt.

Стили виджетов

Показанные ранее экраны были взяты из системы Windows 7, но приложения Qt будут выглядеть привычно для любой поддерживаемой платформы. Qt имитирует изобразительные средства используемой платформы, а не делает попытки все представить средствами, принятыми в какой-то одной платформе или каким-то одним инструментарием.



Различные стили вывода графических элементов

В Qt/X11 и Qtopia Core по умолчанию используется стиль Plastique, который применяет плавные переходы цветов и подавление помех спектрального наложения для обеспечения современного интерфейса пользователя. Пользователи приложений Qt могут переопределять принятый по умолчанию стиль, используя опцию -style в команде запуска приложения. Например, для запуска приложения Age со стилем Motif в X11 необходимо просто задать команду

./age -style motif

в командной строке.

Зависимые от платформы стили

В отличие от других, стили систем Windows XP и Mac доступны только на родных платформах, поскольку они реализованы на базе присущих только данной платформе механизмов работы.

6. Механизм сигналов и слотов

Механизм сигналов и слотов играет решающую роль в разработке программ Qt. Он позволяет прикладному программисту связывать различные объекты, которые ничего не знают друг о друге. Мы уже соединяли некоторые сигналы и слоты, объявляли наши собственные сигналы и слоты, реализовывали наши собственные слоты и генерировали наши собственные сигналы. Рассмотрим этот механизм более подробно.

Слоты почти совпадают с обычными функциями, которые объявляются внутри классов C++ (функции-члены). Они могут быть виртуальными, они могут быть перегруженными, они могут быть открытыми (public), защищенными (protected) и закрытыми (private), они могут вызываться непосредственно, как и любые другие функции-члены C++ и их параметры, могут быть любого типа. Однако слоты (в отличие от обычных функций-членов) могут подключаться к сигналам, и в результате они будут вызываться при каждом генерировании соответствующего сигнала.

Оператор connect() выглядит следующим образом:

```
connect(отправитель, SIGNAL(сигнал), получатель, SLOT(слот));
```

где отправитель и получатель являются указателями на объекты QObject и где сигнал и слот являются сигнатурами функций без имен параметров. Макросы SIGNAL() и SLOT() фактически преобразуют свои аргументы в строковые переменные.

В приводимых ранее примерах мы всегда подключали разные слоты к разным сигналам. Существует несколько вариантов подключения слотов к сигналам.

К одному сигналу можно подключать много слотов:

```
connect(slider, SIGNAL(valueChanged(int)),
        spinBox, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)),
        this, SLOT(updateStatusBarIndicator(int)));
```

При генерировании сигнала последовательно вызываются все слоты, причем порядок их вызова неопределен.

Один слот можно подключать ко многим сигналам:

```
connect(lcd, SIGNAL(overflow()),
        this, SLOT(handleMathError()));
connect(calculator, SIGNAL(divisionByZero()),
        this, SLOT(handleMathError()));
```

Данный слот будет вызываться при генерировании любого сигнала.

Один сигнал может соединяться с другим сигналом:

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
        this, SIGNAL(updateRecord(const QString &)));
```

При генерировании первого сигнала будет также генерироваться второй сигнал, остальном связь «сигнал-сигнал» не отличается от связи «сигнал-слот».

Связь можно аннулировать:

```
disconnect(lcd, SIGNAL(overflow()),
           this, SLOT(handleMathError()));
```

Это редко приходится делать, поскольку **Qt автоматически убирает все связи при удалении объекта.**

При успешном соединении сигнала со слотом (или с другим сигналом) их параметры должны задаваться в одинаковом порядке и иметь одинаковый тип:

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),
        this, SLOT(processReply(int, const QString &)));
```

Имеется одно исключение, а именно: если у сигнала больше параметров, чем у подключенного слота, то дополнительные параметры просто игнорируются:

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),
        this, SLOT(checkErrorCode(int)));
```

Если параметры имеют несовместимые типы либо будет отсутствовать сигнал или слот, то Qt выдаст предупреждение во время выполнения программы, если сборка программы проводилась в отладочном режиме. Аналогично Qt выдаст предупреждение, если в сигнатуре сигнала или слота будут указаны имена параметров.

До сих пор мы использовали сигналы и слоты только при работе с

виджетами. Но сам по себе этот механизм реализован в классе QObject, и его не обязательно применять только в пределах программирования графического пользовательского интерфейса. Этот механизм можно использовать в любом подклассе QObject:

```
class Employee : public QObject
{
    Q_OBJECT
public:
    Employee() { mySalary = 0; }
    int salary() const { return mySalary; }
public slots:
    void setSalary(int newSalary);
signals:
    void salaryChanged(int newSalary);
private:
    int mySalary;
};
void Employee::setSalary(int newSalary)
{
    if (newSalary != mySalary) {
        mySalary = newSalary;
        emit salaryChanged(mySalary);
    }
}
```

Обратите внимание на реализацию слота setSalary(). Мы генерируем сигнал salary-Changed() только при выполнении условия newSalary != mySalary. Это позволяет предотвратить бесконечный цикл генерирования сигналов и вызовов слотов.

Мета объектная система Qt

Одним из главных преимуществ средств разработки Qt является расширение языка C++ механизмом создания независимых компонентов программного обеспечения, которые можно соединять вместе, несмотря на то что они могут ничего не знать друг о друге.

Этот механизм называется метаобъектной системой, и он обеспечивает две основные служебные функции: взаимодействие сигналов и слотов и анализ внутреннего состояния приложения (introspection). Анализ внутреннего состояния необходим для реализации сигналов и слотов и позволяет прикладным программистам получать «метаинформацию» о подклассах QObject во время выполнения программы, включая список поддерживаемых объектом сигналов и слотов и имена их классов. Этот механизм также поддерживает свойства (для Qt Designer) и перевод текстовых значений (для интернационализации приложений), а также создает основу для системы сценариев в Qt (Qt Script for Applications – QSA).

В стандартном языке C++ не предусмотрена динамическая поддержка метаданных, необходимых системе метаобъектов Qt. В Qt эта проблема решена за счет применения специального инструментального средства компилятора moc, который просматривает определения классов с макросом Q_OBJECT и делает соответствующую информацию доступной функциям C++. Поскольку все функциональные возможности moc обеспечиваются только с помощью «чистого» C++, мета-объектная система Qt будет работать с любым компилятором C++.

Этот механизм работает следующим образом:

- макрос `Q_OBJECT` объявляет некоторые функции, которые необходимы для анализа внутреннего состояния и которые должны быть реализованы в каждом подклассе `QObject:metaObject()`, `TR()`, `qt_metacall()` и некоторые другие.
- компилятор `moc` генерирует реализации функций, объявленных макросом `Q_OBJECT`, и всех сигналов.
- такие функции-члены класса `QObject`, как `connect()` и `disconnect()`, во время своей работы используют функции анализа внутреннего состояния.

Все это выполняется автоматически при работе qmake, moc, и при компиляции QObject, и поэтому у вас крайне редко может возникнуть необходимость вспомнить об этом механизме. Однако если вам интересны детали реализации этого механизма, вы можете воспользоваться документацией по классу QMetaObject и просмотреть файлы исходного кода C++, сгенерированные компилятором moc.

7. Методические указания по выполнению работы

Лабораторную работу необходимо выполнить в следующей последовательности:

1. Изучите теоретические разделы 2-6, выполните на компьютере все приведенные в данных разделах примеры, скомпилируйте и запустите на исполнение все записанные программы. В качестве дополнительных источников информации, воспользуйтесь сведениями раздела 8, а так же указанными в нем пособиями, а также открытыми источниками в сети Интернет.
2. Последовательно самостоятельно выполните задания, приведенные в таблице 1.
3. Подготовьте исходный код каждой программы в Qt Creator и покажите его преподавателю.
4. В случае отсутствия видимых ошибок в коде, скомпилируйте программу и запустите ее на выполнение.
5. Получите оценку.

Таблица 1 – Задания для самостоятельного выполнения

№ п/п	Задание
1.	Создайте программу «Height», описывающую рост человека в см. Программа состоит из трех виджетов QSpinBox, QLabel и QWidget. QWidget является главным окном приложения. QSpinBox и QLabel отвечают за ввод роста в см. Пределы – от 50 до 250. Вводимые данные должны отображаться в обоих виджетах, аналогично приложению «Age», рассмотренному ранее.
2.	Создайте программу «Counter» («счетчик»). Программа состоит из четырех виджетов QPushButton, QSpinBox, QCheckBox и QWidget. QWidget является главным окном приложения. Нажатие кнопки QPushButton должно вызывать увеличение счетчика QSpinBox на 5 при наличии флажка QCheckBox и уменьшение счетчика QSpinBox на 5 при отсутствии флажка QCheckBox. Начальное значение счетчика – 0.
3.	Создайте программу «Slider». Программа состоит из пяти виджетов QSlider, расположенных друг под другом вертикально. QWidget является главным окном приложения. Виджеты QSlider должны быть связаны друг с другом таким образом, чтобы смещение нижнего слайдера вызывало соответствующее смещение верхнего, но с задержкой в 2 позиции. Начальное положение всех слайдеров – 0.

8. Справочная документация

Справочная документации по средствам разработки Qt является важным инструментом в руках любого разработчика Qt-программ, поскольку в ней есть все необходимые сведения по любому классу и любой функции Qt. Для более эффективного использования Qt вам необходимо хорошо разбираться в справочной документации.

Эта документация имеется в формате HTML в сети Интернет и ее можно просматривать любым веб-браузером. Вы можете также использовать программу Qt Assistant (помощник Qt) – браузер системы помощи в Qt, который обладает мощными средствами поиска и индексирования информации и поэтому быстрее находит нужную информацию и им легче пользоваться, чем веб-браузером. Программный интерфейс программы Qt Assistant приведен на рис. 7.

Справочная документация для текущей версии Qt и нескольких более старых версий можно найти в сети Интернет по адресу http://doc.trolltech.com/. На этом сайте также находятся избранные статьи из журнала Qt Quarterly (Ежеквартальное обозрение по средствам разработки Qt); этот журнал предназначается для программистов Qt и распространяется по всем коммерческим лицензиям.

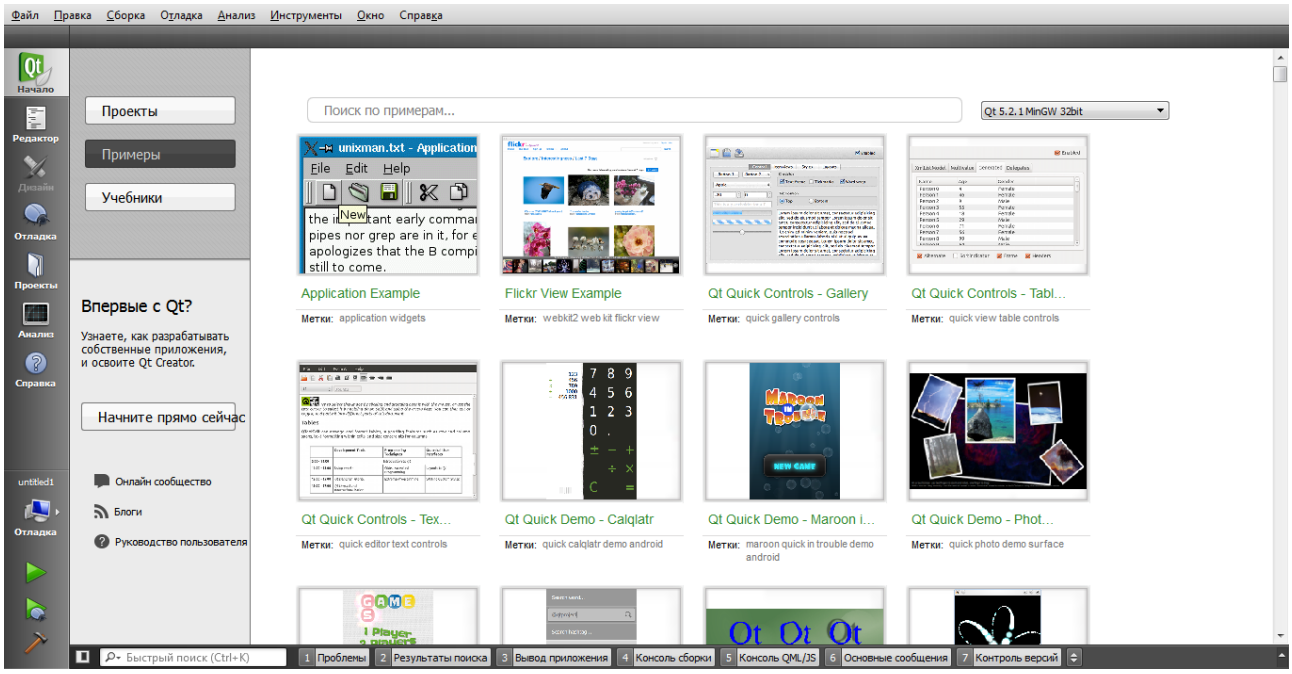


Рис. 7 – Интерфейс программы Qt Assistant в Qt версии 5.0.1 и старше.

8.1. Рекомендуемая литература

1. Qt. Профессиональное программирование на C++: Наиболее полное руководство / М. Шлее. - СПб. : БХВ-Петербург, 2005. – 544 с.
2. Бланшет Ж., Саммерфилд М. Qt 4: программирование GUI на C++. Пер. с англ. 2-е изд., доп. – М.: КУДИЦ-ПРЕСС, 2008. – 736 с.

Учебное издание

Семкин Артем Олегович

Шарангович Сергей Николаевич

ИНФОРМАТИКА

Руководство к лабораторной работе «Создание GUI в Qt Creator. Механизм сигналов и слотов»

Формат 60x84 1/16. Усл. печ. л. _____.
Тираж _____ экз. Заказ_____.
Томскиц государственный университет систем управления и радиоэлектроники.
634050, Томск, пр. Ленина, 40. Тел. (3822) 533018.