**Лекция(практика) по Qt №1**

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

Официально поддерживаются операционные системы MS Windows, Linux, MacOS, а также встраиваемые операционные системы Embedded Linux, Windows CE, Symbian. После того, как Nokia открыла исходный код Qt сообществу на Gitorious, появились различные порты. Также есть некоторые порты, которые могут быть доступны, но более не поддерживаются. Наиболее известными примерами разработки на Qt являются: программа-коммуникатор Skype, медииа-плеер VLC, Google Earth (см. http://qt.nokia.com/qt-in-use ), графический интерфейс пользователя KDE, применяемый в ОС Linux. На сайте http://www.qt-apps.org/ приводится база OpenSource проектов, использующих Qt.

В декабре 2012 состоялся официальный выпуск Qt5. Эта версия кроссплатформенного средства разработки совместима с Qt4. Перенос кода с Qt4 на Qt5 не требует много усилий. В то же время, Qt5 отличается рядом особенностей,

улучшений и большим количеством новых возможностей.

Инструментарий разработки Qt используют для создания кроссплатформенных программ. Здесь под этим утверждением мы подразумеваем программы, исходный текст которых можно скомпилировать на разных программных платформах (различные разновидности Linux, Windows, MacOS и т.д.) практически без изменений или с незначительными изменениями. Кроме того Qt используют и для разработки программ, имеющих характерный («родной», native) для программного окружения или даже собственный стилизованный интерфейс. Всё это благодаря открытому свободному программному коду, удобному и логическому API и широким возможностям применения.

Qt расширяет возможности программиста с помощью набора макросов, метаинформации(так называемая Метаобъектная система) и сигнально-слотовых соединений, но использует при этом лишь средства языка C++ и является совместимым со всеми распространёнными современными его компиляторами. Подход сигналы и слоты на самом деле, применительно к С++ реализованы во многих библиотеках, например, libsigc++, Signals и т.д., которые можно также использовать. При этом при их использовании не нужно использовать метакомпилятор.

Наряду с традиционным для предыдущих версий Qt способом создания пользовательских интерфейсов, основанный на виджетах — визуальных элементах интерфейса (кнопки, флажки, выпадающие списки, поля ввода, слайдеры и т.д.), Qt5 ставит большой акцент на использовании технологии QtQuick (который был внедрен ещё в версии 4.7). В Qt5 некоторые нововведения коснулись и синтаксиса для создания сигнально-слотовых соединений.

**Основные составляющие Qt**

Рассмотрим основные составляющие кроссплатформенного средства разработки Qt: модули и инструменты.

На рис. 1 изображены основные составляющие Qt. Модули и инструменты доступны для разработки под целевые (Reference) и другие (Other) платформы. Средства Qt разделены по назначению на отдельные части — модули. Каждый из модулей выполнен в виде отдельной библиотеки. Разработчик имеет возможность выбрать модули, которые он использует в программе. Модули имеют взаимозависимости: одни модули используют возможности, которые предоставляют другие. Основу составляют основные (Essentials) модули (по факту набор классов, которые доступны в виде API):



Рисунок 1-основные составляющие Qt

• Qt Core — основной модуль, который содержит все базовые средства Qt.  
На его основе построены все другие модули. Каждая программа созданная  
с использованием Qt, использует этот модуль;  
• Qt Network — модуль для работы с сетевыми средствами;  
• Qt Gui — модуль поддержки графического вывода на экран. В Qt4 он также  
содержит содержит набор виджетов для создания графического интерфейса пользователя. В Qt5 виджеты вынесены в отдельный модуль;  
• Qt Widgets — модуль, который содержит набор виджетов для создания  
графического интерфейса пользователя (Qt5)  
• Qt WebKit — средства работы с Веб;  
• Qt WebKit Widgets — виджеты для работы с Веб (Qt5);  
• Qt Multimedia — средства работы с мультимедийными устройствами и  
файлами;  
• Qt Multimedia Widgets — виджеты для работы с мультимедийными  
устройствами и файлами (Qt5);  
• Qt Sql — средства работы с базами данных;  
• Qt Qml — поддержка декларативной языка QML для разработки динамических визуальных интерфейсов (Qt5);

Qt Quick — поддержка создания динамических визуальных интерфейсов  
(Qt5);  
• Qt Quick Controls — использование технологии QtQuick для создания традиционного для рабочих столов графического интерфейса (Qt5);  
• Qt Quick Layouts — компоновка для элементов QtQuick (Qt5).

Существует также много дополнительных (Add-On) модулей.

**Основные принципы работы с библиотекой Qt**

Скачать и установить можно с официального сайта <https://www.qt.io>. Важно при установке на одном из шаге выбрать все требуемые инструменты и компоненты, которые вам будут необходимы, а также требуемую версию Qt. Данные компоненты могут быть добавлены и позднее, через специальную утилиту Qt maintenance tool(см.рис. 2). В нашем случае всю работу мы будем производить в Qt Creator. Можно, конечно, разрабатывать и в Visual Studio, установив для этого специальный аддон Qt для Visual Studio, но в этом случае у вас не будет кроссплатформенных файлов проектов, которые можно открыть в программе Qt Creator на другой ОС, кроме того, будет меньше удобств при работе непосредственно с Qt и QML. Нужно отметить то, что совсем недавно появилась возможность интегрирования Qt и IDE Eclipse.

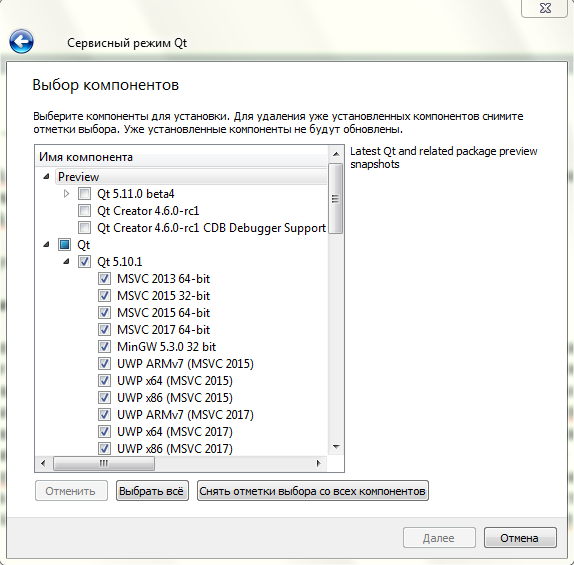


Рис.2-Qt maintenance tool

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

**Использование QtDesigner**

При необходимости быстрого получения результата, проведения экспериментов по

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

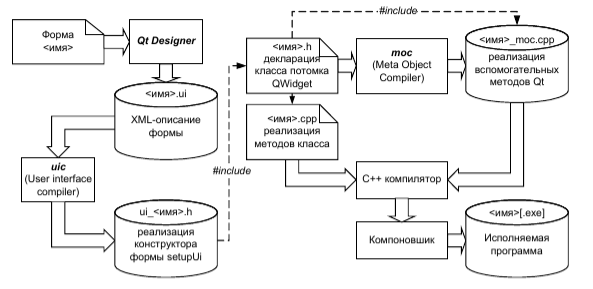


Рисунок 3 Общая схема создания исполняемого файла(может немного изменяться)

**Структура Qt-проекта на C++**

Важно! Сборка проекта осуществляется утилитой qmake.

***Файл MyProject.pro***

Главный pro-файл нашего проекта. Является по своей сути файлом конфигурации. Проектные файлы содержат всю неоходимую информацию для построения qmake Вашего приложения, библиотеки или плагина. Ресурсы, используемые Вашим проектом, определяются с помощью набора деклараций, но поддержка упрощенного программирования, позволяет Вам описать различные процессы постройки для различных платформ и окружений. Формат проектного файла, используемый qmake, может применяться и для создания простых приложений, и для создания сложных систем. Файлы простых проектов могут использовать стиль прямых деклараций, определяя стандартные переменные для указания ресурсов и заголовочных файлов используемых в проекте. Сложные проекты могут использовать структуры потоков управления для точной настройки процесса построения.

Что может находиться в файлах проекта? Переменные, комментарии, встроенные функции и поток управления, шаблоны проекта и т.д. Об этом поговорим позже.

***Файл с расширением .pri***

Общий для всех модулей файл с определениями путей и некоторых констант, задействованных при сборке.

***Main.cpp***

Файл содержащий функцию int main(int argc, char \*\*argv). Важное замечание!!!! Раньше, большинство сред разработок, которые вы использовали, допускали использование функции main без аргументов. Теперь и везде используйте только формат int main(int argc, char \*\*argv).

***Файлы с расширением .h***

Здесь могут находиться классы и т.д.

***Остальные файлы с расширением .cpp***

**Создание простейшего приложения с использованием Qt**

Открываем Qt Creator и создаём приложение QT Widgets(Можно убрать галочку «создать форму». Нам она пока что не нужна.) После создания можно удалить файлы с расширением .h и файл mainwindow.cpp.

**#include <QApplication>**

**#include <QTextEdit>**

**int** **main**(**int** argv, **char** \*\*args)

{

QApplication **app**(argv, args);

QTextEdit textEdit;

textEdit.show();

**return** app.exec();

}

Рассмотрим подробно каждую строчку кода. В первых двух, мы подключаем заголовочные файлы классов [QApplication](http://qt-project.org/doc/qt-4.8/qapplication.html) и [QTextEdit](http://qt-project.org/doc/qt-4.8/qtextedit.html), которые необходимы для работы этого примера. Каждому классу в Qt соответствует заголовочный файл с таким же названием.

В строке 5 создается объект класса [QApplication](http://qt-project.org/doc/qt-4.8/qapplication.html), который управляет основными ресурсами, необходимыми для запуска любой программы с графическим интерфейсом Qt. Ему необходимо передать аргументы argv и args функции main(), т.к. Qt использует некоторые параметры командной строки, передаваемые при запуске программы.

В 6 строке кода создается объект класса [QTextEdit](http://qt-project.org/doc/qt-4.8/qtextedit.html). QTextEdit — это визуальный элемент графического интерфейса. В Qt используются определенные виджеты — например, полосы прокрутки, метки и кнопки radio. Виджет может быть контейнером для хранения других виджетов. Наглядным примером является главное диалоговое окно программы.

В строке 7(textEdit.show();), окно редактирования текста выводится на экран в главном фрейме программы. Поскольку виджеты могут работать как контейнеры (В экземпляре класса [QMainWindow](http://qt-project.org/doc/qt-4.8/qmainwindow.html) находятся полосы панели инструментов, меню, строка состояния и несколько других виджетов), мы может отобразить только один виджет в окне нашей программы. По умолчанию, виджеты не видны. Функция show() используется для их отображения.

В строке 8, создается объект [QApplication](http://qt-project.org/doc/qt-4.8/qapplication.html), который генерирует цикл событий в процессе работы приложения. События генерируются и передаются на обработку виджетам. Примерами событий могут являться клики мыши, нажатия клавиш на клавиатуре и т.п. Когда вы вводите текст в окне редактирования виджета QTextEdit, нажатия клавиш обрабатываются средствами виджета, и вводимый текст отображается в процессе набора.

**Программа «Привет, Мир»**

#include <QApplication>

#include <QTextEdit>

#include <QLabel>

int main(int argv, char \*\*args)

{

QApplication app(argv, args);

QLabel lbl("Hello");

lbl.show();

return app.exec();

}

Qt поддерживает HTML. Можно сделать так:

#include <QApplication>

#include <QTextEdit>

#include <QLabel>

int main(int argv, char \*\*args)

{

QApplication app(argv, args);

QLabel lbl("<H2>Hello</H2>");

lbl.show();

return app.exec();

}

Попробуйте выполнить такой код

#include <QApplication>

#include <QTextEdit>

#include <QLabel>

int main(int argv, char \*\*args)

{

QApplication app(argv, args);

QTextEdit textEdit;

QLabel lbl("Hello");

textEdit.show();

lbl.show();

return app.exec();

}

**Сигналы и слоты**

Ключевым механизмом взаимодействия объектов в Qt являются сигналы и слоты. Сигналы и слоты стали возможны благодаря мета-объектной системе Qt.

При программировании GUI, когда изменяется один из виджетов, мы зачастую хотим, чтобы другие виджеты были об этом уведомлены. В общем случае, мы хотим, чтобы объекты могли взаимодействовать друг с другом. Например, если пользователь нажал кнопку Закрыть , мы вероятно захотим, чтобы объект window вызвал функцию close().

Другие инструменты разработки обеспечивают подобный функционал использованием callback(небольшое замечание! Технология функций обратного вызова очень старая. Лежит в основе, например, X Window System. Сугубо моё мнение, данный подход, в современных условиях, имеет больше минусов чем плюсов. Конечно же дальнейшее развитие привело к созданию специальные библиотеки классов языка C++, облегчающие программирование для ОС Windows. Одной из самых первых таких библиотек (и до сих пор находящихся в применении у целого ряда индивидуальных разработчиков и компаний) является Microsoft Foundation Classes (MFC). Минус её в том, что объектно-ориентированной её можно назвать с большой натяжкой. ). callback являет указателем на функцию, и если Вы хотите выполнить функцию, которая уведомит Вас о каком-либо событии, то Вы передаёте указатель на другую функцию, то есть callback. Работающая функция вызывает callback тогда, когда это уместно. В то время как существуют фреймворки, которые успешно используют callback функции, тем не менее callback являются неинтуитивным способом, который может вызывать проблемы с обеспечением корректности возвращаемых аргументов.

Механизм сигналов и слотов полностью замещает старую модель функций обратного вызова, он очень гибок и полностью объектно-ориентирован. Сигналы и слоты — это краеугольный концепт программирования с использованием Qt, позволяющий соединить вместе несвязанные друг с другом объекты. Каждый унаследованный от QObject класс способен отправлять и получать сигналы. Эта особенность идеально вписывается в концепцию объектной ориентации и не противоречит человеческому восприятию. Представьте себе ситуацию: у вас звонит телефон, и вы реагируете на это снятием трубки. На языке сигналов и слотов подобную ситуацию можно описать следующим образом: объект «телефон» выслал сигнал «звонок», на который объект «человек» отреагировал слотом «снятия трубки».

Каждый объект, интегрированный в систему управления Qt, т.е. описанный как Q\_OBJECT, может иметь типизированные слоты, обеспечивающие прием и обработку типизированных сигналов от других объектов, и собственные сигналы, прием которых могут осуществлять другие объекты. Связь между сигналами и слотами конкретных объектов устанавливается посредством функции connect(…) (см. рисунок 4).

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

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

Сигналы выпускаются объектом, когда его внутреннее состояние изменилось в определенном направлении, которое может быть интересно другим объектам. Сигналы являются публично доступными функциями и могут быть вызваны где угодно, но рекомендуется их вызывать только в классе, где они были определены, а также в его подклассах.

Слот вызывается тогда, когда сигнал подключенный к нему был вызван. Слоты являются нормальной С++ функцией и может быть вызвана; они особенны только тем, что к ним подключаются сигналы.

Также слоты могут выполняться как обычные функции, они подчиняются обычным правилам С++, когда вызываются непосредственно. Однако, как слоты, они могут быть вызваны другими компонентами, несмотря на их уровень доступа, через сигнал-слотовое подключение. Это означает, что сигнал испускается из одного из классов и может быть передан в приватный слот, который будет вызван из этого несвязанного класса.

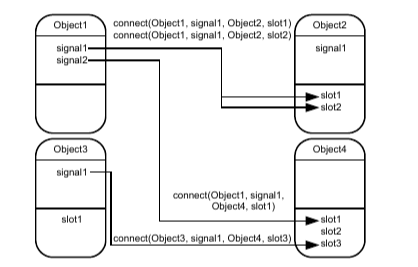


Рисунок 4-Схема связывания сигналов и слотов объектов

Следует отметить, что механизм слотов не исключает возможности использования

средств наследования и полиморфизма языка C++, так что любой класс Qt может быть

переопределен.

**Важное различие 5 версии Qt.**

До пятой версии Qt подключение сигнала к слоту записывалось посредством макросов, тогда как в пятой версии стала применяться запись, основанная на указателях.

*Запись с макросами:*

connect(button, SIGNAL(clicked()), this, SLOT(slotButton()));

*Запись на основе указателей:*

connect(button, &QPushButton::clicked, this, &MainWindow::slotButton);

Преимущество второго варианта заключается в том, что имеется возможность определить несоответствие сигнатур и неверное наименование слота или сигнала ещё на стадии компиляции проекта, а не в процессе тестирования приложения.

**Создание программы с полем и кнопкой выхода.**

**Замечание! В Qt в ряде случаев удобнее работать с указателями.**

Создадим кнопку и поле для ввода.(А кнопку ли я в примере создаю? Можете посмотреть документацию)

#include <QApplication>

#include <QTextEdit>

# include <QLabel>

int main(int argv, char \*\*args)

{

QApplication app(argv, args);

QTextEdit \*textEdit = new QTextEdit;

QLabel \*lbl = new QLabel;

QWidget \*window = new QWidget;

return app.exec();

}

Класс QWidget является фундаментальным для всех классов виджетов. Его интерфейс содержит 254 метода, 53 свойства и массу определений, необходимых каждому из виджетов, например, для изменения размеров, местоположения, обработки событий и др. Сам класс QWidget, как видно из рис. 5.1, унаследован от класса QObject, а значит, может использовать механизм сигналов/слотов и механизм объектной иерархии. Благодаря этому виджеты могут иметь потомков, которые отображаются внутри предка. Это очень важно, так как каждый виджет может служить контейнером для других виджетов, — то есть в Qt нет разделения между элементами управления и контейнерами. Виджеты в контейнерах могут выступать в роли контейнеров для других виджетов, и так до бесконечности. Например, диалоговое окно содержит кнопки Ok и Cancel (Отмена) — следовательно, оно является контейнером. Это удобно еще и потому, что если виджет-предок станет недоступным или невидимым, то виджеты-потомки автоматически примут его состояние.

Виджеты без предка называются виджетами верхнего уровня (top-level widgets) и имеют свое собственное окно. Все виджеты без исключения могут быть виджетами верхнего уровня. Позиция виджетов-потомков внутри виджета-предка может изменяться методом setGeometry ( ) вручную или автоматически, с помощью специальных классов компоновки (layouts). Для отображения виджета на экране вызывается метод show(), а для скрытия — метод hide ( ).

**Как отобразить две кнопки(два виджета) в одном окне?** Мы будем использовать классы компоновки виджетов (Layouts), Конечно, можно вручную размещать виджеты(аналогия с подсчётами координат в WINapi) в окнах приложения, но это существенно усложняет разработку. Qt предоставляет так называемые менеджеры компоновки, позволяющие организовать размещение виджетов на поверхности другого виджета. Основу их работы определяет возможность каждого виджета сообщать о том, сколько ему необходимо места, может ли он быть растянут по вертикали и/или горизонтали и т. п.

Менеджеры компоновки предоставляют возможности для горизонтального, вертикального и табличного размещения не только виджетов, но и встроенных компоновок. Это позволяет конструировать довольно сложные размещения.

Фундаментом для всей группы менеджеров компоновки является класс QLayout— абстрактный класс, унаследованный сразу от двух классов: QObject и QLayoutitem (рис. 5). Этот класс определен в заголовочном файле QLayout.

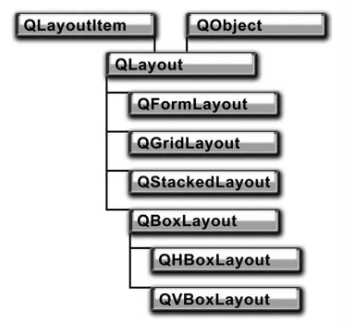


Рис.5 Иерархия классов менеджеров компоновки

От класса QLayout унаследованы классы QGridLayout И QBoxLayout (см. рис. 6). Класс QGridLayout управляет табличным размещением, а от QBoxLayout унаследованы два класса QHBoxLayout и QVBoxLayout для горизонтального и вертикального размещения.

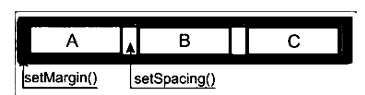


Рис.6

По умолчанию между виджетами остается небольшое расстояние. Это расстояние необходимо для их визуального разделения. Задать его можно с помощью метода setSpacing(), передав в него нужное значение в пикселах. Методом setMargin ( ) можно установить отступ виджетов от границы компоновки — обычно типичными значениями могут быть 5 или 10 пикселов. Рисунок 6.2 иллюстрирует смысл этих методов на примере горизонтального размещения.

При помощи метода addwidget ( ) выполняется добавление виджетов в компоновку, а с помощью метода addLayout ( ) можно добавлять встроенные менеджеры компоновки. Если понадобится удалить какой-либо виджет из компоновки, то следует воспользоваться методом removewidget() , передав ему указатель на этот виджет.

Объекты размещений отвечают за правильное размещение виджетов и присвоение им нужных виджетов-предков. То есть, вам не нужно беспокоиться о том, чтобы присваивать объекты предков, так как это будет сделано за вас автоматически.

Организуем, к примеру, горизонтальное размещение, воспользовавшись классом QHBoxLayout.