**Лекция №2**

**Соединение объектов**  
 Соединение объектов осуществляется при помощи статического метода connect( ), который определен в классе QObject. В общем виде вызов метода connect () выглядит следующим образом:

QObject::connect (const QObject\* sender,  
const char\* signal,  
const QObject\* receiver,  
const char\* slot,  
Qt::ConnectionType type = Qt: Autoconnection  
);

Ему передаются пять следующих параметров:  
♦ sender — указатель на объект, отправляющий сигнал;  
♦ signal — это сигнал, с которым осуществляется соединение. Прототип (имя и аргументы) метода сигнала должен быть заключен в специальный макрос SIGNAL(method( ) );

receiver — указатель на объект, который имеет слот для обработки сигнала;

♦ slot — слот, который вызывается при получении сигнала. Прототип слота должен быть

заключен в специальный макрос SLOT(method ( ) );

♦ type — управляет режимом обработки. Имеется три возможных значения:

*Qt::DirectConnection* — сигнал обрабатывается сразу вызовом соответствующего метода слота, *Qt::QueuedConnection* — сигнал преобразуется в событие (см. главу 14) и ставится в общую очередь для обработки, *Qt: Autoconnection* — это автоматический режим, который действует следующим образом: если отсылающий сигнал объект находится в одном потоке с принимающим его объектом, то устанавливается режим

Qt::DirectConnection, В противном случае — режим Qt::QueuedConnection. Этот режим

(Qt: Autoconnection)определен в методе connection () по умолчанию. Вам вряд ли придется изменять режимы «вручную», но полезно знать, что такая возможность есть.

Существует альтернативный вариант метода connect( ), преимущество которого заключается в том, что все ошибки соединения сигналов со слотами выявляются на этапе компиляции программы, а не при ее исполнении, как это происходит в классическом варианте метода connect(). Прототип альтернативного метода выглядит так:

QObject::connect (const QObject\* sender,

const QMetaMethods signal,

const QObject\* receiver,

const QMetaMethods slot,

Qt::ConnectionType type = Qt: Autoconnection

);

Параметры этого метода полностью аналогичны предыдущему за исключением тех параметров, которые были объявлены в предыдущем методе как const char\*. Вместо них используются указатели на методы сигналов и слотов классов напрямую. Благодаря именно этому, если вы вдруг ошибетесь с названием сигнала или слота, ваша ошибка будет выявлена сразу в процессе компиляции программы. К недостаткам альтернативного метода можно отнести то, что при каждом соединении нужно явно указывать имена классов для сигнала и слота и следить за совпадением их параметров.

**Кириллица в Qt.**

Я уже говорил о том, что нам важно какая версия Qt установлена. Также я отметил, что текущая версия Qt 5.\*. С отображением кириллицы на формах, кнопках, в консоли и т.д., есть ряд небольших трудностей. Данные проблемы могут немного затруднять кроссплатформенность приложений. До 5 версии данная проблема решалась несколькими способами.

Текст ниже не надо воспринимать как руководство к действию. Никогда. Совсем никогда.

**Первый вариант** -- использовать функцию QString::fromLocal8Bit. Дело в том, что для работы со строковыми значениями в Qt имеется класс QString, который оперирует двухбайтными символами Unicode. Для преобразования обычных строк с однобайтными символами к типу QString как раз и предназначен метод fromLocal8Bit. При этом используется системная кодировка (в Windows --- cp1251, в Linux -- обычно koi8-r).

Недостаток данного варианта заключается в том, что если программа, например, разрабатывалась в системе Windows, а затем мы захотели скомпилировать её исходные тексты в Linux, где используется другая кодовая таблица для символов кириллицы, то предварительно придётся осуществить перекодировку исходных текстов, иначе после компиляции все русские сообщения окажутся нечитаемыми.

**Второй вариант** работы с символами кириллицы -- явное использование кодеков (специальных объектов для перекодировки строк). Соответствующий класс QTextCodec определён в заголовочном файле с тем же именем (без расширения! Заголовочные файлы с расширением .h используются только в старых проектах Qt3). При создании кодека надо указать название используемой кодовой таблицы, например:

QTextCodec \*codec = QTextCodec::codecForName("CP1251");

Затем надо связать этот кодек со всеми строками C++:

QTextCodec::setCodecForCStrings(codec);

Поскольку при создании кодека указана конкретная кодовая таблица, то исход компиляции исходных текстов не зависит от системной кодировки той платформы, на которой производится сборка программы: результат будет везде одним и тем же. Но файлы с исходными текстами программ в этом случае нельзя подвергать перекодировке (или после перекодировки требуется изменить название кодовой таблицы, заданной при создании кодека).

Третий (наиболее предпочтительный) метод работы с символами национальных алфавитов связан с использованием специальной функции перевода tr, с помощью которой осуществляется интернационализация приложений. Подробнее этот вопрос мы обсудим позже, а пока договоримся все строковые константы, указанные в тексте программы, передавать в качестве параметра функции tr. Эта статическая функция является членом всех классов Qt, порождённых от базового класса QObject, но если, как сейчас, мы собираемся вызвать её в главной программе, а не в каком-либо методе класса, то приходится указывать какой-нибудь подходящий объект, например, QObject::tr. Для указания кодировки, используемой функцией перевода, надо создать соответствующий кодек и передать его в качестве аргумента методу setCodecForTr. Окончательный вариант нашей программы показан в листинге 4.

**Внимание!!!!! В современных технологиях уже повсеместно используется кодировка Utf-8. Зачем она была придумана и как она решает проблемы, которые были раньше почитайте на сторонних источниках.**

Для Qt5 важно, чтобы все исходники были в Utf-8, тогда никаких проблем возникнуть особо не должно. Например, код

QPushButton \*button = new QPushButton("&Запуск");

button->show();

Позволит отобразить на кнопке текст без «крякозябр».

Но как желательно всё-таки создавать интерфейсы? И как это делается правильно и в «промышленности» Если приложение планируется для распространения и использования во многих странах Мира, то желателен следующий подход:

1. все надписи только на английском.
2. Далее, все строки, которые увидит пользователь должны быть обработаны функциями QObject::tr() или QcoreApplication::translate(). Например,

#include <QtGui>

#include <QtCore>

int main(int argc, char \*argv[]) {

 QApplication app(argc, argv);

QLabel label(app.translate("QLabel", "Hello, World!"));

 label.show();

return app.exec();

}

1. Далее, используя специальную программу Qt\_Linguist создать перевод на русский и другие языки.
2. Загружать перевод из файла или внутренних ресурсов при старте программы.

Подробно процесс интернационализации приложений описан в <http://doc.qt.io/qt-5/internationalization.html> . Можете почитать.

Таким образом, если при реализации приложения Вам важна поддержка национального языка, то нужно обязательно следить за кодировкой исходных файлов, поэтому, крайне желательно, разработку вести в Qt Creator.

**Обработка нажатия левой кнопкой мыши на кнопку выход.**

Создадим программу, которая будет содержать текстовое поле и кнопку, при нажатии на которую левой кнопкой мыши программа будет просто закрываться.

Итак мы имеем код, который размещает в автоматическом режиме виджет «текстовое поле» и виджет «кнопка» в окне.

#include <QApplication>

#include <QPushButton>

#include <QTextEdit>

# include <QHBoxLayout>

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

{

QApplication app(argc, argv);

QPushButton \*button = new QPushButton("&Выход");

QTextEdit \*textEdit = new QTextEdit;

QHBoxLayout \*layout = new QHBoxLayout;

QWidget \*window = new QWidget;

layout-> addWidget(button);

layout->addWidget(textEdit);

window->setLayout(layout);

window->show();

return app.exec();

}

Напоминаю про то, что механизм наследования позволяет сократить код в несколько раз. Также класс QApplication напрямую унаследован от QGuiApplication и конечно же он дополнен своими возможностями. Механизм наследования в с++, как вы знаете, допускает так делать. Именно этот класс устанавливает стиль приложения. Можно, кстати, делать и свои стили.

Итак, нам нужно, чтобы при нажатии на кнопку приложение закрывалось. Сигнал будет «исходить» от кнопки(будет источником сигнала), если будет нажатие(сигналом в данном случае будет нажатие), слот будет получать данный сигнал и реагировать на него. Что здесь будет слотом? Нам нужно завершить приложение. Мы имеем объект app. Нам необходимо использовать функцию- обработчик quit(). Таким образом, следующий код установить связь между нажатием кнопки и закрытием приложения.

QObject::connect(button,SIGNAL(clicked(bool)),&app, SLOT(quit()));

Я уже говорил, что фактически, класс QObject — основной, базовый класс. Подавляющее большинство классов Qt являются его наследниками. Классы, имеющие сигналы и слоты, должны быть унаследованы от этого класса.

Общий код, который позволяет решить задачу выглядит следующим образом.

#include <QApplication>

#include <QPushButton>

#include <QTextEdit>

# include <QHBoxLayout>

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

{

QApplication app(argc, argv);

QPushButton \*button = new QPushButton("&Выход");

QTextEdit \*textEdit = new QTextEdit;

QHBoxLayout \*layout = new QHBoxLayout;

QWidget \*window = new QWidget;

layout-> addWidget(button);

layout->addWidget(textEdit);

window->setLayout(layout);

window->show();

QObject::connect(button,SIGNAL(clicked(bool)),&app, SLOT(quit()));

return app.exec();

}

Важно. Символ "&" перед буквой "В" в тексте надписи на кнопке позволяет активировать её не только по щелчку левой кнопкой мыши или нажатием клавиши Enter, но также с помощью комбинации клавиш Alt+в (к сожалению, только в режиме ввода кириллицы).

**Небольшое резюме.**

**Немного про «ручное» размещение виджетов**

С помощью метода setGeometry(int x, int y, int w, int h) или setGeometry(const QRect&) можно задать положение и размер любого визуального элемента в пикселах. Для установки размеров без изменения положения может использоваться метод resize(int w, int h) или resize(const QSize&). Наоборот, для перемещения элемента в нужную позицию с сохранением прежних размеров служит метод move(int x, int y) или move(const QPoint&).

Недостатком жёсткого варианта размещения элементов интерфейса является то, что пользователь не может изменить размер окна диалога (или изменение размеров окна не влияет на взаимное положение и размеры всех его элементов). В результате при низком разрешении монитора всё выглядит слишком крупно, а то и вовсе не помещается на экран, при высоком -- наоборот, слишком мелко. Кроме того, в различных операционных системах используются разные шрифты, поэтому надписи и поля ввода, прекрасно смотревшиеся в одной системе, при переносе на другую платформу могут не уместиться в прежних границах. К тому же в солидных программных продуктах принято давать пользователю возможность настраивать интерфейс программы по своему вкусу, в частности, изменять гарнитуру и размер шрифта. А при локализации (переводе интерфейса программы на другой язык) всё ещё больше усложняется.

На следующей лекции подробнее рассмотрим размещение элементов и напишем программу.