Урок 2

1. Слоты

Сигналы и слоты используются для коммуникации между объектами. Механизм сигналов и слотов – главная особенность Qt и, вероятно, та часть, которая отличается от особенностей, предоставляемых другими фреймворками.

Механизм сигналов и слотов является расширением языка программирования С++ в Qt, который используется для установления связи между объектами. Если происходит какое-либо определенное событие, то при этом может генерироваться сигнал. Данный сигнал попадает в связанный с ним слот. В свою очередь, слот – это обычный метод в языке C++, который присоединяется к сигналу; он вызывается тогда, когда генерируется связанный с ним сигнал. Ничего сложного здесь нет.

Все графические приложения управляются событиями: всё, что происходит в приложении является результатом обработки тех или иных событий. Они являются важной частью любой графической программы. В большинстве случаев события генерируются пользователем приложения, но они также могут быть сгенерированы и другими средствами, например, подключением к интернету, оконным менеджером или таймером. При разработке программ в Qt задумываться о событиях приходится довольно редко, поскольку виджеты генерируют сигналы, когда происходит нечто значительное. Сами же события приобретают значение в том случае, когда необходимо создать, например, новый виджет или расширить функционал существующего.

Сигналы и слоты вместе составляют мощный механизм обработки пользовательских действий, но стоит заметить, что пользоваться ими можно только в классах унаследованных от QObject.

Для знакомства со слотами и программным созданием компонентов разберем простой пример с кнопкой, закрывающей окно при нажатии (код добавлен в “чистый проект”).



Следует отметить подключенные классы в строках 3-6. Это позволяет создавать экземпляры программно и дальше использовать их для реализации определенной логики приложения. В строке 14 создается экземпляр класса QHBoxLayout и далее дополняются его параметры отступом. Этот слой (Layout) создается не просто так. В пустом приложении существует только CentralWidget, а с ним удобнее взаимодействовать через метод setLayout(), который, к сожалению, не сможет принять как аргумент кнопку. Простыми словами, дочерний элемент получится добавить только через переходник в виде слоя. В 17 строке создается экземпляр класса QPushButton – кнопка, к которой и добавим слушатель на событие. В 19-20 строке происходит вставка элементов друг в друга. В методе addWidget() можно наблюдать дополнительные параметры, которые отвечают только за расположение элемента на слое.

Поподробнее следует остановиться на строке 22. В ней используется неотъемлемый для реализации примера метод – connect. Он соединяет сигнал со слотом. При нажатии на кнопку Quit генерируется сигнал щелчка кнопки мыши. qApp – это глобальный указатель на объект приложения. Он определяется в заголовочном файле QApplication. Метод quit() вызывается при появлении сигнала щелчка мышкой.

Существует еще несколько заготовленных методов (как quit), которые не нужно реализовывать самостоятельно. К ним относятся:

* exit() – закрытие кона приложения;
* aboutQt() – открытие окна со справкой по Qt;
* closeAllWindows() – закрытие всех окон приложения.

Слоты можно создавать и самостоятельно. Реализуем пример со сменой цвета кнопки при ее нажатии. Для этого объявим поля класса MainWindow, хранящие цвет и кнопку. Так будет удобнее получать доступ к кнопке в слоте. Объявим метод changeButtonColor() в специальной секции для слотов public slots (стоит отметить, что она может быть и приватной, и защищенной, в зависимости от ваших нужд).

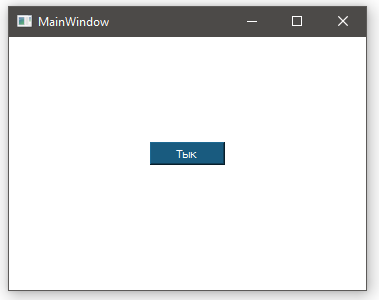
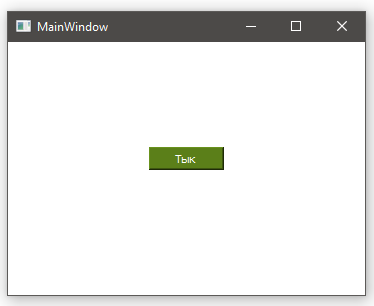
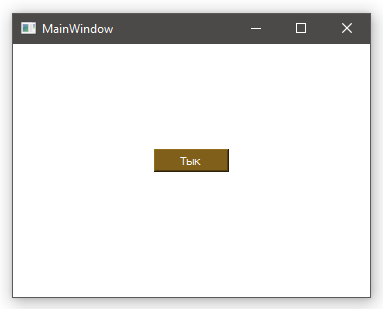


Далее в файле mainwindow.cpp реализуется инициализация переменных и слот для смены цвета кнопки.



Подробнее остановимся на строке 20. Объект qApp был заменен на this. Такая манипуляция объясняется тем, что слот changeButtonColor() может влиять только на его владельца, поэтому вместо экземпляра приложения как аргумент используем экземпляр класса MainWindow, ссылка на который и хранится в специальном слове this.

Приложение будет работать следующим образом:



1. Сигналы

Если можно создавать собственные слоты, то легко сделать вывод, что и сигналы тоже можно реализовать самостоятельно. Реализуем собственный сигнал, который будет менять цвет нашей кнопки каждые 10 миллисекунд.

Для решения задачи воспользуемся классом QTimer. Он унаследован от QObject, а значит умеет работать с сигналами и слотами. Предоставляет регулярные таймеры, которые по истечении заданного промежутка времени посылают сигнал timeout(). При этом класс имеет статический метод singleShot(), он посылает сигнал через заданный промежуток времени лишь один раз.

Кроме того, QTimer включает в себя:

Public-функции:

* int interval() – возвращает числовое(int) значение установленного интервала таймера в миллисекундах
* int remainingTime() – возвращает оставшееся время в миллисекундах
* bool isActive() – возвращает логическое(bool) значение true, если таймер запущен
* bool isSingleShot() – возвращает true, если таймер включен для срабатывания только один раз
* void setInterval(int msec) – позволяет установить интервал таймера, где msec – время в миллисекундах
* void setTimerType(Qt::TimerType atype) – устанавливает тип таймера;
* int timerId() – возвращает id таймера;
* Qt::TimerType timerType() – возвращает тип таймера.

Слоты:

* start() – запускает таймер;
* start(int msec) – через msec миллисекунд запускает таймер, если таймер не запущен, в противном случае таймер останавливается и запускается снова;
* stop() – останавливает таймер.

Сигнал:

* timeout() – посылается во время срабатывания таймера.

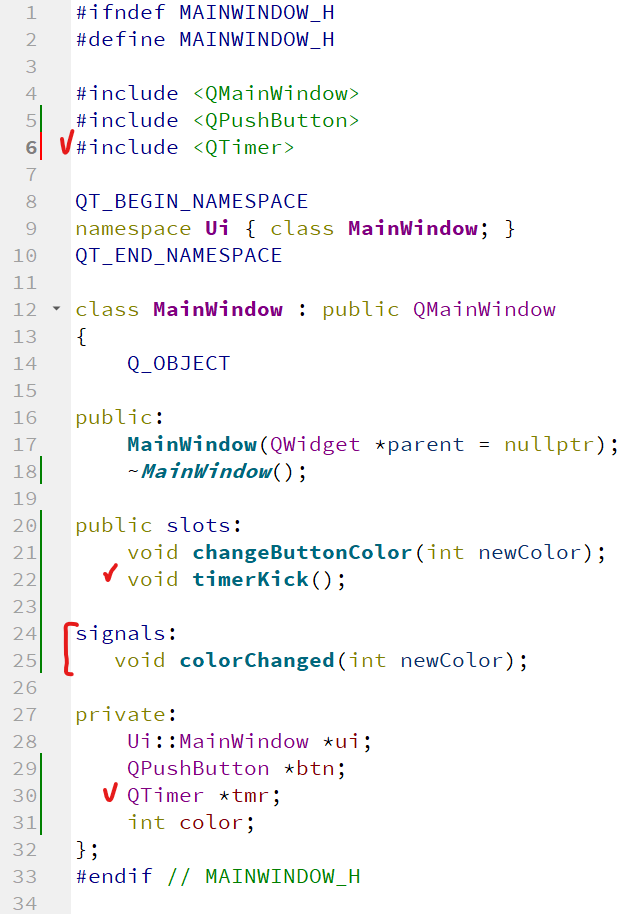
Приступим к реализации задачи. Только собственным сигналом обойтись не получится. Сигнал должен быть чем-то вызван. В данном примере понадобятся два сигнала:

* timeout() – существующий сигнал, который будет инициатором всех изменений в интерфейсе;
* colorChanged() – разработанный нами сигнал, который будет использован для смены цвета

и два слота:

* timerKick() – слот, реагирующий на timeout() и отправляющий сигнал о смене цвета;
* changeButtonColor() – наш старый слот, который в этот раз будет отвечать только за отображение состояния в интерфейсе.

Заголовочный файл mainwindow.h будет выглядеть следующим образом:



Отметим строки 24-25. Сигналы в Qt также, как и слоты, добавляются в специальную секцию. Для сигналов эта секция имеет название signals. Далее объявляется метод colorChanged(). Если сравнить его с методом changeButtonColor(), то станет заметным совпадение аргументов. Это сделано не просто так, сигнал с аргументами является неким проводником данных для слота. Простыми словами, аргументы, с которыми вызовется сигнал, передадутся в слот.

Теперь реализуем методы в файле mainwindow.cpp:

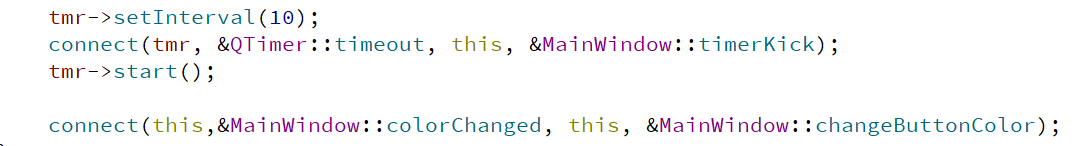


В 11 строке инициализируется таймер, а в 19-21 строках устанавливается его интервал. Далее соединяется сигнал таймера со слотом timerKick() и запускается программа. В 23 строке установлена связь между сигналом изменения цвета и слотом, меняющим цвет кнопки.

В коде следует обратить внимание на три аспекта:

* В строке 30 используется директива emit, которая и вызывает сигнал colorChanged (int newColor).
* В коде отсутствует тело метода colorChanged (int newColor). Это не ошибка. Сигналы используется только как переходники для вызова слотов, поэтому им не нужно тело.
* Передача слотов и сигналов произведена через макросы SLOT и SIGNAL. Они преобразуют свои аргументы в строки. Такой вариант является альтернативой передаче по ссылке, но со своими ограничениями. Соединяемые сигналы и слоты должны иметь идентичные сигнатуры. Исключение составляет случай, когда сигнал имеет большее число аргументов, чем слот. В этом случае "лишние" аргументы просто не передаются в слот.

Вариант кода со старым подключением:



Следует отметить, что подключение одного сигнала к одному слоту – не единственный вариант. Существует следующие возможности:

* Подключение одного сигнала к другому сигналу.
* Подключение одного слота к нескольким сигналам.
* Подключение нескольких слотов к одному сигналу.

Иногда при работе приложения создавать соединения и отключать их приходится из кода. Доработаем программу таким образом, чтобы при нажатии кнопки включался или отключался процесс изменения ее цвета.

Добавим слот toogleProcess(), который будет отвечать за запуск и остановку таймера:



1. Метаобъектная система в Qt

Раз речь зашла о глобальных макросах и директивах, то пора разобраться с тем, что они из себя представляют.

Одно из самых значительных достижений Qt – это расширение возможностей языка C++ механизмом создания независимых компонентов, которые могут взаимодействовать между собой, не имея информации друг о друге. Этот механизм получил название Meta Object System и предоставляет два ключевых сервиса: сигналы-слоты и интроспекцию. Интроспекция позволяет получать метаинформацию о потомках класса QObject во время исполнения, включая список поддерживаемых сигналов, слотов и имя класса объекта. Этот механизм также реализует поддержку свойств объектов (используются в Qt Designer) и перевод текста (для нужд интернационализации).

Стандарт C++ не обеспечивает возможность получения динамической метаинформации, которая необходима метаобъектной системе Qt. Поэтому данная проблема была решена созданием дополнительного инструмента moc (метаобъектный компилятор). Он собирает всю необходимую информацию из классов Q\_OBJECT и делает ее доступной через вызовы обычных функций языка C++, что позволяет метаобъектной системе работать с любым компилятором C++.

Механизм работает следующим образом:

* Макрос Q\_OBJECT объявляет ряд функций, который должны присутствовать в каждом потомке QObject: metaObject(), className(), tr() и ряд других.
* Утилита moc генерирует реализацию сигналов и функций, объявленных макросом Q\_OBJECT.
* Эти функции используются методами connect() и disconnect().

Все действия выполняются автоматически утилитами qmake и moc, так что приходится довольно редко вспоминать об этом.

До сих пор в примерах сигналы и слоты использовались исключительно с виджетами. Однако этот механизм реализован непосредственно в классе QObject, и область его применения не ограничивается графическим интерфейсом. Он может использоваться любым классом – наследником QObject:



Обратите внимание на реализацию слота setSalary(). Сигнал salaryChanged() посылается только в том случае, когда newSalary != mySalary. Такой способ предотвращает попадание в бесконечный цикл при наличии обратной связи с другим объектом.