Урок 5

Qt Quick

1. Введение

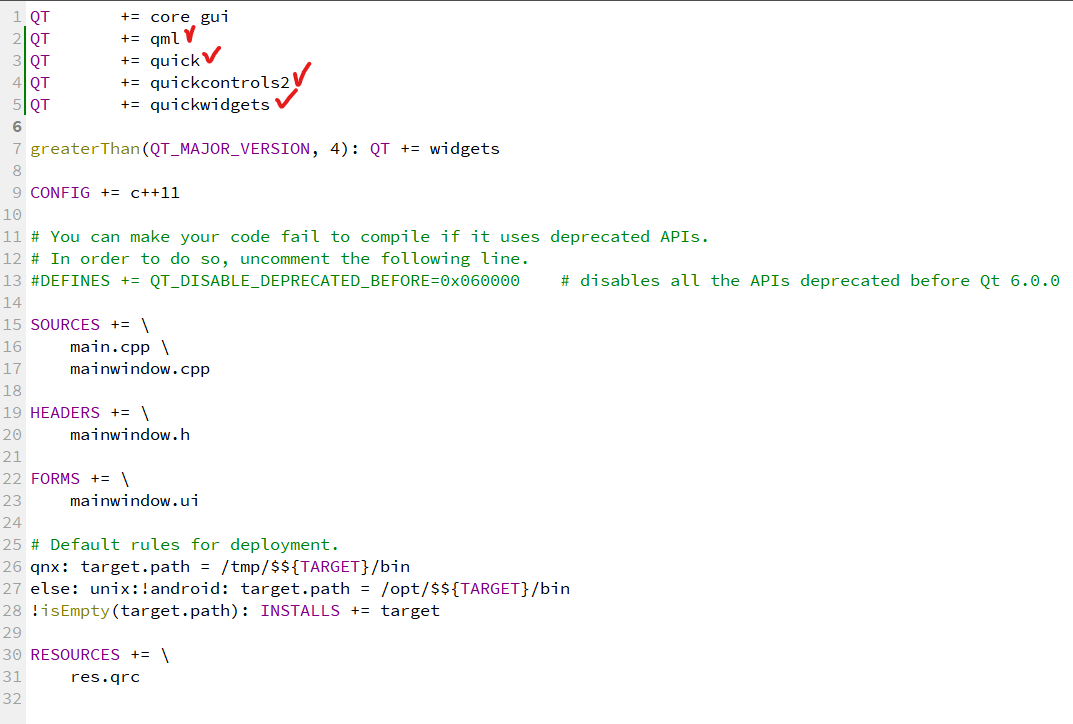
Фреймворк, разработанный и поддерживаемый проектом Qt в рамках Qt framework, обеспечивает способ создания пользовательских высокодинамичных графических пользовательских интерфейсов с плавными переходами и эффектами, которые становятся все более распространенными, особенно в мобильных устройствах. Qt Quick включает декларативный язык сценариев, называемый QML.

Возможностями фреймворка можно успешно пользоваться, начиная с Qt 4.7.1. После версии 5.1 разработчики смогли разрабатывать не только статичный qml интерфейс, но и шаблонно внедрять виджеты, компоненты и элементы взаимодействия в свои приложения. Под термином шаблонного внедрения подразумевается использование одного qml файла для создания нескольких однотипных элементов.

Тема использования декларативных qml интерфейсов достаточно обширна и поэтому урок нацелен на простое знакомство с возможностями Qt Quick.

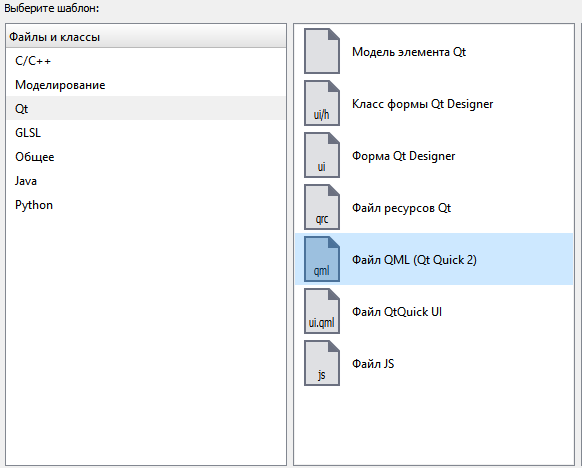
1. Подключение

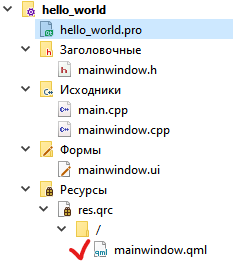
Для внедрения возможностей Qt Quick в проект следует подключить его в файле конфигурации:



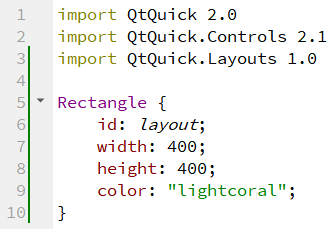
В 4 и 5 строках подключены конкретные элементы, которые будут использованы в уроке: виджеты и элементы взаимодействия.

Осталось создать файл .qml через контекстное меню древа проекта. Стоит обратить внимание, что в уроке акцент сделан на работу через текстовый редактор, но также qml можно изменять и через специальную форму Qt Designer.



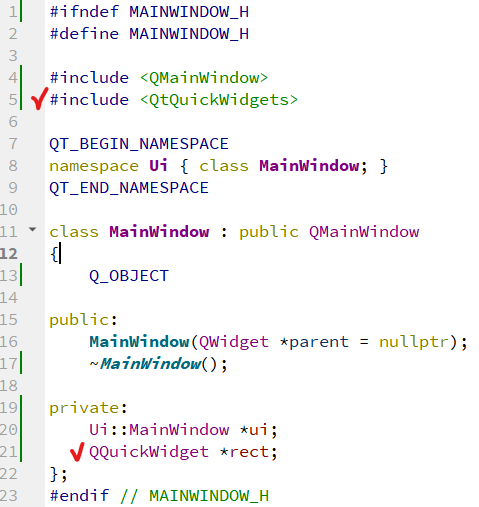


Создание элементов сводится к перечислению их свойств в блочном стиле. Например, для создания простого прямоугольника будет достаточно создать блок Rectangle и описать его свойства:



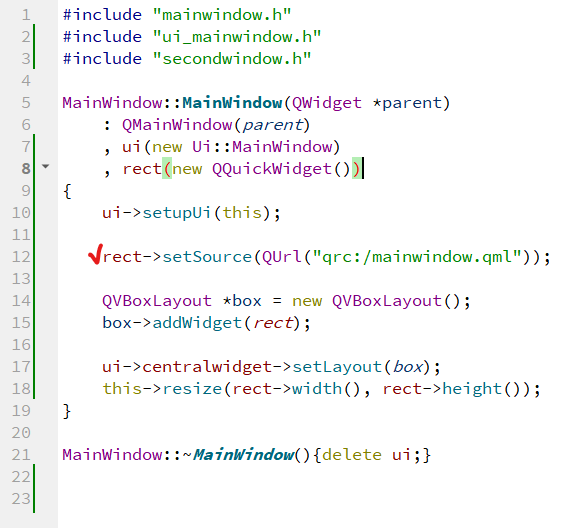
В 1 строке подключается QtQuick, а ниже – элементы взаимодействия с пользователем и слои, которые пригодятся в дальнейшем. Такое разбиение на пакеты позволяет подключать к программе только нужные компоненты, а не весь набор целиком.

Для использования qml элементов достаточно немного изменить файлы mainwindow.h и mainwindow.cpp:

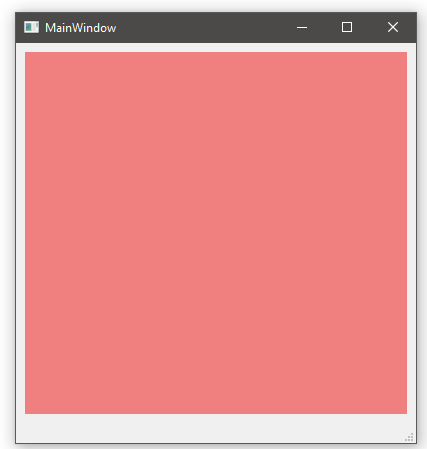


В класс добавлено новое поле типа QQuickWidget, которое будет хранить элемент, сгенерированный по qml файлу. Стоит отметить, что существует альтернативная возможность изменить тип переменной ui и использовать ее как компонент.

Для генерации элемента используется метод setSource():



Запуск программы приведет к созданию окна с прямоугольником, описанным в файле qml:



1. Динамическое создание элементов по шаблону qml

Разработчики стараются описать идентичные объекты в виде абстракций не только при написании кода, но и при их визуализации. Отображение списка элементов – часто встречающийся кейс при создании приложений. Например, контакты в телефонной книге отображаются по определенному шаблону.

Упрощенный вариант класса контакта:

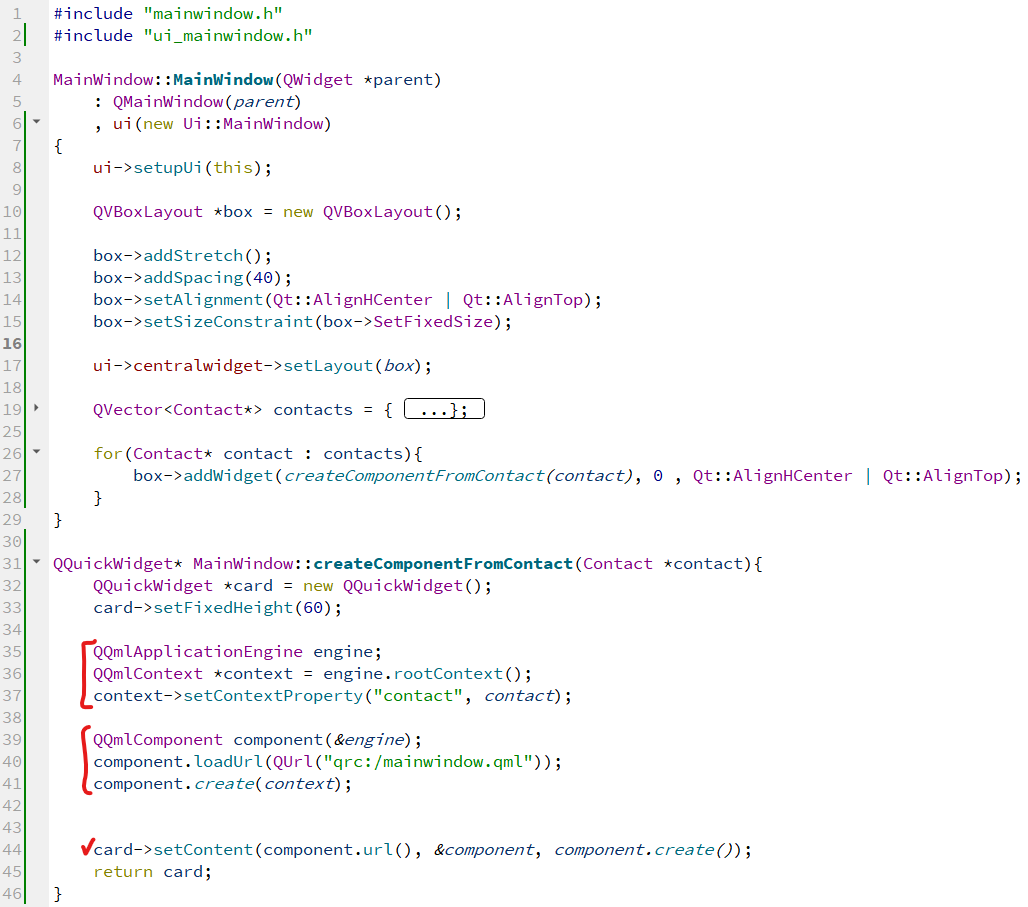


Макрос Q\_PROPERTY необходим для того, чтобы программист имел возможность создавать класс со свойствами, поддерживаемыми метаобъектной подсистемой Qt. Простыми словами – для обозначения свойств и методов взаимодействия с ними другими фреймворками. Таким образом разработчик обозначает, какие свойства следует подготавливать для подстановки их в шаблон qml. Для отображения контакта подойдет следующий код:



Объект Rectangle используется для создания контейнера с определенным фоном. RawLayout – специальный слой, позволяющий расположить элементы в одну строку. Компонент Text представляет собой текстовое поле. Стоит отметить, что свойств, отвечающих за отображение того или иного компонента, достаточно много, и выбор их сводится к вкусовым предпочтениям разработчика. Поэтому описывать их все в рамках учебного курса не имеет смысла. Подробнее о компонентах и свойствах можно прочитать в [документации](https://doc.qt.io/qt-6/qtquick-qmlmodule.html).

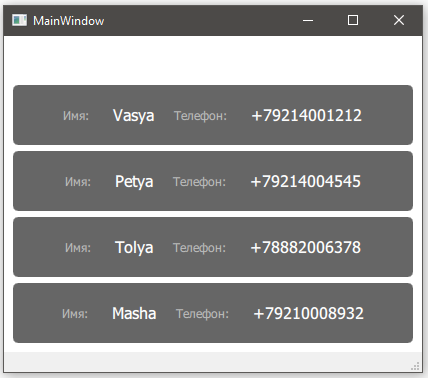
Осталось описать метод создания одной карточки с контактом в классе MainWindow и реализовать создание списка карточек из массива контактов:



В логике создания виджета по шаблону участвует три класса:

* QQmlApplicationEngine – класс, позволяющий обращаться к QML контексту приложения.
* QQmlContext – QML контекст, в который можно сохранять связи qml-шаблонов и экземпляров классов.
* QQmlComponent – компонент интерфейса, сопряженный с предыдущими двумя классами, и позволяющий преобразовать данные в QQuickWidget.

При запуске приложения появится окно с карточками контактов:



1. Использование метаобъектов qml

Если стоит задача управлять состоянием объектов, изображенных на экране, то подход переиспользования qml шаблонов будет не актуален. Существует два способа соединения qml объектов с классами. Если начальное состояние экземпляра всегда одинаковое, то следует пользоваться методом qmlRegisterType(), который позволяет зарегистрировать класс как qml компонент и дальше использовать его слоты и сигналы внутри qml файла. Как альтернатива регистрации компонента, существует возможность прокидывать данные в виде уже существующего экземпляра класса. Важный аспект этого способа в том, что названия свойств, которые используются для получения данных из контекста, должны быть известны и прописаны в qml файле заранее.

В примере будет разобран второй способ. Для работы с состоянием контакта из предыдущего примера следует доработать класс следующим образом:

**

Создание соединений между классами и их отображениями Qt производит самостоятельно. Поэтому существует множество правил и ограничений, без которых функционал не будет работать. Для получения результата следует учитывать следующие аспекты:

* Поля класса, которые будут использованы для передачи данных, должны начинаться с префикса “m\_” и быть настроены с помощью Q\_PROPERTY.
* Методы для обновления данных должны являться слотами и подкрепляться макросами Q\_INVOKABLE.
* Для полей класса должны существовать соответствующие сигналы обновления, имеющие один аргумент, совпадающий по типу с полем.
* Для поддержки всех типов инъектирования, к которым может прибегнуть Qt, следует реализовать 3 конструктора.

При реализации методов также следует поддерживать требуемый стиль написания:



Важно при использовании слотов вызывать соответствующие сигналы. Иногда при инъектировании данных методы могут быть вызваны несколько раз, поэтому требуется проверять наличие обновлений перед выполнением логики. Таким образом получится избежать зацикливания.

Если учесть все правила и ограничения во время написания программы, то появится возможность изменять состояние интерфейса динамически, не прибегая к обращению к виджетам:



В строке 33 достаточно вызвать метод для изменения состояния класса, а интерфейс изменится самостоятельно.