

ООП

Семестр 2. Лекция 8.

Многооконные приложения

Языки описания пользовательского интерфейса

Кафедра ИВТ и ПМ
ЗабГУ

2018

План

Прошлые темы

Многооконные приложения

Язык описания UI
QML

Ссылки

Outline

Прошлые темы

Многооконные приложения

Язык описания UI
QML

Ссылки

- ▶ Что такое бизнес-логика?
- ▶ Что такое шаблон проектирования
Модель-Представление-Вид?
- ▶ Что такое сигнал?
- ▶ Что такое слот?
- ▶ Какие классы могут соержжать сигналы и слоты?
- ▶ Что такое UI?
- ▶ Что такое API?

Outline

Прошлые темы

Многооконные приложения

Язык описания UI
QML

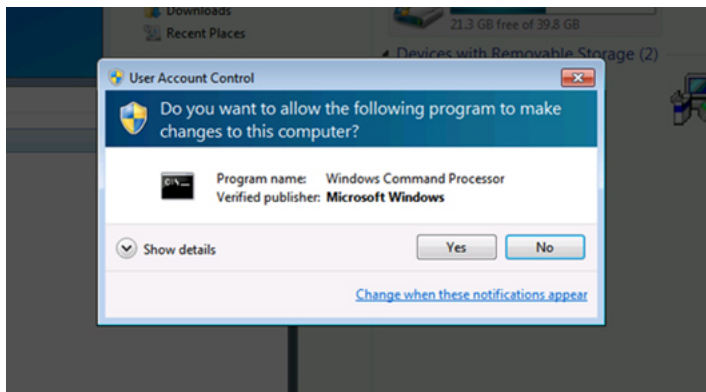
Ссылки

Однодокументный интерфейс (Single document interface, SDI) — способ организации графического интерфейса приложений в отдельных окнах. Не существует «фонового» или «родительского» окна, содержащего меню или панели инструментов, по отношению к активному — каждое окно несёт в себе эти элементы.

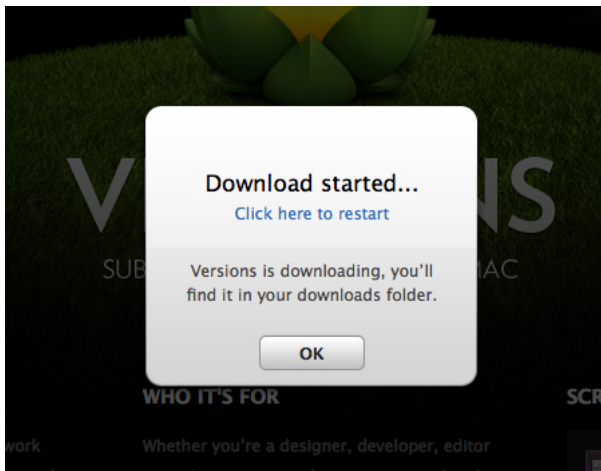
Модальное окно

Модальное окно в графическом интерфейсе пользователя — окно, которое блокирует работу пользователя с родительским приложением до тех пор, пока пользователь это окно не закроет.

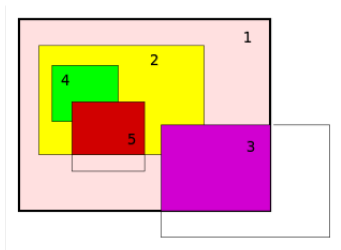
Модальные окна



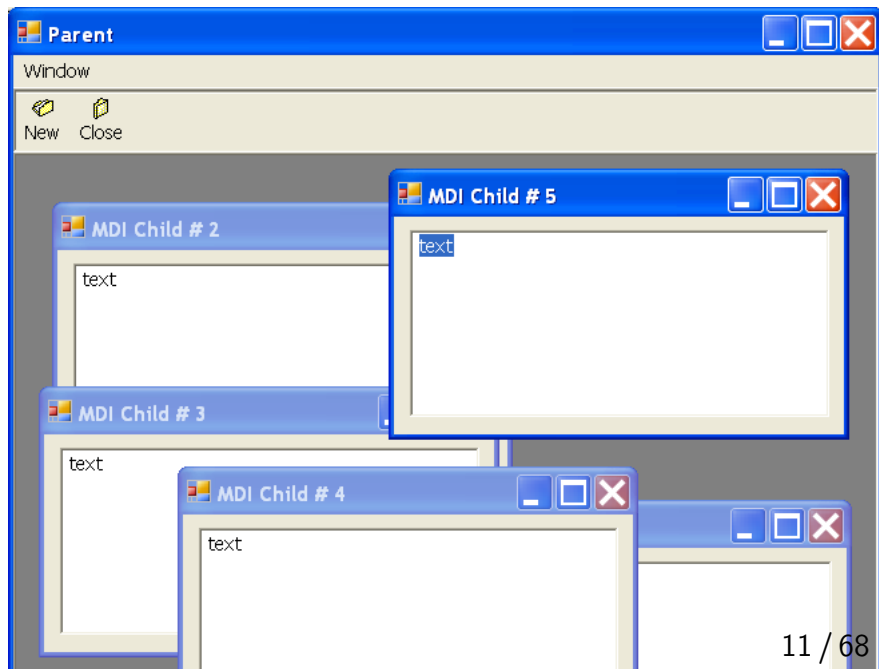
Модальные окна

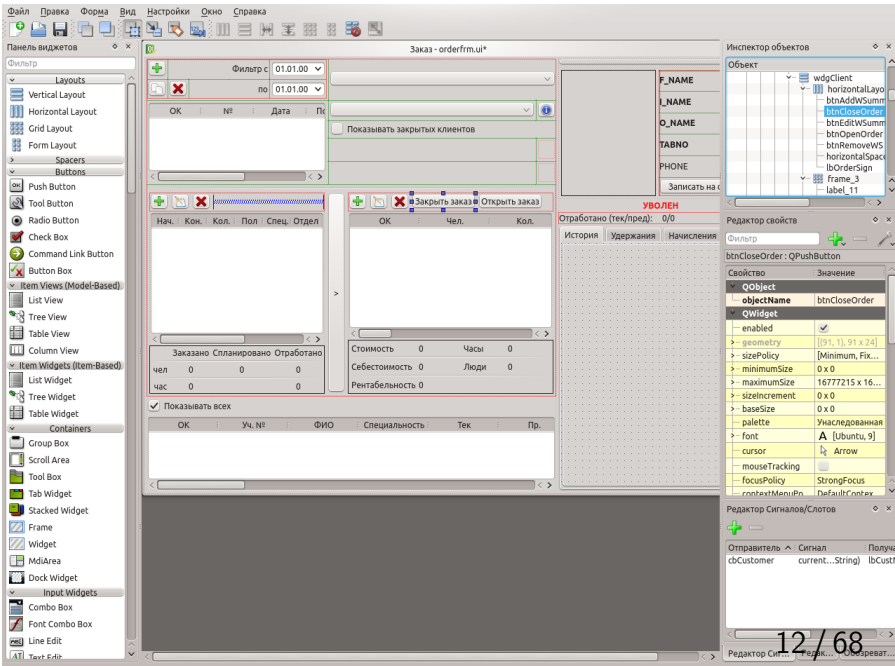


Многодокументный интерфейс (multiple document interface, **MDI**) — способ организации графического интерфейса пользователя, предполагающий использование оконного интерфейса, в котором большинство окон расположены внутри одного общего окна.



MDI



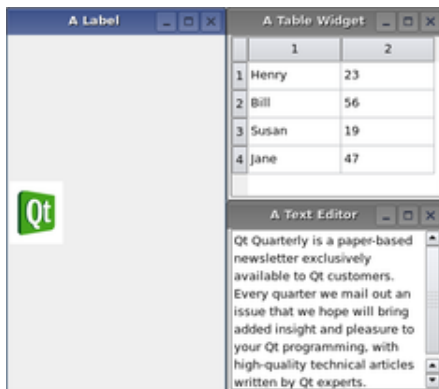


Многооконные приложения Qt

Существуют несколько подходов к созданию многооконных приложений в Qt. Каждый из них применяется в зависимости от того, нужно ли показывать несколько окон одновременно или в один момент времени пользователю должно быть доступно окно.

Многооконные приложения и вкладки Qt

► QMdiArea



Возможно динамическое создание окон. Можно использовать отдельные ui файлы (файлы форм) для каждого окна.

Многооконные приложения и вкладки Qt

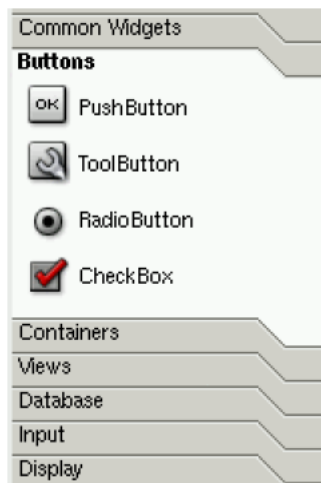
- ▶ QTabWidget



Содержание отдельных вкладок можно редактировать в дизайнера форм. Отдельные ui файлы обычно не используются.

Многооконные приложения и вкладки Qt

► QToolBox



Многооконные приложения и вкладки Qt

- ▶ QStackedWidget

Виджет который позволяет показывать только одно окно из нескольких в один момент времени.

Многооконные приложения Qt

youtube.com/watch?v=VigUMAfE2q4 How to Show Another Window From MainWindow in QT

Outline

Прошлые темы

Многооконные приложения

Язык описания UI

QML

Ссылки

Front-end и back-end

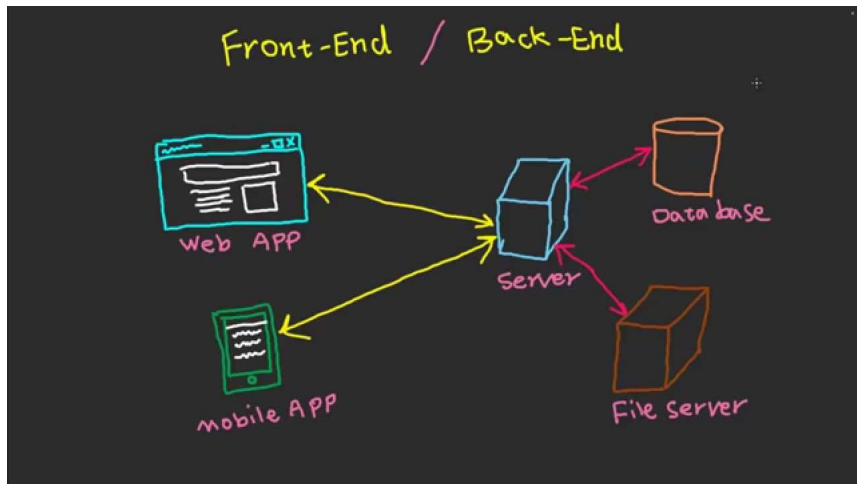
Фронтенд (Front-end) — клиентская сторона ПО. Как правило к front-end относят разработку интерфейсов пользователя и т.п., например создание дизайн-макета сайта, вёрстку сайта, создание скриптов описывающих поведение пользовательского интерфейса.

Front-end и back-end

Бекенд (англ. back-end) — программно-аппаратная часть, ядро сайта или программы. Включает в себя бизнес-логику.

Back-end создает, некоторое API, которое использует front-end. Таким образом front-end разработчик не должен знать особенностей реализации сервера, а back-end разработчик реализацию front-end.

Front-end и back-end

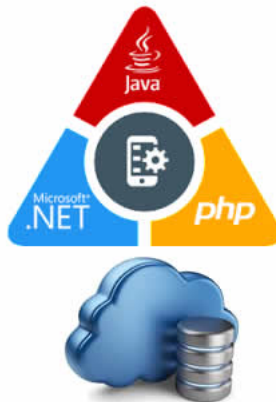


Front-end и back-end

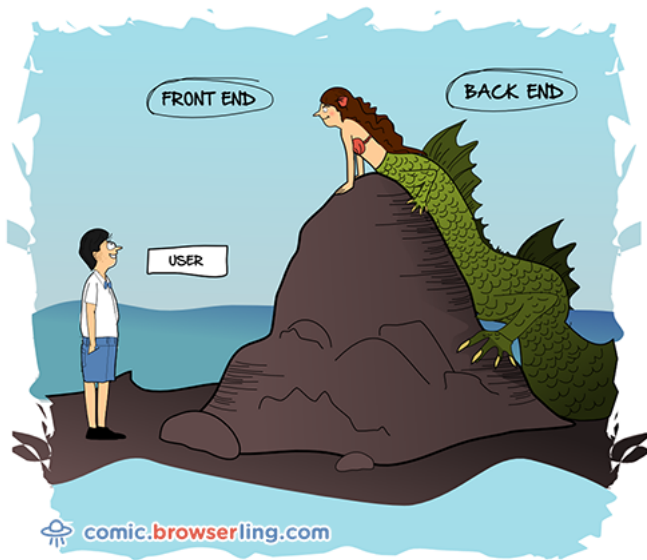
FRONT END



BACK END



Front-end и back-end



Front end vs. Back end.

Язык описания UI

Язык описания интерфейса пользователя (User Interface, UI) - язык разметки¹ предназначенный для рисования и описания графического интерфейса пользователя (GUI).

Такие языки переносят идея повторного использования кода в область создания интерфейса пользователя, позволяют детально описывать внешний вид элементов интерфейса и иногда их поведение.

¹примеры языков разметки: HTML, TEX, XML

Языки описания UI

- ▶ **QML** (Qt Modeling Language) - декларативный язык программирования, основанный на JavaScript, предназначенный для дизайна приложений, делающих основной упор на пользовательский интерфейс. QML код может содержать Java Script.
- ▶ **XAML** (eXtensible Application Markup Language) — основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft. XAML также может описывать логику приложений.

XAML. Пример

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid Name="MainGrid">
        <StackPanel Name="StackPanel1">
            <Button Name="Button1">First Button</Button>
            <Button Name="Button2">Second Button</Button>
        </StackPanel>
    </Grid>
</Window>
```

XAML

Создание приложения "Hello, world" (XAML)

youtu.be/4DDonpLldLM - Видеокурс Windows Presentation Foundation (WPF). Урок 1. Введение в WPF и XAML

Outline

Прошлые темы

Многооконные приложения

Язык описания UI
QML

Ссылки

Доклад на конференции "C++ Siberia" о QML и Qt Quick:
Преимущества и использование.

[youtube: Сергей Хомяков, QML Qt Quick на практике](#)

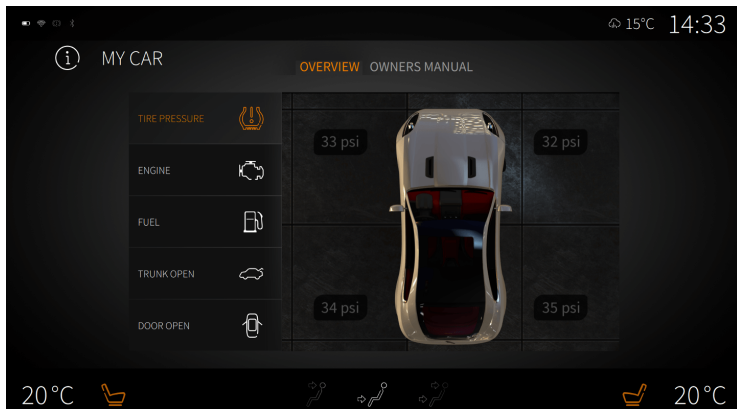
Особенности QML

- ▶ Декларативный язык ²
Код не компилируется.
- ▶ Синтаксис похож на JSON ³
- ▶ Интегрируется с C++ кодом (с использованием Qt)
- ▶ Создание гибко настраиваемых элементов интерфейса
- ▶ Может включать JavaScript, HTML, CSS
- ▶ Использует встроенный в Qt Creator QLM дизайнер (Qt Quick Designer)
см. демонстрацию: youtube.com/watch?v=cOVIdcYWNCI
- ▶ Интерфейс - дерево элементов

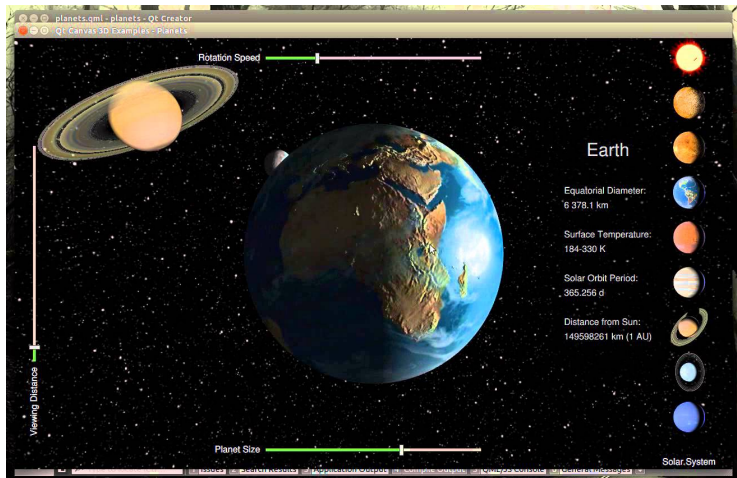
²см. декларативное и императивное программирование

³wikipedia: JSON

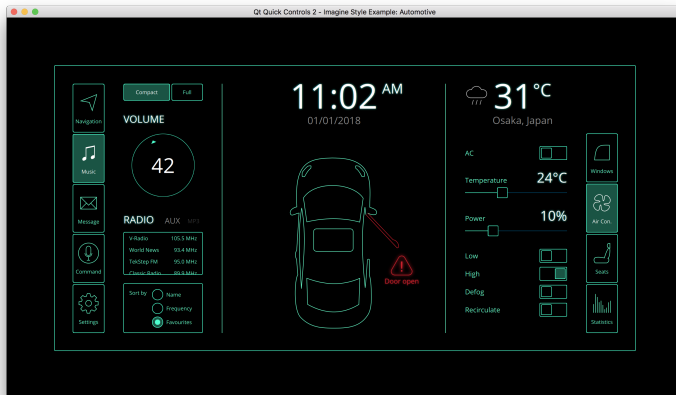
Примеры QML интерфейсов



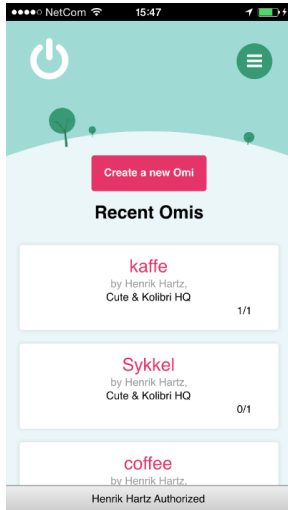
Примеры QML интерфейсов



Примеры QML интерфейсов

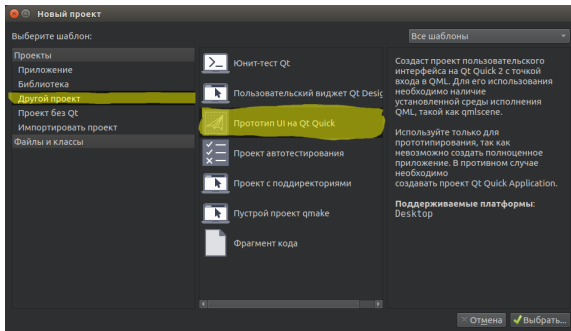


Примеры QML интерфейсов



Создание QML проекта

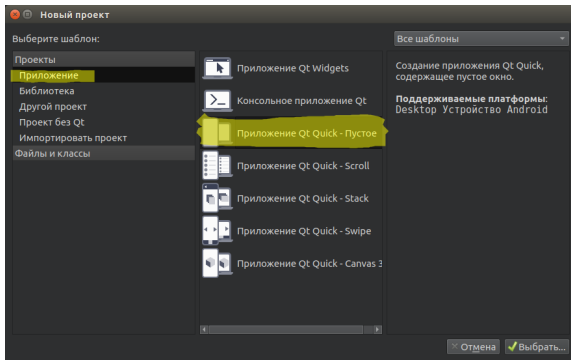
Создание QML проекта из одного QML файла.



Такой QML файл можно запускать в Qt Creator как самостоятельный проект или использовать QML файл в другом проекте.

Создание QML проекта

Создание проекта использующего QML и C++



Hello World

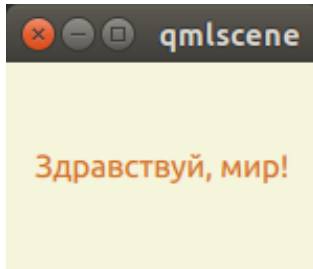
```
import QtQuick 2.0

Rectangle {
    color: "beige"
    width: 150; height: 100

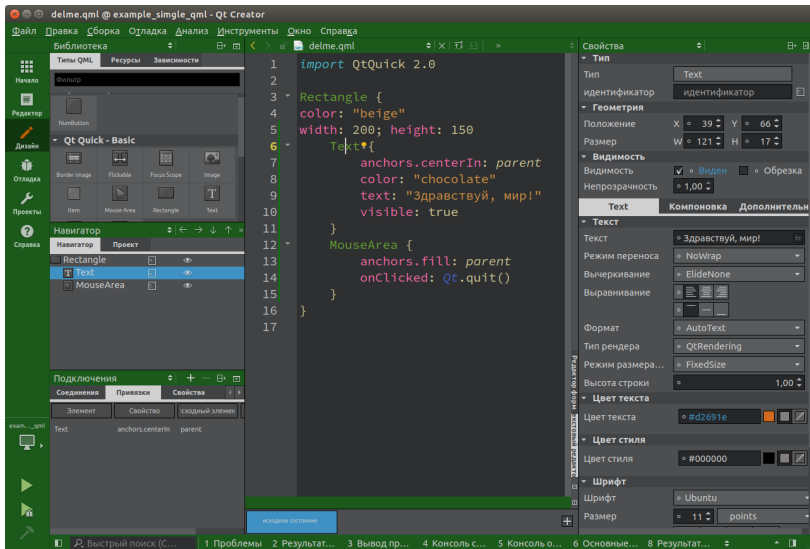
    Text {
        anchors.centerIn: parent
        color: "chocolate"
        text: "Здравствуй, мир!"
    }

    MouseArea {
        anchors.fill: parent
        onClicked: Qt.quit()
    }
}
```

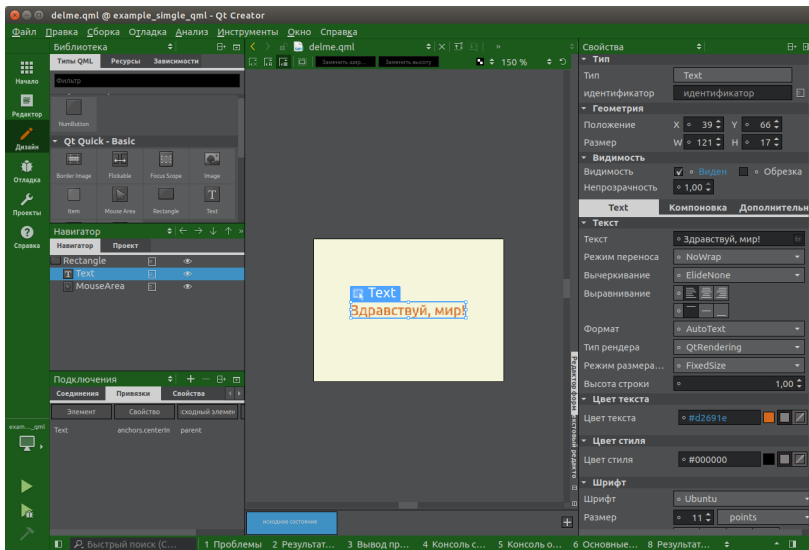
Hello World



Qt Quick Designer



Qt Quick Designer



Qt Quick Designer

<https://www.youtube.com/watch?v=cOViDcYWNCI> - работа в Qt Quick Designer

Повторное использование QML

Элементы интерфейса описанные с помощью QML можно сохранять в отдельных файлах, а затем использовать так, как будто это стандартные компоненты.

В Qt Quick Designer для этого нужно выбрать элемент интерфейса в навигаторе затем сохранить в отдельном файле через контекстное меню.

Имя созданного элемента интерфейса будет совпадать с именем файла, в котором он описан.

Qt Quick – современная технология создания UI, особенностью которой является разделение декларативного описания дизайна интерфейса и императивной логики программирования.

QML является составляющей Qt Quick.

Выполнение QML кода

- ▶ с помощью Qt Creator
- ▶ с помощью программы **qmlscene**⁴

```
qmlscene my_beautiful_ui.qml
```

Однако qml файл не является отдельной программой, так как его необходимо интерпретировать.

⁴расположена в каталоге где установлен Qt, например:
Qt5/5.11.0/gcc_64/bin

Qt Quick

В Qt существуют классы, которые выполняют QML код:

- ▶ QQuickView

```
QQuickView *view = new QQuickView;  
view->setSource(QUrl::fromLocalFile("myqmlfile.qml"));  
view->show();
```

Перед использование qml файл должен быть добавлен к проекту. В этой программе не реализовано никакой бизнес логики вне QML файла.

В Qt существуют классы, которые выполняют QML код:

- ▶ QQmlApplicationEngine

```
// Загрузка и разбор qml файла  
QQmlApplicationEngine engine;  
engine.load(QUrl(QStringLiteral("qrc:/main.qml")));  
  
if (engine.rootObjects().isEmpty()) return -1;
```

В QML файле все элементы должны быть помещены
внутри Window.

Перед использование qml файл должен быть добавлен к
проекту. В этой программе не реализовано никакой бизнес
логики вне QML файла.

Рекомендуется использовать именно этот класс.

Элементы интерфейса

```
import QtQuick.Controls 2.2
```

doc.qt.io/qt-5/qtquick-controls-qmlmodule.html - список элементов интерфейса и документация

Некоторые элементы интерфейса

- ▶ Window - главное окно приложения
- ▶ Item - базовый элемент интерфейса (аналог QWidget)
- ▶ Button
- ▶ TextField - поле ввода
- ▶ TextArea - многострочное поле ввода
- ▶ SpinBox - поле ввода для чисел
- ▶ Label
- ▶ Image
- ▶ ComboBox
- ▶ CheckBox

Настройка отдельных элементов интерфейса

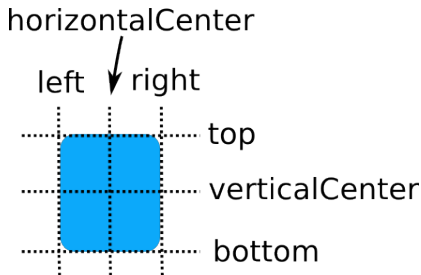
Так как все видимые элементы интерфейса так или иначе построены на основе `Item`, то у них есть общие наборы свойств

- ▶ `id` - идентификатор объекта
- ▶ `x`, `y`
- ▶ `z` - задаёт "глубину" расположения элемента интерфейса
- ▶ `width`, `height`
- ▶ `color`
- ▶ `rotation` - угол поворота (по часовой стрелке, в градусах)
- ▶ `scale` - масштаб
- ▶ `visible anchors` - составное свойство, описывающее привязку расположения и границ объекта (якоря)
- ▶ `opacity` - прозрачность (по умолчанию 1.0)

anchors

anchors (якоря) используются для привязки размеров и положения объектов интерфейса к другим объектам.

Например можно привязать надпись к горизонтальному центру окна, или привязать правую границу одного элемента интерфейса, к левой другого.



см. документацию doc.qt.io/qt-5/qtquick-positioning-anchors.html

anchors. Примеры

```
Rectangle { id: rect1; ... }
```

```
Rectangle {  
  id: rect2;  
  anchors.left: rect1.right;  
  anchors.leftMargin: 5; ... }
```

`anchors.fill: parent` - заполнить родительский объект
`centerIn: parent` - всегда в центре родительского объекта
`horizontalCenter: parent.horizontalCenter` - на вертикальной середине родительского объекта

Layouts

Для управления взаимным положением элементов интерфейса также как и в Qt Designer используются специальные компоновщики.

```
Row { // Горизонтальное расположение
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
}
```



Здесь для динамического изменения размеров содержимого компоновщика, а не только контроля положения, нужно использовать `RowLayout`.

Изображения. Image

```
Image {  
  id: img  
  anchors.fill: parent  
  // В компонент Image можно помещать не только  
  // предварительно добавленные в проект изображения  
  // но и изображения из Интернета  
  source:  
  "https://apod.nasa.gov/apod/image/1805/Pleiades_WiseAntonucci_960.jpg"  
}
```

Image и BusyIndicator

Если используется изображение из сети, то можно снабдить программу индикатором загрузки

```
...
Image {
    id: img
    anchors.fill: parent
    source:
        "https://apod.nasa.gov/apod/image/1805/Pleiades_WiseAntonucci_960.jpg"
}

BusyIndicator {
    id: load_indicator
    anchors.centerIn: parent
    running: img.status === Image.Loading
}
...
```

Image

Помимо якоря для изображения можно задавать *способ* заполнения родителя - `fillMode`:



Stretch



PreserveAspectFit



PreserveAspectCrop



Tile



TileHorizontally



TileVertically

Градиент

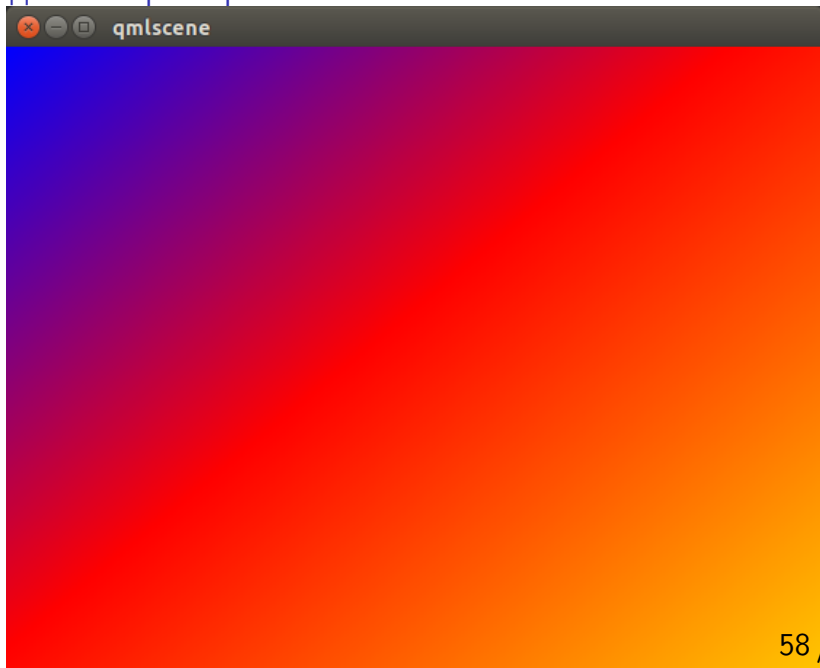
```
LinearGradient{
    anchors.fill: parent
    start: Qt.point(0,0)    // Начало градиента
    end: Qt.point(rect.width, rect.height) // конец градиента

    gradient: Gradient{
        // Список из точек градиента
        // расположение задаётся относительной координатой position
        // на прямой от start до end
        GradientStop{position: 0; color: "blue"}
        GradientStop{position: 0.4; color: "red"}
        GradientStop{position: 1; color: "yellow"} }
    }
```

Градиент изображается средствами CPU, не GPU поэтому его стоит использовать как статический элемент, чтобы не расходовать много процессорного времени.

Помимо использования отдельного компонента для градиента (например LinearGradient) свойство gradient можно задавать для различных элементов интерфейса. Однако такой градиент всегда вертикальный.

Градиент. Пример



Анимации

- ▶ Анимация является отдельным компонентом
- ▶ Анимация обычно включает в себя набор базовых анимаций, которые изменяют одно или несколько свойств конкретного объекта.
Например базовая анимация `NumberAnimation` может задавать изменение одного числового свойства объекта.
- ▶ Базовые анимации могут выполняться последовательно или параллельно внутри соответствующих компонентов
 - ▶ **`ParallelAnimation`**
 - ▶ **`SequentialAnimation`**
- ▶ Для запуска анимации нужно задать её свойство *running*
`my_anim.running = true`

<http://doc.qt.io/qt-5/qtquick-usecase-animations.html>

Анимации. Пример

Одновременное изменение ширины и высоты прямоугольника

```
ParallelAnimation {  
    id: my_anim  
  
    // изменение ширины компонента my_rect  
    // от 50 до 1000 пикселей  
    // в течении 350 миллисекунд  
    // easing.type задаёт функцию, согласно которой  
    // будет изменяться целевой параметр (width)  
    NumberAnimation{target: my_rect; property: "width";  
        easing.type: Easing.InCubic  
        from: 50; to: 1000; duration: 350 }  
  
    NumberAnimation{target: my_rect; property: "height";  
        easing.type: Easing.InCubic  
        from: 50; to: 1000; duration: 350 }  
  
    // дополнительно можно описать действия выполняемые после  
    // завершения анимации  
    onStopped: {my_rect.color = "red";}  
}
```

Некоторые обработчики событий

- ▶ **onCompleted** - обработчик запускаемый при создании объекта интерфейса
- ▶ **onClicked** (для `MouseArea`⁵)
- ▶ **onEditingFinished** - закончено редактирование поля ввода
- ▶ **onValueChanged** - изменено значение `SpinBox`
- ▶ **onPressed** - обработчик нажатия кнопки

⁵ невидимый объект, который может получать события мыши

Обработчики событий. Пример

```
import QtQuick 2.0

Rectangle {
    id: rect
    width: 100; height: 100

    MouseArea {
        anchors.fill: parent
        onClicked: {
            rect.color =
                Qt.rgb(Math.random(), Math.random(), Math.random(), 1)
        }
    }
}
```

doc.qt.io/qt-5/qtqml-syntax-signals.html

Использование JavaScript

List of JavaScript Objects and Functions

Для работы с математическими функциями используется модуль **Math**. Модуль подключается автоматически.

для преобразования типов используются методы

- ▶ `toString`
- ▶ `toFixed(число знаков)` - преобразование вещественного числа в строку
- ▶ `console.log("hello")` - вывод в консоль (используется для отладки)

В QML можно загружать JavaScript файлы, которые могут содержать сложную логику работы UI. [Например D3.js](#)

Взаимодействие C++ и QML

- ▶ Программист не реализует собственный класс представляющий главное окно приложения.
- ▶ Вместо этого задача программиста - создать набор классов с бизнес-логикой, которые будут подключены в QML файл и использованы там.
- ▶ Проектировщик интерфейса использует классы с бизнес логикой так, как будто это стандартные компоненты QML
- ▶ События генерируемые пользователем привязываются к классам с бизнес логикой.

см. пример github.com/VetrovSV/OOP/tree/master/example_qml

Взаимодействие C++ и QML

Перед использование C++ класса (производного от QObject) его применение нужно явно обозначить:

Программа интерфейс который описан с помощью QML будет выглядеть так:

```
...  
// сделать класс Calc доступным в QML коде  
qmlRegisterType<Calc>("calc", 1, 0, "Calc");  
// calc - имя модуля  
// 1,0 - версия модуля  
// Calc имя типа  
  
// Загрузка и разбор qml файла  
QQmlApplicationEngine engine;  
...
```

Outline

Прошлые темы

Многооконные приложения

Язык описания UI
QML

Ссылки

Документация:

- ▶ [Qt Documentation: QML](#)
- ▶ [Integrating QML and C++](#)

Примеры и обзоры:

- ▶ [youtube: Вебинар по QML и QtQuick: часть первая](#)
- ▶ [youtube: Introduction to Qt – Intro to QML \(tutorial\)](#)
- ▶ [Начинаем работу Python + Qt5 + QML](#)
- ▶ [youtube: Сергей Хомяков, QMLQuick на практике](#)

Материалы курса

Слайды, вопросы к экзамену, задания, примеры

github.com/VetrovSV/OOP