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

Прошлые темы

- Что такое бизнес-логика?
- Что такое шаблон проектирования *Модель-Контролёр-Вид*?
- Что такое сигнал?
- Что такое слот?
- Какие классы могут содержать сигналы и слоты?
- Что такое UI?
- Что такое API?

Outline

Прошлые темы

Многооконные приложения

Язык описания UI
QML

Ссылки

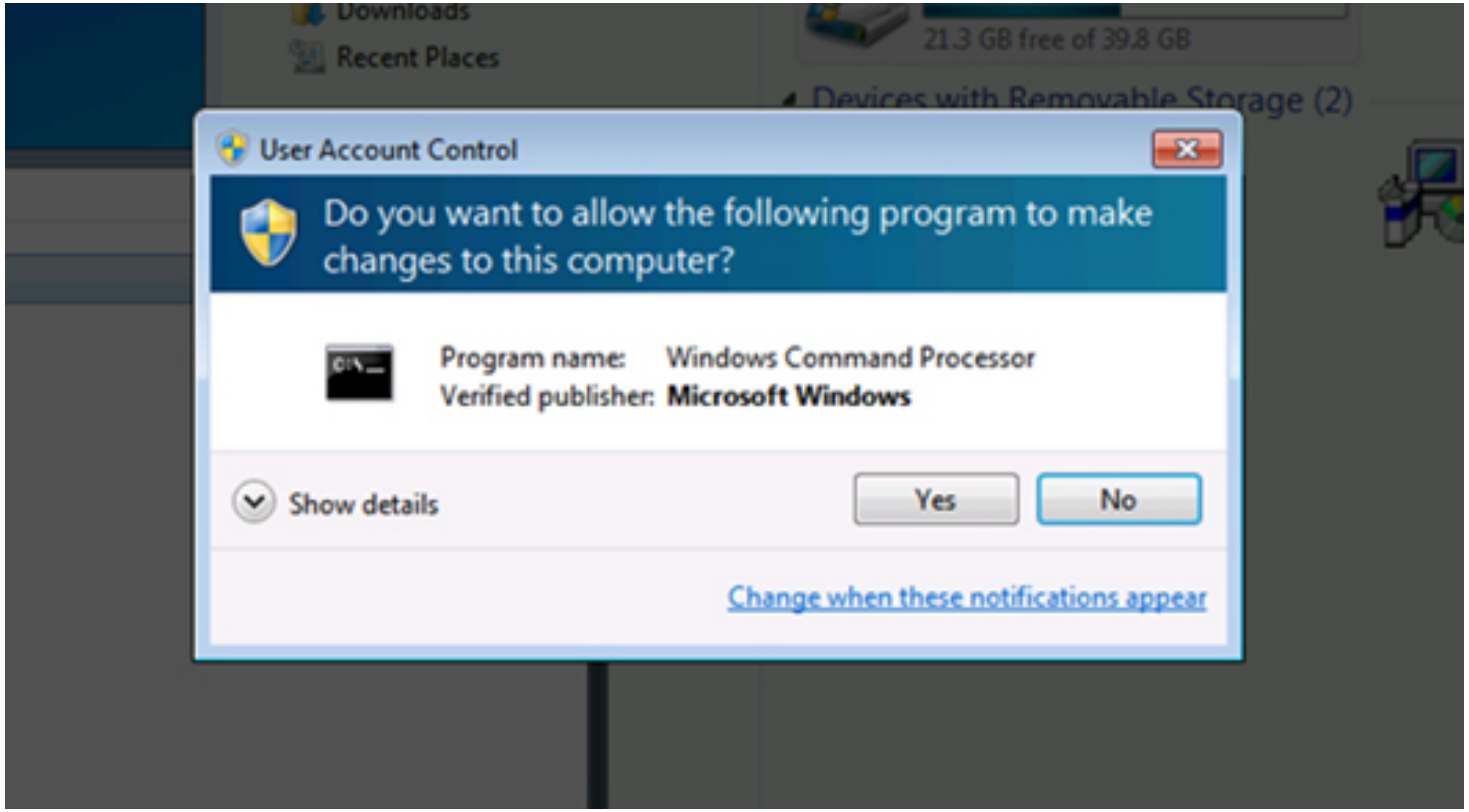
SDI

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.

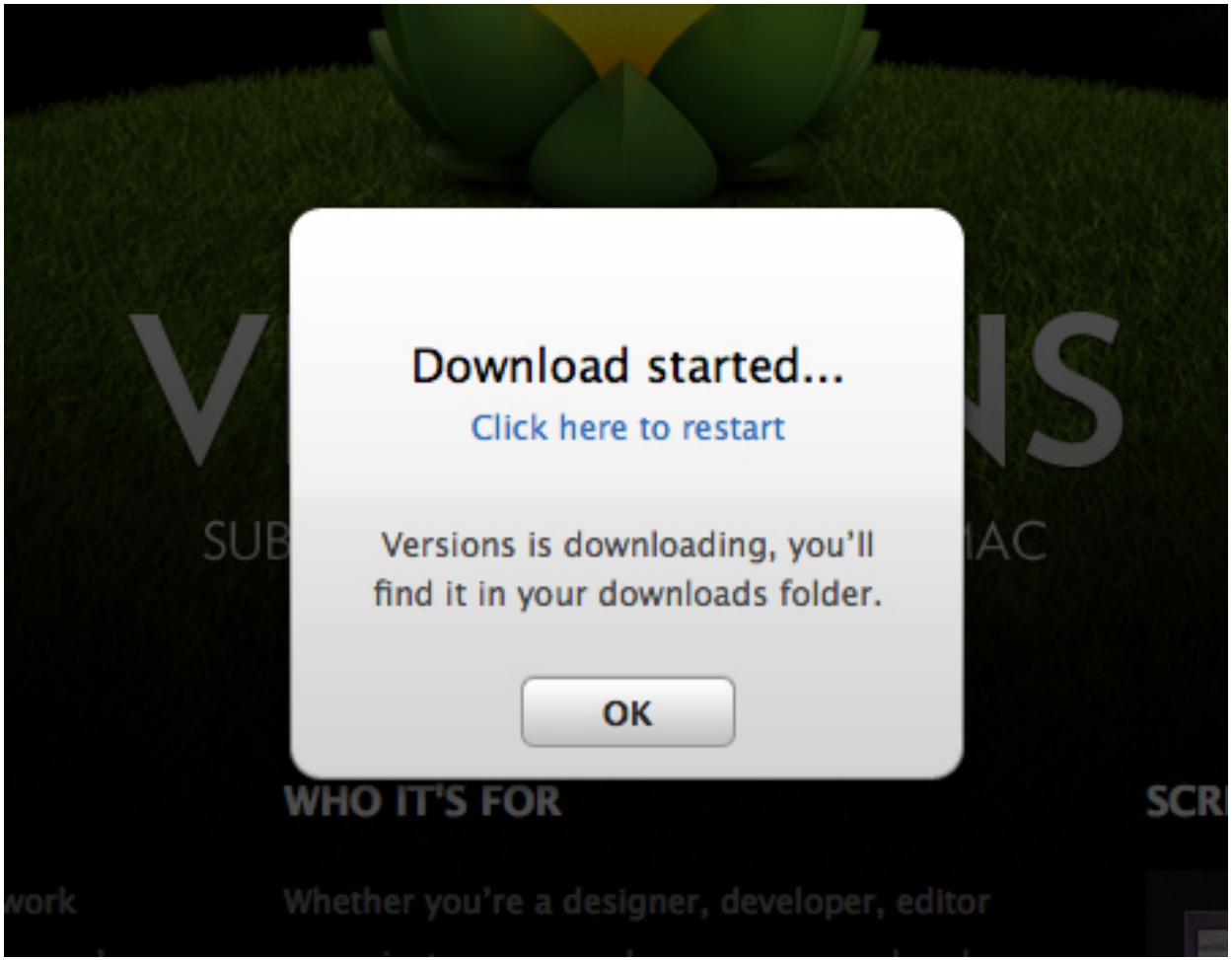
Модальное окно

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

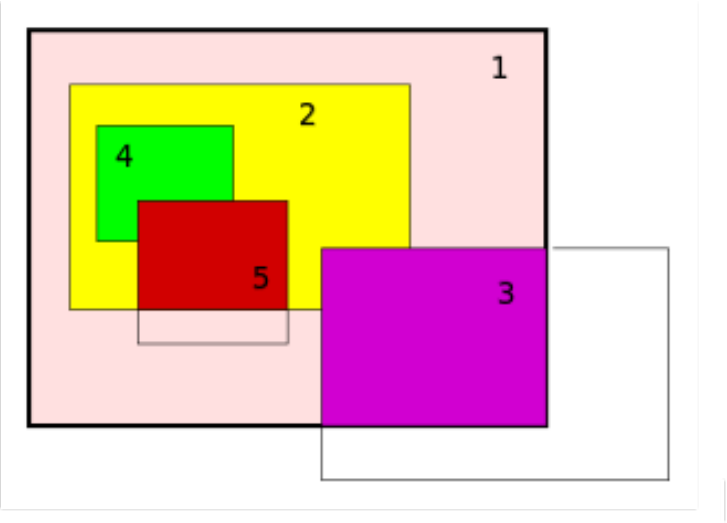


Модальные окна

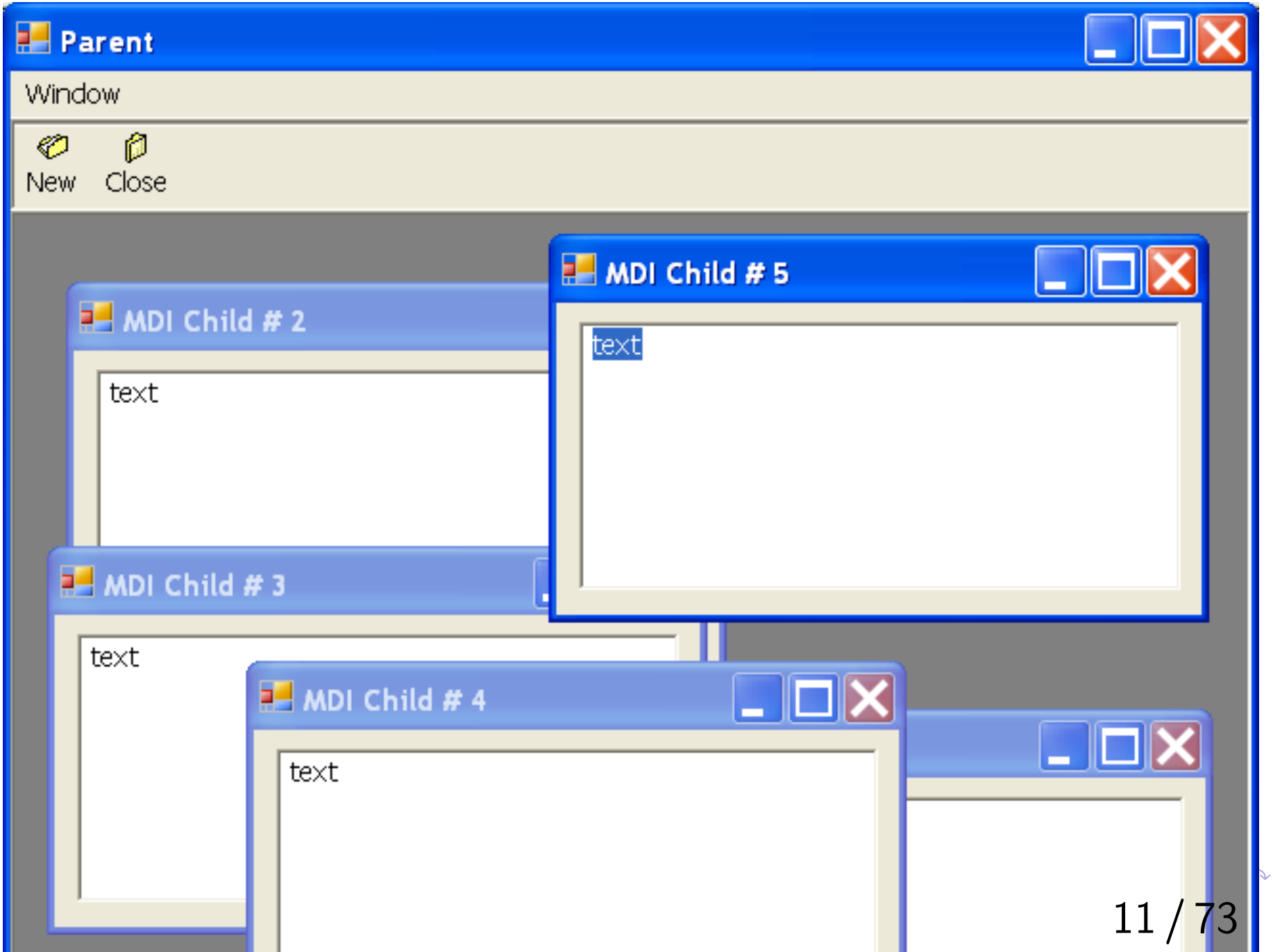


MDI

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


Parent
Window
New
Close
MDI Child # 5
text
MDI Child # 2
text
MDI Child # 3
text
MDI Child # 4
text

MDI

Файл Правка Форма Вид Настройки Окно Справка

Панель виджетов

Фильтр

Layouts
- Vertical Layout
- Horizontal Layout
- Grid Layout
- Form Layout

Spacers

Buttons
- Push Button
- Tool Button
- Radio Button
- Check Box
- Command Link Button
- Button Box

Item Views (Model-Based)
- List View
- Tree View
- Table View
- Column View

Item Widgets (Item-Based)
- List Widget
- Tree Widget
- Table Widget

Containers
- Group Box
- Scroll Area
- Tool Box
- Tab Widget
- Stacked Widget
- Frame
- Widget
- MdiArea
- Dock Widget

Input Widgets
- Combo Box
- Font Combo Box
- Line Edit
- Text Edit

Заказ - orderfrm.ui*

Фильтр с 01.01.00 по 01.01.00

OK № Дата По

Показывать закрытых клиентов

Нач. Кон. Кол. Пол Спец. Отдел

Закрыть заказ Открыть заказ

OK Чел. Кол.

	Заказано	Спланировано	Отработано
чел	0	0	0
час	0		0

Стоимость	0	Часы	0
Себестоимость	0	Люди	0
Рентабельность	0		

Показывать всех

OK Уч. № ФИО Специальность Тек Пр.

F_NAME
I_NAME
O_NAME
TABNO
PHONE
Записать на

УВОЛЕН

Отработано (тек/пред): 0/0

История Удержания Начисления

Инспектор объектов

Объект
- wdgClient
  - horizontalLayo
    - btnAddWSumm
    - btnCloseOrder
    - btnEditWSumm
    - btnOpenOrder
    - btnRemoveWS
    - horizontalSpace
    - lbOrderSign
  - frame_3
    - label_11

Редактор свойств

Фильтр

btnCloseOrder : QPushButton

Свойство	Значение
QObject	
objectName	btnCloseOrder
QWidget	
enabled	✓
geometry	[(91, 1), 91 x 24]
sizePolicy	[Minimum, Fix...
minimumSize	0 x 0
maximumSize	16777215 x 16...
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Унаследованная
font	A [Ubuntu, 9]
cursor	Arrow
mouseTracking	
focusPolicy	StrongFocus
contextMenuPo	DefaultContex

Редактор Сигналов/Слотов

Отправитель	Сигнал	Получа
cbCustomer	current...String)	lbCust

Редактор Сиг... Редак... Обозреват...

Многооконные приложения Qt

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

Многооконные приложения и вкладки Qt

- QMdiArea

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

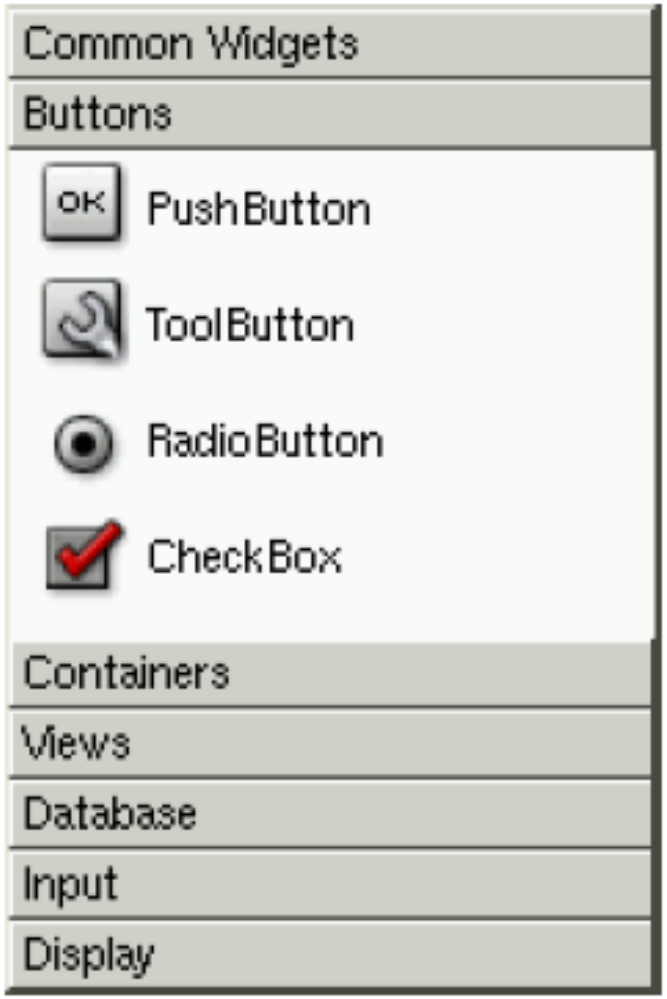
Многооконные приложения и вкладки Qt

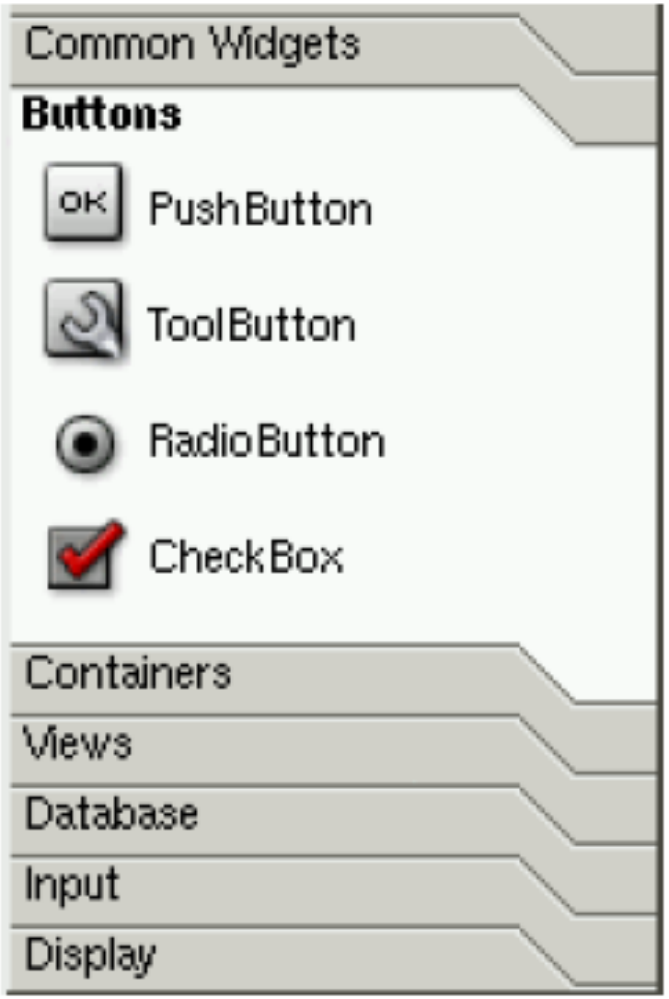
- QTabWidget

Содержание отдельных вкладок можно редактировать в дизайнере 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

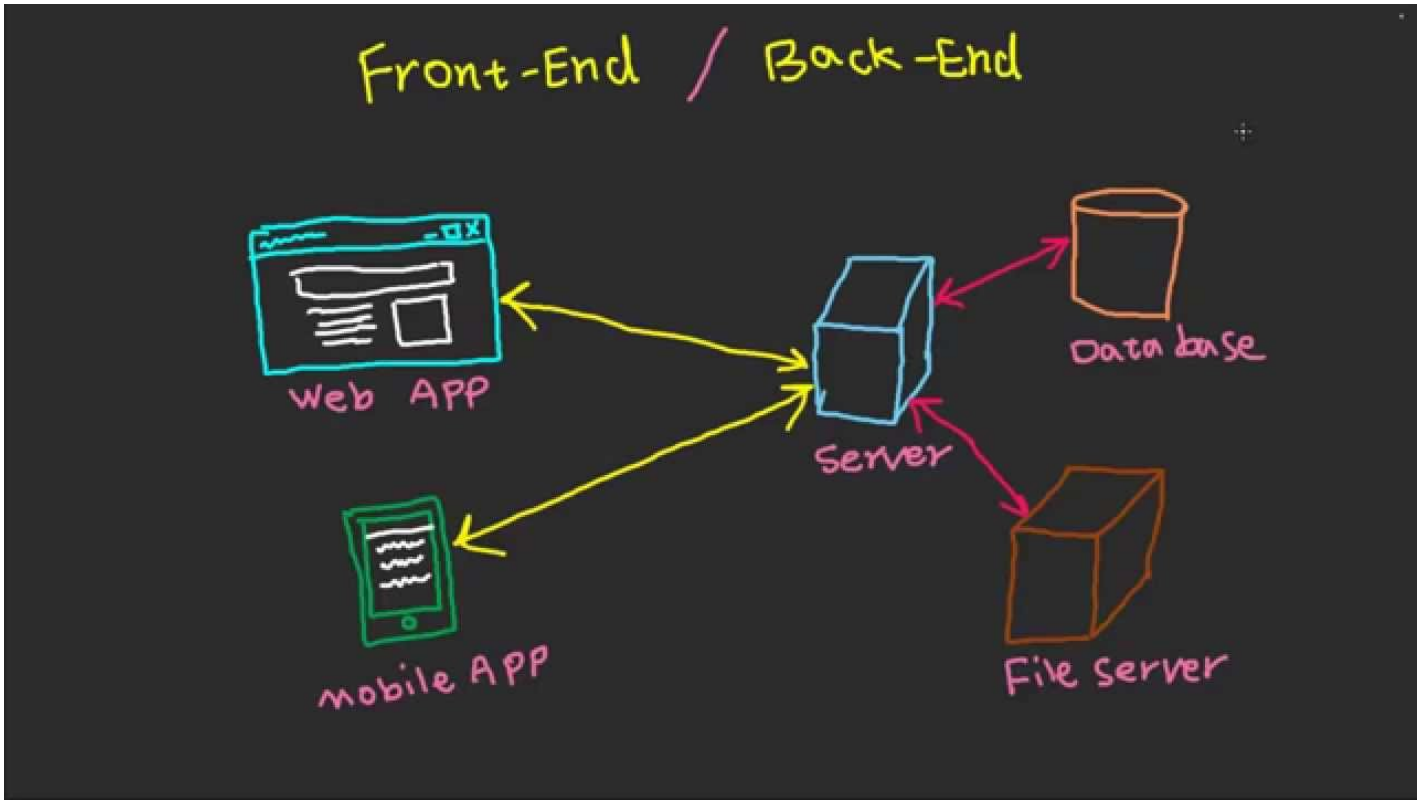
Фронтенд (Front-end) — клиентская сторона ПО. Как правило к front-end относят разработку интерфейсов пользователя и т.п., например создание дизайн-макета сайта, вёрстку сайта, создание скриптов описывающих поведение пользовательского интерфейса.

Front-end и back-end

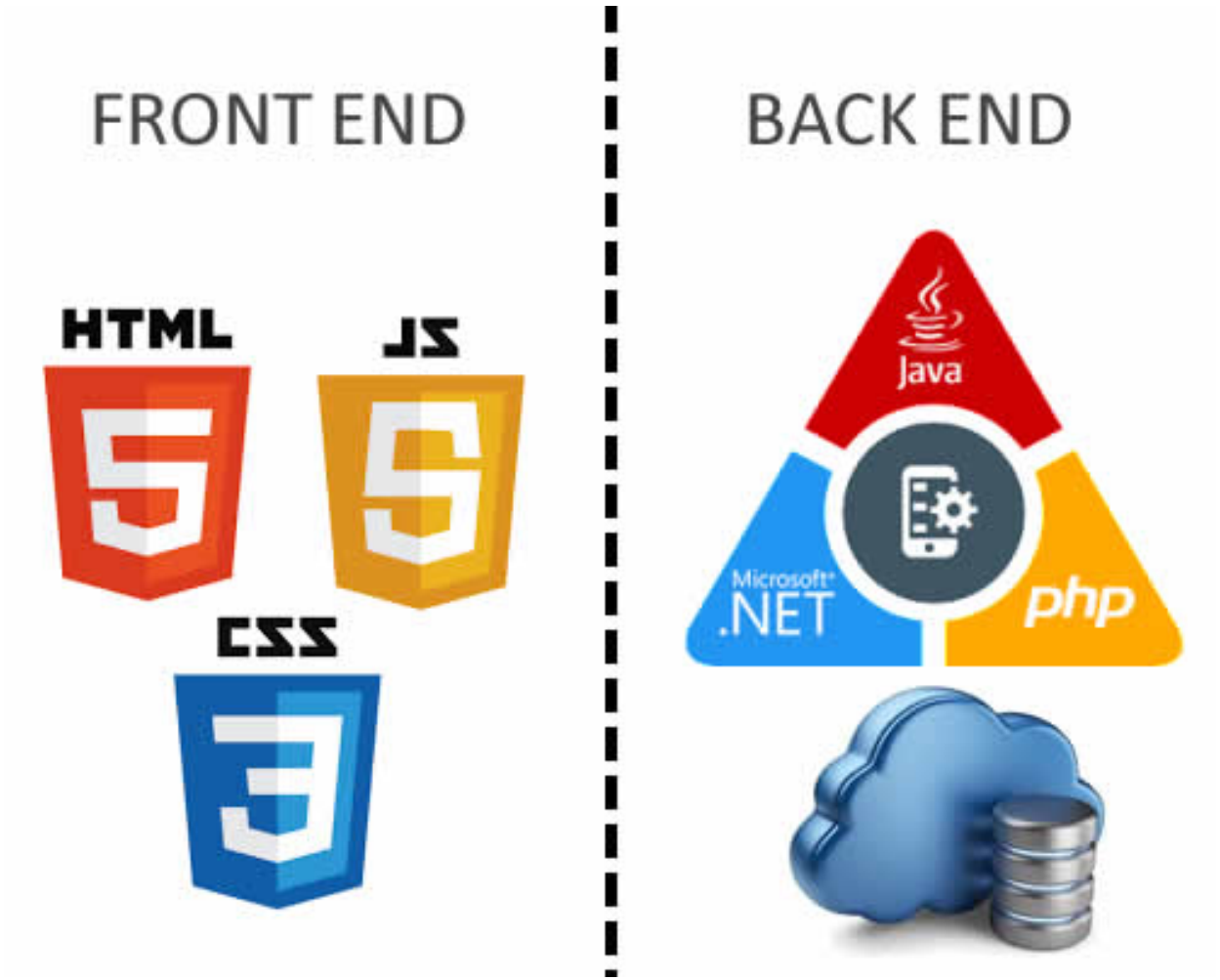
Бекенд (англ. back-end) — программно-аппаратная часть, ядро сайта или программы. Включает в себя бизнес-логику.

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.

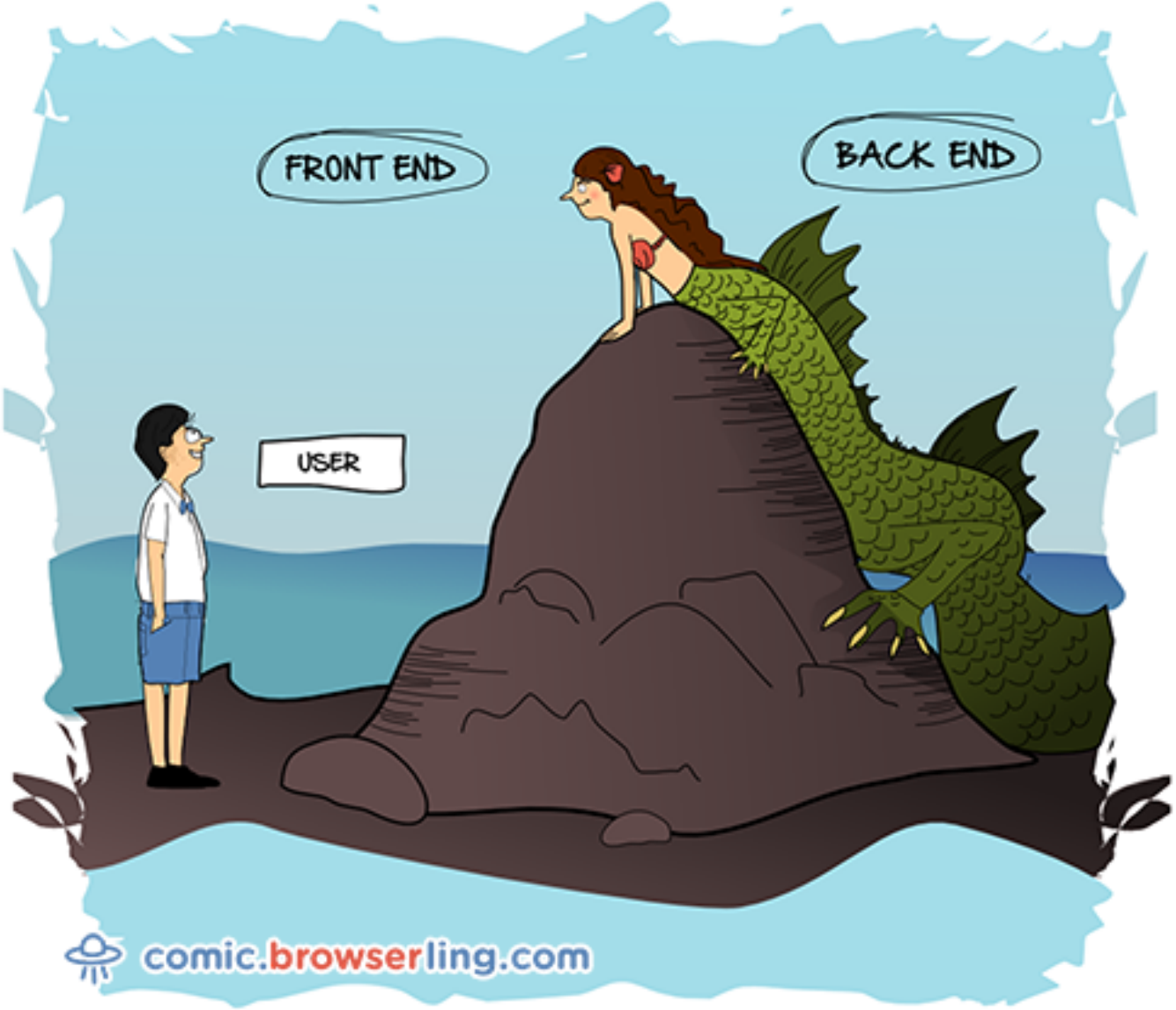
Front-end и back-end



Front-end и back-end



Front-end и back-end



Front end vs. Back end.

Язык описания UI

Язык описания интерфейса пользователя (User Interface, UI) - язык разметки[1] 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.

[1] примеры других языков разметки: HTML, TEX, XML

Языки описания UI

- **QML** (Qt Modeling Language) - декларативный[2] язык программирования, основанный на JavaScript, предназначенный для дизайна приложений, делающих основной упор на пользовательский интерфейс.
- **XAML** (eXtensible Application Markup Language) — основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft. XAML также может описывать логику приложений.

[2] см. декларативное и императивное программирование

XAML. Пример

Окно с двумя кнопками

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentatio
        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

QML

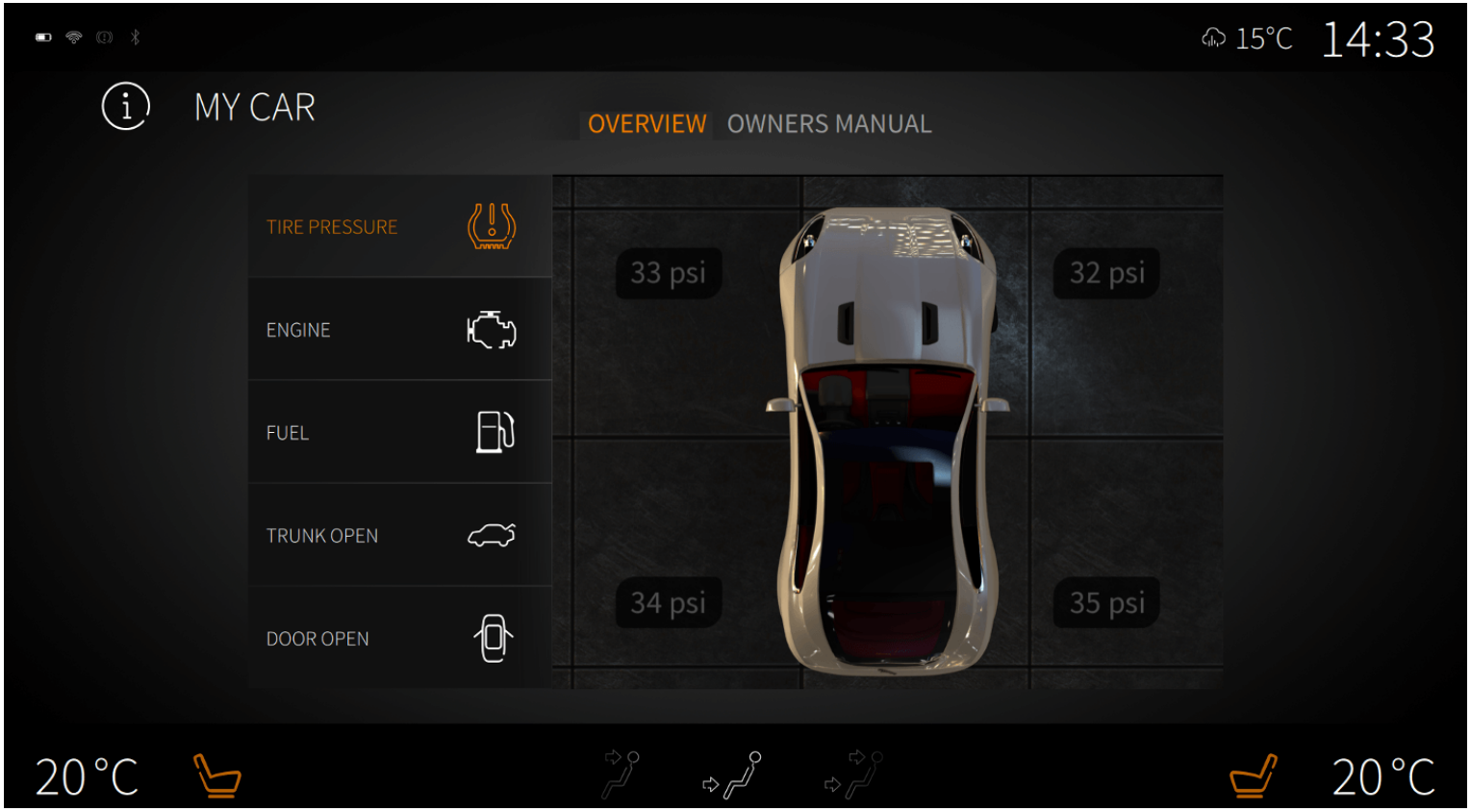
Доклад на конференции "C++ Siberia"о QML и Qt Quick: Преимущества и использование.
youtube: Сергей Хомяков, QML Qt Quick на практике

Особенности QML

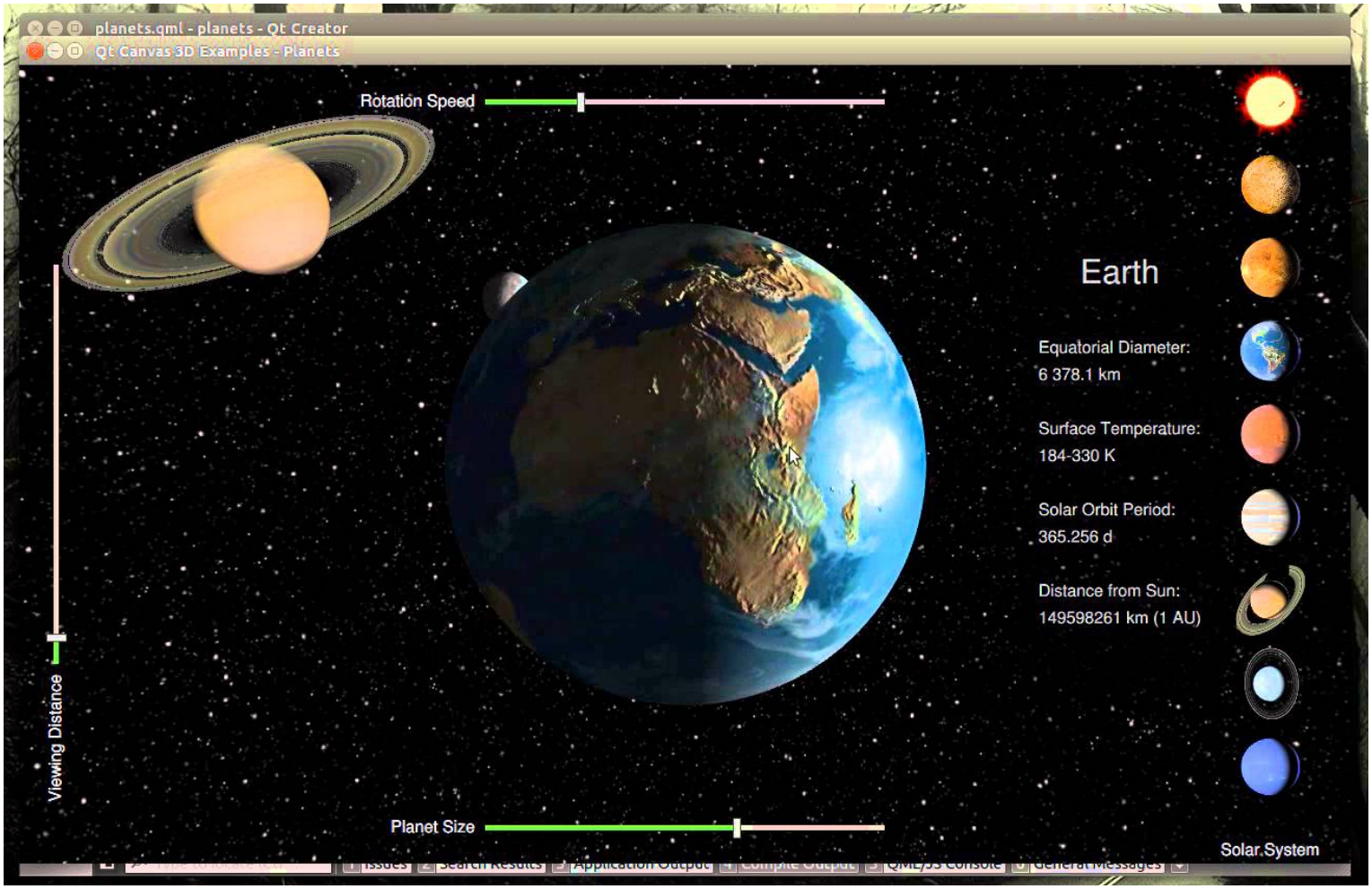
- Декларативный язык
  Код не компилируется.
- Синтаксис похож на JSON [3]
- Интегрируется с C++ кодом (с использованием Qt)
- Может включать JavaScript, HTML, CSS
- Структура QML файла - дерево из отдельных элементов UI

[3] wikipedia: JSON

Примеры QML интерфейсов

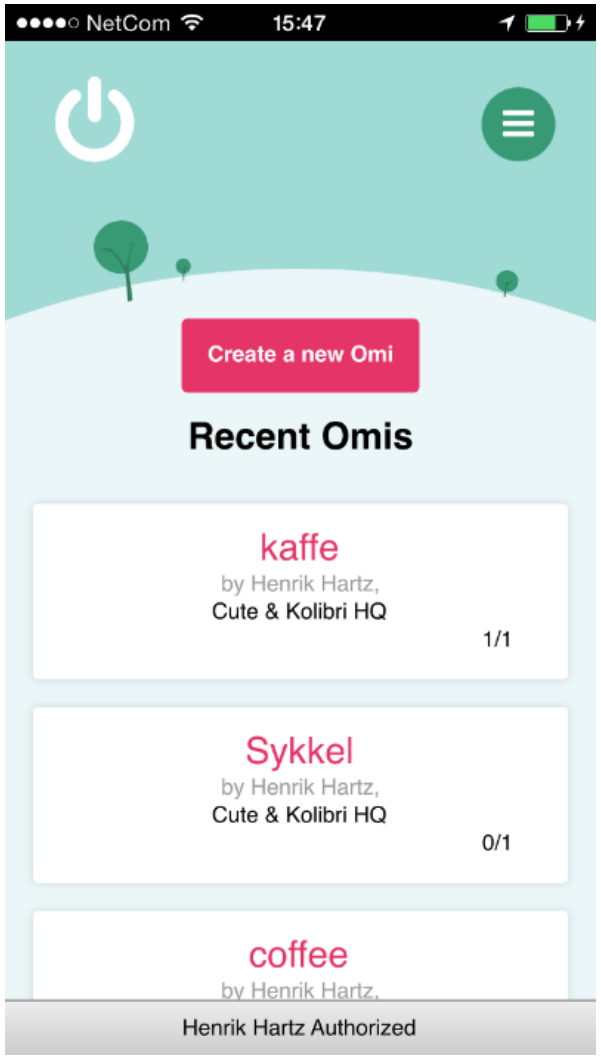


Примеры QML интерфейсов



Примеры QML интерфейсов

Примеры QML интерфейсов

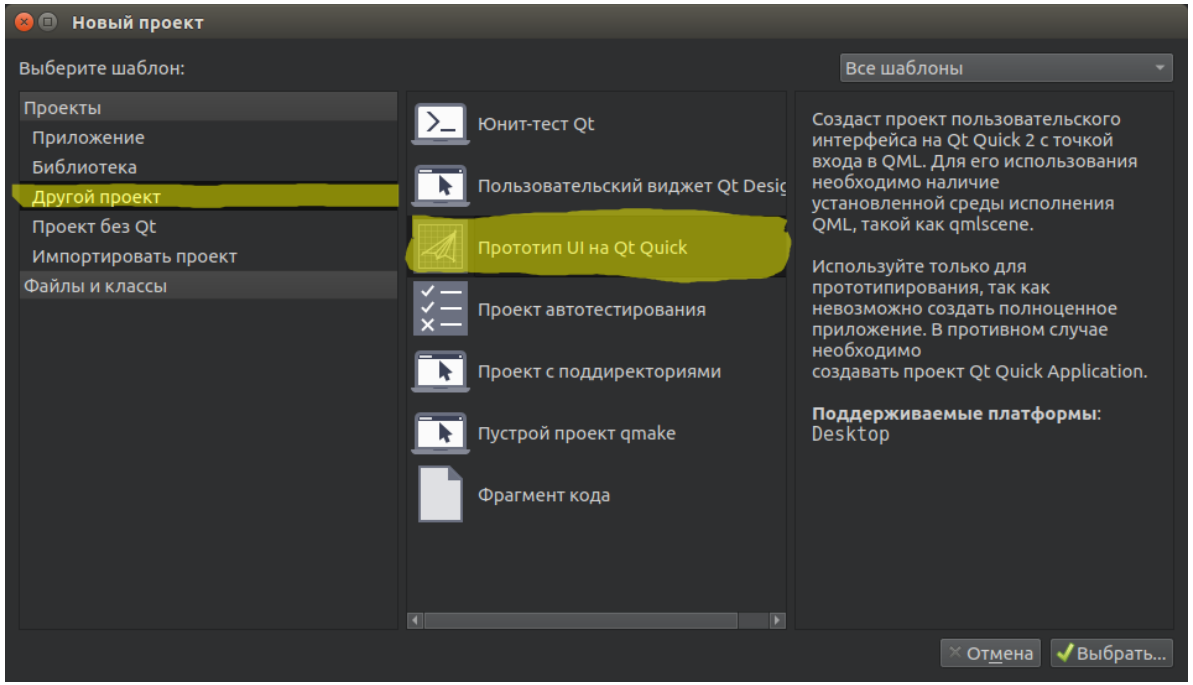


Особенности QML

- Гибкий способ описания UI (внешнего вида, поведения, анимаций и т.д.)
- Использует встроенный в Qt Creator QLM дизайнер (Qt Quick Designer)
  youtube.com/watch?v=cOViDcYWNCI
- возможно использование отдельно от основного приложения для создания прототипов UI

Создание QML проекта

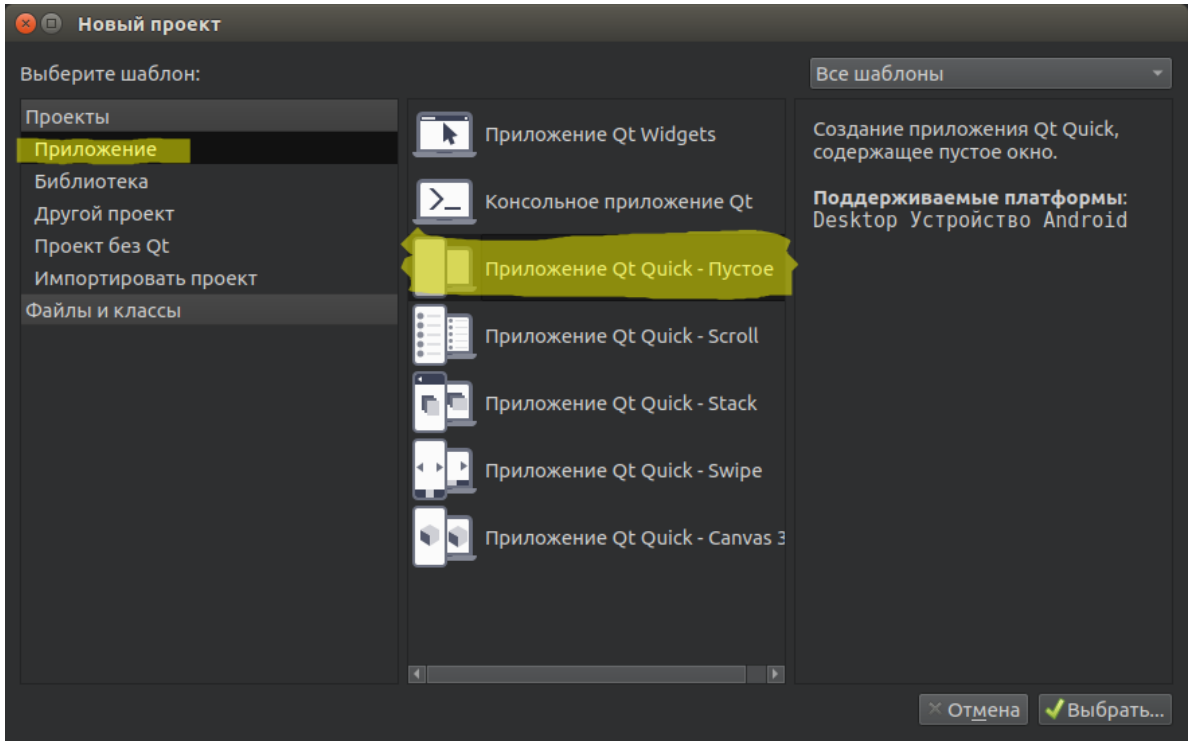
Создание QML проекта из одного QML файла.



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

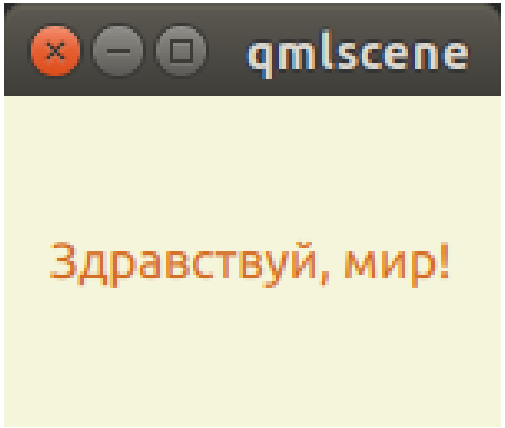
```
import QtQuick 2.0

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

delme.qml @ example_simgle_qml - Qt Creator

Файл Правка Сборка Отладка Анализ Инструменты Окно Справка

Библиотека

Типы QML Ресурсы Зависимости

Фильтр

NumButton

Qt Quick - Basic

Border Image Flickable Focus Scope Image

Item Mouse Area Rectangle Text

Навигатор

Навигатор Проект

Rectangle

Text

MouseArea

Подключения

Соединения Привязки Свойства

Элемент Свойство Исходный элемен

Text anchors.centerIn parent

delme.qml

```
1  import QtQuick 2.0
2
3  Rectangle {
4  color: "beige"
5  width: 200; height: 150
6      Text {
7          anchors.centerIn: parent
8          color: "chocolate"
9          text: "Здравствуй, мир!"
10         visible: true
11     }
12     MouseArea {
13         anchors.fill: parent
14         onClicked: Qt.quit()
15     }
16 }
17
```

исходное состояние

Свойства

Тип

Тип Text

идентификатор идентификатор

Геометрия

Положение X 39 Y 66

Размер W 121 H 17

Видимость

Видимость Виден Обрезка

Непрозрачность 1,00

Text Компоновка Дополнительн

Текст

Текст Здравствуй, мир!

Режим переноса NoWrap

Вычеркивание ElideNone

Выравнивание

Формат AutoText

Тип рендера QtRendering

Режим размера... FixedSize

Высота строки 1,00

Цвет текста

Цвет текста #d2691e

Цвет стиля

Цвет стиля #000000

Шрифт

Шрифт Ubuntu

Размер 11 points

Быстрый поиск (C... 1 Проблемы 2 Результат... 3 Вывод пр... 4 Консоль с... 5 Консоль о... 6 Основные... 8 Результат...

Qt Quick Designer

delme.qml @ example_simgle_qml - Qt Creator

Файл Правка Сборка Отладка Анализ Инструменты Окно Справка

Библиотека

Типы QML Ресурсы Зависимости

Фильтр

NumButton

Qt Quick - Basic

Border Image Flickable Focus Scope Image

Item Mouse Area Rectangle Text

Навигатор

Навигатор Проект

Rectangle

Text

MouseArea

Подключения

Соединения Привязки Свойства

Элемент Свойство Исходный элемен

Text anchors.centerIn parent

delme.qml

Заменить шир... Заменить высоту 150 %

Text

Здравствуй, мир!

исходное состояние

Свойства

Тип

Тип Text

идентификатор идентификатор

Геометрия

Положение X 39 Y 66

Размер W 121 H 17

Видимость

Видимость Виден Обрезка

Непрозрачность 1,00

Text Компоновка Дополнительн

Текст

Текст Здравствуй, мир!

Режим переноса NoWrap

Вычеркивание ElideNone

Выравнивание

Формат AutoText

Тип рендера QtRendering

Режим размера... FixedSize

Высота строки 1,00

Цвет текста

Цвет текста #d2691e

Цвет стиля

Цвет стиля #000000

Шрифт

Шрифт Ubuntu

Размер 11 points

Быстрый поиск (C... 1 Проблемы 2 Результат... 3 Вывод пр... 4 Консоль с... 5 Консоль о... 6 Основные... 8 Результат...

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

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.

[4] расположена 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.

Qt Quick

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 внутрь 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

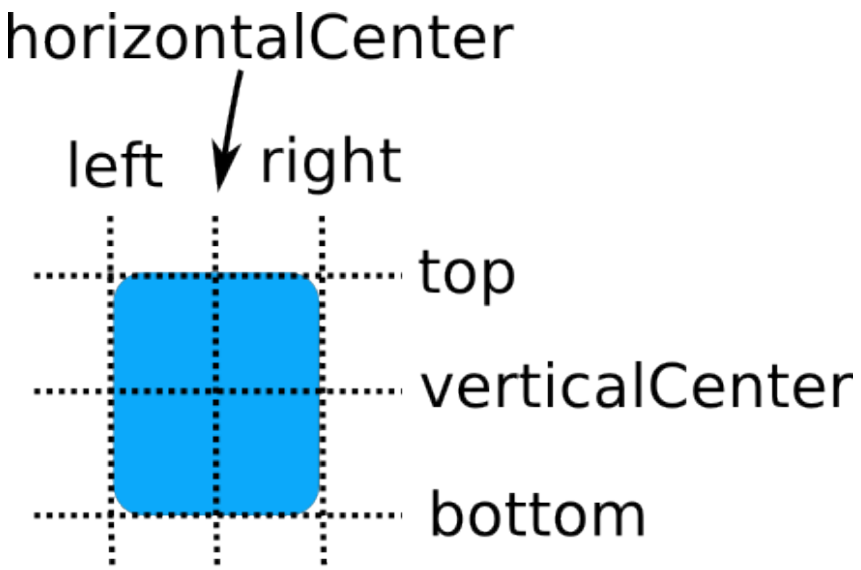
Так как все видимые элементы интерфейса так или иначе построены на основе Item, то у ни есть общие наборы свойств

- 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;
// левая граница второго прямоугольника привязана
// к правой границе первого
anchors.left: rect1.right;
// поле между границами
anchors.leftMargin: 5; ... }
```

Другие варианты задания якорей:

`anchors.fill: parent` - *заполнить родительский объект*

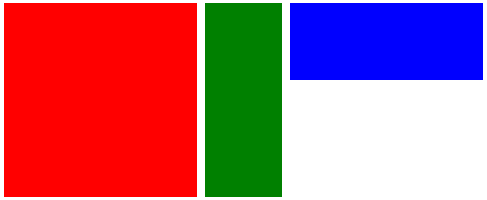
`centerIn: parent` - *всегда в центре родительского объекта*

`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 { // Горизонтльное 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:

doc.qt.io/qt-5/qml-qtquick-image.htmlfillMode-prop

Градиент

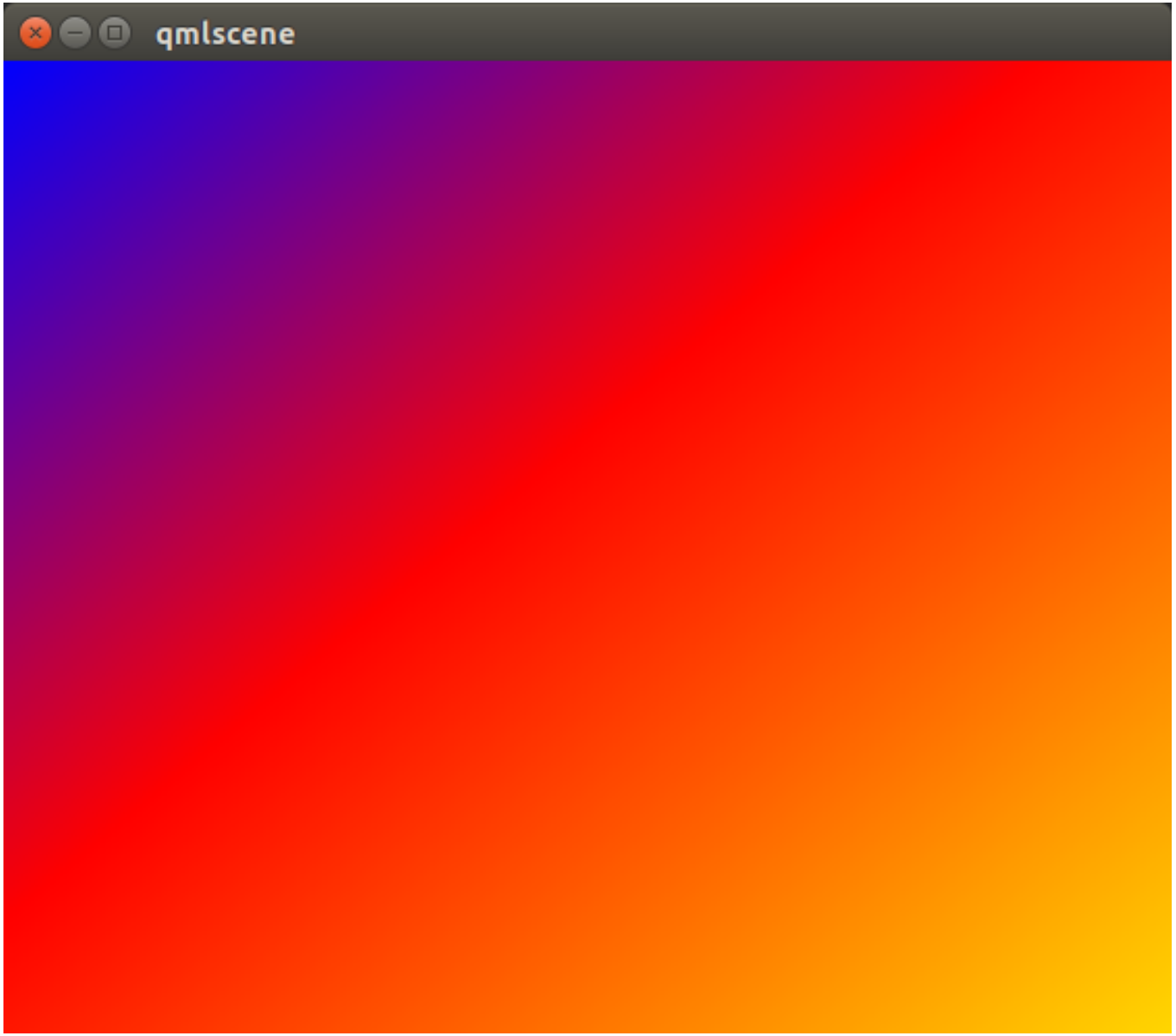
```
LinearGradient{
    anchors.fill:  parent
    start: Qt.point(0,0)   // Начало градиента
    end: Qt.point(rect.width, rect.width) // конец градиента

    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



Анимация

- 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 выполнятся 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

Анимация. 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 изменятся 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

Обработчики событий описываются подобно простым свойствам объекта, только значения свойства - код на Java Script

- **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.rgba(Math.random(), Math.random(), Math.random(), 1)
        }
    }
}
```

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

Обработка событий. Пример

Для описания реакции на нажатие клавиши предусмотрены специальные обработчики:

```
Keys.onUpPressed: rectangle.y -= 10
Keys.onDownPressed: rectangle.y += 10
Keys.onLeftPressed: rectangle.x += 10
Keys.onRightPressed: rectangle.x -= 10

focus: true
```

Чтобы компонент с обработчиками перехватил нажатие клавиши он должен находится в фокусе ввода.

doc.qt.io/qt-5/qtquick-usecase-userinput.html

Использование JavaScript

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

для преобразования числа в строку используются *методы*

- toString
- toFixed(число-десятичных-знаков)

List of JavaScript Objects and Functions
http://doc.qt.io/qt-5/qtqml-javascript-expressions.html

Использование JavaScript

для преобразования строки в число используются *функции*

- parseInt(string, radix);
- parseFloat(string);

Использование JavaScript

- 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

Переменные и функции

Если есть необходимость использовать глобальную переменную в QML её можно описать как свойство некоторого компонента:

```
property var aNumber: 100
```

Определение новых функций не отличается от такового в JavaScript

```
function factorial(a) {
    a = parseInt(a);
    if (a <= 0)
        return 1;
    else
        return a * factorial(a - 1);
}
```

doc.qt.io/qt-5/qml-var.html
doc.qt.io/qt-5/qtqml-javascript-expressions.html

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**$_q ml$

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

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