ML Самоучитель

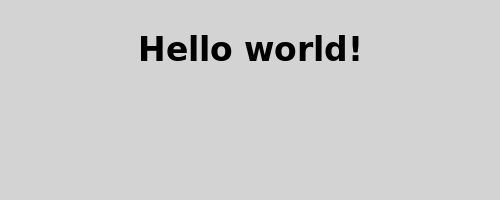
Главы руководства:

1. Базовые Типы.
2. Компоненты QML.
3. Состояния и переходы.

Глава 1 - Базовые Типы

Первая программа представляет собой очень простой "Hello world" пример, который познакомит вас с некоторыми базовыми концепциями QML.

На рисунке ниже показан скриншот этой программы:



QML код приложения:

import QtQuick 2.0

Rectangle {

id: page

width: 320; height: 480

color: "lightgray"

Text {

id: helloText

text: "Hello world!"

y: 30

anchors.horizontalCenter: page.horizontalCenter

font.pointSize: 24; font.bold: true

}

}

Разбор по шагам

Импорт

Сперва нужно импортировать типы, которые нужны вам для этого примера. Большинство файлов QML импортируют встроенные типы QML (например, [Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html), [Image](https://doc.qt.io/qt-6/qml-qtquick-image.html), ...), которые поставляются с Qt, используя:

import QtQuick 2.0

Тип - Прямоугольник

Rectangle {

id: page

width: 320; height: 480

color: "lightgray"

Здесь объявляется корневой объект типа [Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) (Прямоугольник). Это один из основных строительных блоков, которые вы можете использовать для создания приложения на QML. Далее ему присваивается идентификатор, чтобы была возможность ссылаться на него в дальнейшем. В данном случае идентификатор имеет значение "page". Также устанавливаются свойства ширины, высоты и цвета. Тип [Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) содержит много других свойств (таких как *x* и *y*), но они будут инициированы со значениями по умолчанию.

Тип - Текст

Text {

id: helloText

text: "Hello world!"

y: 30

anchors.horizontalCenter: page.horizontalCenter

font.pointSize: 24; font.bold: true

}

Здесь добавляется тип [Text](https://doc.qt.io/qt-6/qml-qtquick-text.html) в качестве дочернего элемента корневого типа Rectangle, который отображает текст "Hello world!".

Свойство *y* используется для позиционирования текста по вертикали на расстоянии 30 пикселей от верхней части его родительского элемента.

Свойство *anchors.horizontalCenter* прикладывается к горизонтальному центру типа. В этом случае указывается, что тип Text должен быть горизонтально центрирован в элементе "page" (см. [Anchor-Based Layout](https://doc.qt.io/qt-6/qtquick-positioning-anchors.html)).

Размер *font.pointSize* и *font.bold* связаны со шрифтами и используют точечную нотацию.

Просмотр примера

Чтобы просмотреть то, что вы создали, запустите инструмент [qml](https://doc.qt.io/qt-6/qtquick-qml-runtime.html) (расположенный в каталоге bin) с вашим именем файла в качестве первого аргумента. Например, чтобы запустить предоставленный пример завершенной Главы 1 из места установки, вы должны ввести:

qml tutorials/helloworld/tutorial1.qml

2 - Компоненты QML

В этой главе описывается средство для изменения цвета текста.



На скриншоте выше изображен набор цветов, состоящий из шести ячеек с разными цветами. Чтобы избежать повторения одного и того же кода для каждой ячейки, создается новый компонент ячейки (компонент *Cell*). Компонент предоставляет способ определения нового типа, который можно повторно использовать в других файлах QML. Компонент QML подобен черному ящику и взаимодействует с внешним миром через свойства, сигналы и функции и обычно определяется в его собственном файле QML. (см. [Документацию по Component](https://doc.qt.io/qt-6/qml-qtqml-component.html)). Имя файла компонента всегда должно начинаться с заглавной буквы английского алфавита.

Ниже показан QML код для *Cell.qml*:

import QtQuick 2.0

Item {

id: container

property alias cellColor: rectangle.color

signal clicked(cellColor: color)

width: 40; height: 25

Rectangle {

id: rectangle

border.color: "white"

anchors.fill: parent

}

MouseArea {

anchors.fill: parent

onClicked: container.clicked(container.cellColor)

}

}

Разбор по шагам

Cell компонент

Item {

id: container

property alias cellColor: rectangle.color

signal clicked(cellColor: color)

width: 40; height: 25

Корневой тип нашего компонента - это элемент ([Item](https://doc.qt.io/qt-6/qml-qtquick-item.html)) с идентификатором (*ID*) равным *container*. Элемент (*Item*) является самым основным визуальным типом в QML и часто используется в качестве контейнера для других типов.

property alias cellColor: rectangle.color

Выше объявляется свойство *cellColor* (цвет ячейки). Данное свойство доступно извне компонента, это позволяет создавать экземпляры ячеек с разными цветами. Это свойство является просто псевдонимом существующего свойства - цвета прямоугольника, составляющего ячейку (см. [Привязку свойства](https://doc.qt.io/qt-6/qtqml-syntax-propertybinding.html)).

signal clicked(cellColor: color)

Также нужно, чтобы у нашего компонента был сигнал, который мы вызываем при нажатии (*clicked*) с параметром цвета ячейки типа цвета (*color*). Позже мы будем использовать данный сигнал для изменения цвета текста в основном файле QML.

Rectangle {

id: rectangle

border.color: "white"

anchors.fill: parent

}

Компонент ячейки в основном представляет собой цветной прямоугольник с идентификатором (*ID*) *rectangle*.

Свойство *anchors.fill* - это удобный способ задать размер визуального типа. В этом случае прямоугольник будет иметь тот же размер, что и его родительский (parent) элемент (см. [Макет на основе привязки](https://doc.qt.io/qt-6/qtquick-positioning-anchors.html)).

MouseArea {

anchors.fill: parent

onClicked: container.clicked(container.cellColor)

}

Чтобы изменить цвет текста при щелчке по ячейке, создается тип [MouseArea](https://doc.qt.io/qt-6/qml-qtquick-mousearea.html) (область мышки) с тем же размером, что и ее родительский элемент.

**Основной файл QML**

В основном файле QML используется ранее созданный компонент Cell для создания средства выбора цвета:

import QtQuick 2.0

Rectangle {

id: page

width: 320; height: 480

color: "lightgray"

Text {

id: helloText

text: "Hello world!"

y: 30

anchors.horizontalCenter: page.horizontalCenter

font.pointSize: 24; font.bold: true

}

Grid {

id: colorPicker

x: 4; anchors.bottom: page.bottom; anchors.bottomMargin: 4

rows: 2; columns: 3; spacing: 3

Cell { cellColor: "red"; onClicked: helloText.color = cellColor }

Cell { cellColor: "green"; onClicked: helloText.color = cellColor }

Cell { cellColor: "blue"; onClicked: helloText.color = cellColor }

Cell { cellColor: "yellow"; onClicked: helloText.color = cellColor }

Cell { cellColor: "steelblue"; onClicked: helloText.color = cellColor }

Cell { cellColor: "black"; onClicked: helloText.color = cellColor }

}

}

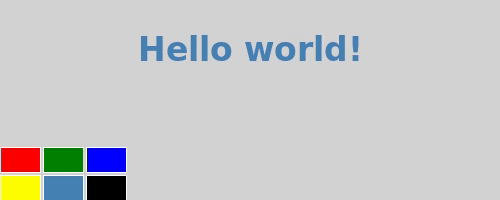
Создается средство выбора цвета, представляя собой сетку из 6 ячеек с разными цветами.

Когда срабатывает сигнал щелчка мыши (*clicked signal*) по ячейке, установится цвет текста равным цвету ячейки, переданному в качестве параметра *cellColor*. Можно реагировать на любой сигнал нашему компоненту с помощью свойства с именем *'onSignalName'* (см. [Атрибуты сигнала](https://doc.qt.io/qt-6/qtqml-syntax-objectattributes.html#signal-attributes)).

Глава 3 - Состояния и переходы

В данной главе предыдущий пример станет немного более динамичным, используя состояния и переходы.

Нужно, чтобы текст перемещался в нижнюю часть экрана, поворачивался и становился красным при нажатии.



Ниже приведен QML код:

import QtQuick 2.0

Rectangle {

id: page

width: 320; height: 480

color: "lightgray"

Text {

id: helloText

text: "Hello world!"

y: 30

anchors.horizontalCenter: page.horizontalCenter

font.pointSize: 24; font.bold: true

MouseArea { id: mouseArea; anchors.fill: parent }

states: State {

name: "down"; when: mouseArea.pressed == true

PropertyChanges {

helloText {

y: 160

rotation: 180

color: "red"

}

}

}

transitions: Transition {

from: ""; to: "down"; reversible: true

ParallelAnimation {

NumberAnimation { properties: "y,rotation"; duration: 500; easing.type: Easing.InOutQuad }

ColorAnimation { duration: 500 }

}

}

}

Grid {

id: colorPicker

x: 4; anchors.bottom: page.bottom; anchors.bottomMargin: 4

rows: 2; columns: 3; spacing: 3

Cell { cellColor: "red"; onClicked: helloText.color = cellColor }

Cell { cellColor: "green"; onClicked: helloText.color = cellColor }

Cell { cellColor: "blue"; onClicked: helloText.color = cellColor }

Cell { cellColor: "yellow"; onClicked: helloText.color = cellColor }

Cell { cellColor: "steelblue"; onClicked: helloText.color = cellColor }

Cell { cellColor: "black"; onClicked: helloText.color = cellColor }

}

}

Разбор по шагам

states: State {

name: "down"; when: mouseArea.pressed == true

PropertyChanges {

helloText {

y: 160

rotation: 180

color: "red"

}

}

}

Сначала создается новое состояние *down* для текстового типа. Это состояние будет активироваться при нажатии кнопки мышки и деактивироваться при ее отпускании.

Состояние *down* включает в себя набор изменений свойств из неявного дефолтного (default) состояния (элементы в том виде, в каком они были изначально определены в QML). В ручную (специально) устанавливаются свойства текста равными: положение по вертикали (*y*) равное 160, поворот - 180 и цвет - красный.

transitions: Transition {

from: ""; to: "down"; reversible: true

ParallelAnimation {

NumberAnimation { properties: "y,rotation"; duration: 500; easing.type: Easing.InOutQuad }

ColorAnimation { duration: 500 }

}

}

Чтобы текст появлялся внизу не мгновенно, а перемещался плавно - добавляется переход между нашими двумя состояниями.

"from" и "to" определяют состояния, между которыми будет выполняться переход. В этом случае происходит переход из состояния по умолчанию в состояние *down*.

Поскольку требуется, чтобы тот же переход выполнялся в обратном порядке при возврате из состояния *down* в дефолтное состояние, мы устанавливаем значение reversable равным true. Это эквивалентно записи двух переходов по отдельности.

Тип *ParallelAnimation* гарантирует, что два типа анимации (число и цвет) запускаются одновременно (параллельно). Также возможно бы запускать их один за другим, используя вместо этого *SequentialAnimation*.

Дополнительные сведения о состояниях и переходах см. в разделе [Состояния QtQuick](https://doc.qt.io/qt-6/qtquick-statesanimations-states.html), а также [Пример реализации состояний и переходов](https://doc.qt.io/qt-6/qtquick-animation-example.html#states).