

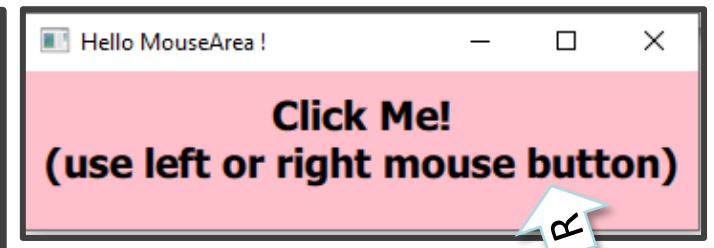
Пользовательский ВВОД

Механизмы взаимодействия с
пользователем

1. Область мыши

Для получения событий мыши в *QML* служат специальные элементы, которые называются ***MouseArea*** - область мыши. То, что они являются элементами, даёт возможность устанавливать их расположение и изменять размеры как у обычных элементов. По своей сути они представляют собой прямоугольные области, определяющие регионы, в которых должен осуществляться ввод информации от мыши.

В качестве примера рассмотрим прямоугольную область, которая реагирует на нажатие **левой и правой кнопки** мыши и отпускание кнопки мыши изменением цвета. (app: “**ClickMe**”)



2. Область мыши

```
import QtQuick 2.15
```

```
Rectangle {
```

```
    width: txt.width + 20; height: txt.height + 20
```

```
    color: "blue"
```

```
    Text {
```

```
        id: txt
```

```
        anchors.centerIn: parent
```

```
        text: "<h1>Click Me!<br>(use left or right mouse button)</h1>"
```

```
        horizontalAlignment: Text.AlignHCenter
```

```
    }
```

```
    ...
```

```
}
```



```
MouseArea {
```

```
    anchors.fill: parent
```

```
    acceptedButtons: Qt.LeftButton | Qt.RightButton
```

```
    onPressed: { // нажатие
```

```
        if (mouse.button == Qt.RightButton)
```

```
            parent.color = "pink"
```

```
        else
```

```
            parent.color = "gray"
```

```
    }
```

```
    onReleased: parent.color = "blue" // отпускание
```

```
}
```

События
мыши

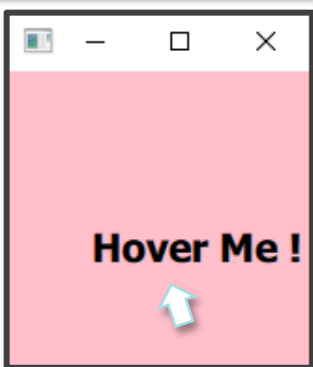
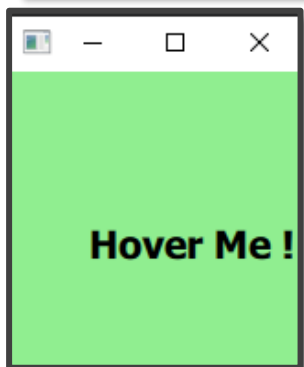
3. Обработка события мыши

```
import QtQuick 2.15
Rectangle {
    width: 200; height: 200
    color: mousearea.containsMouse ? "red" : "lightgreen"
    Text {
        anchors.centerIn: parent
        text: "<h1>Hover Me !</h1>"
    }
    MouseArea {
        id: mousearea
        anchors.fill: parent
        hoverEnabled: true
    }
}
```

Св-во позволяет
реагировать на
наведение
курсора



```
import QtQuick 2.15
Rectangle {
    width: 200; height: 200
    color: "lightgreen"
    Text {
        text: "<h1>Hover Me !</h1>"
        anchors.centerIn: parent
    }
    MouseArea {
        id: mousearea
        anchors.fill: parent
        hoverEnabled: true
        onEntered: parent.color = "red " // входение
        onExited: parent.color = "lightgreen" // выход
    }
}
```



4. Сигналы

Сигналы в *QML* - это просто события, которые прикреплены к свойствам с кодом для исполнения. В языке *QML* эти свойства называются обработчиками сигналов. Они имеют префикс **on**. Таким образом, **сигнал - это событие**, а **слот - это свойство с функцией**, которое выполняется по этому событию.

Сигнал и обработчик определяют в одном и том же *QML*- элементе.

```
Window {  
    width: 250; height: 250  
    visible: true  
    function setText(){  
        header.text = "Signal clicked"  
    }  
    Column {  
        padding: 5  
        Button{  
            text: "Click me"  
            onClicked: setText()  
        }  
        Text {  
            id: header  
            font.pixelSize: 18  
        }  
    }  
}
```

В стандартных элементах определены сигналы и обработчики - например, только что рассмотренный элемент **MouseArea** содержит свойства - обработчики сигналов - **onPressed**, **onReleased**, **onClicked**.

При желании и при необходимости можно добавлять в код свои собственные сигналы, а действия обработчика события можно вынести в отдельную функцию *JS*.

*Для добавления собственных сигналов существует ключевое слово *signal*.

5. Сигналы

```
import QtQuick 2.15
```

```
Rectangle {
```

```
width: 300
```

```
height: 150
```

```
signal sigMousePositionChanged(int x, int y)
```

```
onSigMousePositionChanged:
```

```
    txt.text = "<h1>X:" + x + "; Y:" + y + "</h1>"
```

```
Text {
```

```
id: txt
```

```
text: "<h1>Move the Mouse</h1>"
```

```
anchors.centerIn: parent
```

```
}
```

```
MouseArea {
```

```
anchors.fill: parent
```

```
hoverEnabled: true
```

```
onMouseXChanged: parent.sigMousePositionChanged(mouseX, mouseY)
```

```
onMouseYChanged: parent.sigMousePositionChanged(mouseX, mouseY)
```

```
}
```

```
}
```

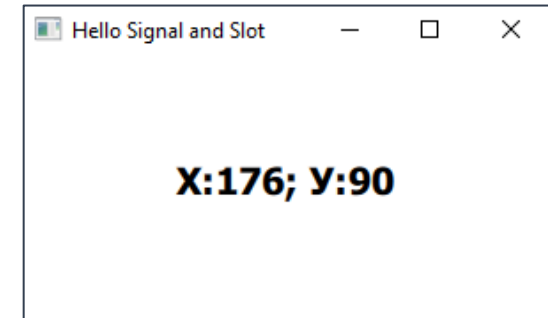
сигнал

Обработчик
сигнала

sig
slot

Обработчик
изменения св-ва
MouseArea

Высылка сигнала:
"mousePositionChanged"



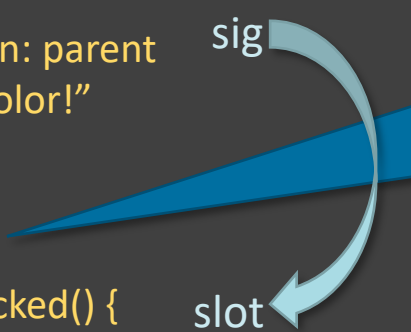
app "signals"

6. Сигналы

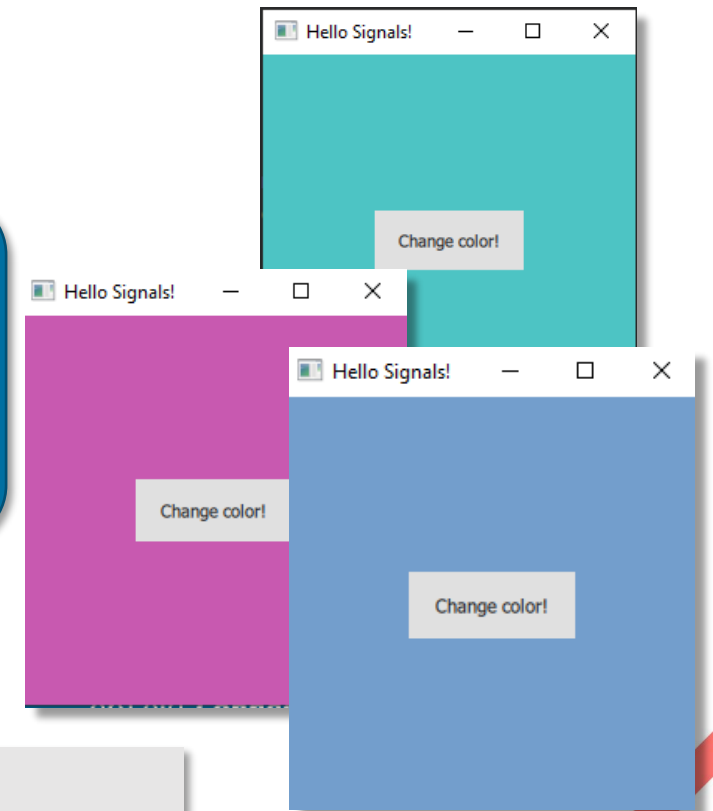
Иногда необходимо получать доступ к сигналу за пределами объекта, который его излучает. Для этого в *QtQuick* имеется тип **Connections** – для подключения к сигналам произвольных объектов. Объект *Connections* может получать любой сигнал от указанной цели (**target**).

```
import QtQuick
import QtQuick.Controls

Rectangle {
    id: rect
    width: 250; height: 250
    Button {
        id: button
        anchors.centerIn: parent
        text: "Change color!"
    }
    Connections {
        target: button
        function onClicked() {
            rect.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
        }
    }
}
```



Обработчик *onClicked* может быть установлен в объекте **Rectangle**, если поместить его в объект **Connections**, целью которого была кнопка **Button**



App:
"signals_study"

7. «Кнопка с сигналом»

```
import QtQuick 2.15
```

```
BorderImage {
```

```
property alias text: txt.text
```

```
signal clicked;
```

```
source: "qrc: /mybutton.png"
```

```
width: txt.width + 15
```

```
height: txt.height + 15
```

```
border { left: 15; top: 12; right: 15; bottom: 12 }
```

```
Text {
```

```
id: txt
```

```
color: "white"
```

```
anchors.centerIn: parent
```

```
}
```

```
MouseArea {
```

```
anchors.fill: parent
```

```
onClicked: parent.clicked();
```

```
onPressed: parent.source = "qrc:/mybuttonpressed.png"
```

```
onReleased: parent.source = "qrc:/mybutton.png"
```

```
}
```

```
}
```

“MyButton.qml”

Кнопка с сигналом ,
который будет
использоваться
извне

Испускание
сигнала
BorderImage

Обработчик
сигнала
MouseArea



*Элементы для повторного использования должны быть помещены в отдельные файлы. Имя файла является именем элемента. Код, приведенный в листинге находится в файле: “MyButton.qml”.

8. Использование «Кнопки с сигналом»

Использование элемента кнопки **MyButton**, ничем не отличается от использования любого другого стандартного элемента. С помощью свойства **text** мы присваиваем элементу кнопки начальный текст, а при нажатии кнопки в свойстве обработки сигнала **onClicked** присваиваем кнопке другой текст.

```
import QtQuick 2.15
Item {
    width: 150
    height: 100
    MyButton {
        anchors.centerIn: parent
        text: "Please, Click me !"
        onClicked: {
            text = "Clicked !"
        }
    }
}
```

Обработчик
сигнала
BorderImage

Так как связанные свойства тоже генерируют события, то сигналы можно практически всегда заменить свойствами. Разница в них следующая. **Сигналы** отправляются в одном направлении - от отправителя к получателю. Получатель при этом **не может изменить в высланном элементе принятые значения**. Со **свойством** же все обстоит иначе — **значения могут быть изменены, что также может привести к изменениям в поведении других элементов**, которые подсоединены к этим свойствам. Поэтому сигналы лучше использовать в случаях, например, когда взаимодействие должно осуществляться между автономными элементами. Свойства же более целесообразно использовать для связи элементов внутри элемента родителя.

9. «Кнопка со свойством»

```
import QtQuick 2.15
```

```
Window {
```

```
width: 150
```

```
height: 100
```

```
visible: true
```

```
MyButton1 {
```

```
anchors.centerIn: parent
```

```
text: "Please, Click me!"
```

```
onClickedChanged: {
```

```
text = "Clicked!"
```

```
    }  
}  
}
```

Использование
MyButton1

Обработка
изменения
св-ва

Разница в использовании элемента кнопки, заключается только в том, что свойство обработки события называется уже не **onClicked**, а **onClickedChanged**, так как теперь генерируется событие изменения значения свойства.

```
import QtQuick 2.15
```

```
BorderImage {
```

```
property alias text: txt.text
```

```
property bool clicked;
```

```
source: "qrc:/mybutton.png"
```

```
width: txt.width + 15
```

```
height: txt.height + 15
```

```
border {left: 15; top: 12; right: 15; bottom: 12}
```

```
Text {
```

```
id: txt
```

```
color: "white"
```

```
anchors.centerIn: parent
```

```
}
```

```
MouseArea {
```

```
anchors.fill: parent
```

```
onPressed: {
```

```
parent.source = "qrc:/mybuttonpressed.png"
```

```
parent.clicked = false
```

```
}
```

```
onReleased: {
```

```
parent.source = "qrc:/mybutton.png"
```

```
parent.clicked = true
```

```
}
```

```
}
```

```
}
```

"MyButton1.qml"

Св-во
Опред-е

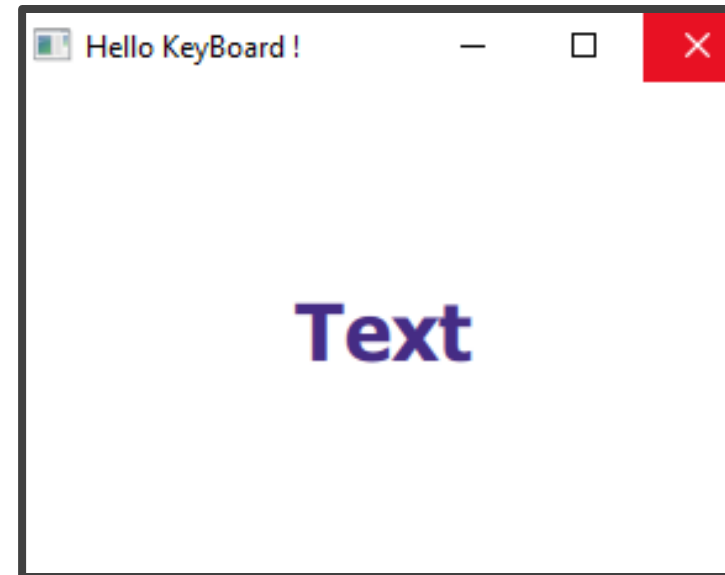
Св-во
задание
значения



10. Клавиатура

Ввод с клавиатуры можно обрабатывать двумя способами: с помощью элементов ***TextEdit*** или ***TextInput*** (аналогично *QTextEdit*, *QLineEdit*)

```
import QtQuick 2.15
Rectangle {
    width: 200; height: 100
    TextInput {
        anchors.centerIn: parent
        color: "red"
        text: "Text"
        font.pixelSize: 32
        font.bold: true
        focus: true
    }
}
```



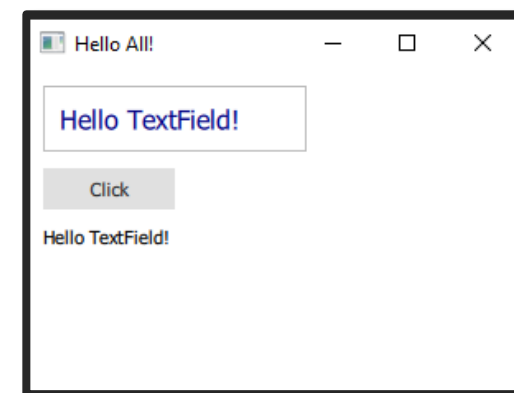
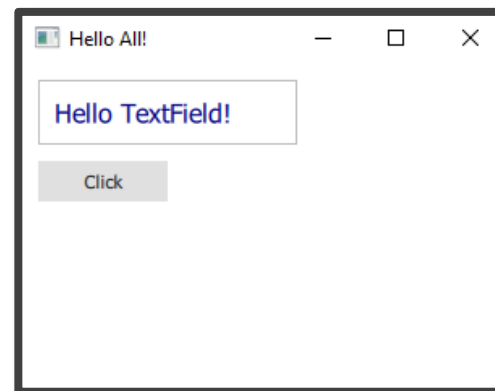
11. Элемент TextField

Поле текстового ввода **TextField**. Используется для создания одиночного текстового поля. Элемент из пакета **QtQuick.Controls**.

```
import QtQuick.Control 2.15
Window {
    width: 300; height: 200; visible: true
    title: qsTr("Hello All!")
    Column { // вертикальная компоновка
        anchors.fill: parent
        padding: 10; spacing: 10
        TextField {
            id: input; width: 160; height: 40
            placeholderText: "Введите текст"
            color: "navy"
            text: "Hello TextField!"
            font.family: Verdana; font.pixelSize: 16
            font.capitalization: Font.Capitalize
        }
        ...
    }
}
```

```
Text {
    id: output
}
```

```
Button {
    width: 80; height: 25
    text: "Click"
    onClicked: {
        output.text = input.text
    }
}
```



12. Элемент TextField

Основные методы:

- **clear()** – очистка поля ввода
- **copy** – копирует выделенный текст в буф. Обмена
- **cut()** - вырезает выделенный текст
- **deselect()** – отменяет выделение
- **getText()** – получает текст между определенными индексами
- **insert()**- вставляет текст на определённую позицию
- **paste()**- вставляет текст из буф. обмена
- **redo()**-восстанавливает последнее отмененное действие
- **remove()**-удаляет текст между индексами
- **select()**- выделяет текст между индексами
- **selectAll()**- выделяет весь текст
- **selectWord()** – выделяет слово ближайшее к курсору
- **undo()**- отменяет последнее действие

Установка отступов:

- **leftInsert** – отступ слева
- **topInsert** – отступ сверху
- **bottomInsert** – отступ снизу
- **rightInsert** – отступ справа

События ввода текста:

- **pressed** – при нажатии на поле ввода
- **released** – при уходе из места ввода
- **textEdited** – при каждом изменении введенных символов
- **accepted** – при нажатии *Enter* или *Backspace*
- **editingFinished** – при нажатии *Enter* или *Backspace* и потере фокуса

Режим ввода:

- **TextInput.Normal** – отображает текст как он введен в текстовое поле
- **TextInput.Password** – отображает символ маски, скрывающий введенный символ
- **TextInput.NoEcho** – ничего не делает
- **TextInput.PasswordEchoOnEdit** – во время ввода отображает текст как есть, а после - скрывает с помощью маски.

13. Определение маски ввода

Свойство элемента ***TextField – inputMask*** позволяет задать маску ввода. Маска ввода может применяться для автоматической валидации ввода. Для создания маски могут применяться следующие символы:

Символ	Значение
A	Обязательный алфавитный символ
a	Разрешенный, необязательный алфавитный символ
N	Обязательный алфавитно-цифровой символ
n	Разрешенный, необязательный алфавитно-цифровой символ
X	Обязательный символ, не являющийся пробелом
x	Разрешенный, необязательный символ не являющийся пробелом
9	Обязательный цифровой символ
0	Разрешенный необязательный цифровой символ
D	Обязательный цифровой символ больше 0
d	Разрешенный но необязательный цифровой символ, который больше 0

...

inputMask: "+9(999)-999-9999"

Вводить будем:

+_()- - -

...

inputMask: "+7(999)-999-9999"

Вводим будем:

+7()- - -

...

Чтобы ввести: +9()- - -

inputMask: "+\9(999)-999-9999"

* Установка шрифта у элемента TextField

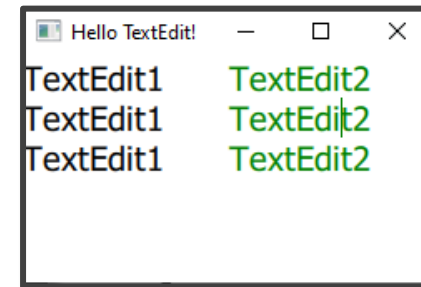
Для настройки шрифта применяется ряд свойств:

- **color:** устанавливает цвет текста
- **font.bold:** устанавливает выделение жирным при значении true
- **font.capitalization:** устанавливает режим перевода в верхний регистр. Возможные значения:
 - **Font.MixedCase:** регистр символов не изменяется
 - **Font.AllUppercase:** весь текст переводится в верхний регистр
 - **Font.AllLowercase:** весь текст переводится в нижний регистр
 - **Font.SmallCaps:** текст отображается прописными буквами
 - **Font.Capitalize:** первая буква каждого слова переводится в верхний регистр
- **font.family:** устанавливает применяемое семейство шрифтов
- **font.italic:** устанавливает выделение курсивом при значении true
- **font.letterSpacing:** устанавливает отступы между буквами
- **font.pixelSize:** устанавливает высоту шрифта в пикселях
- **font.pointSize:** устанавливает высоту шрифта в точках
- **font.strikeout:** устанавливает зачеркивание текста при значении true
- **font.styleName:** устанавливает имя стиля шрифта
- **font.underline:** устанавливает подчеркивание текста при значении true
- **font.weight:** устанавливает вес текста
- **font.wordSpacing:** устанавливает отступы между словами

14. Фокус

Фокус между элементами ввода может быть перемещен клавишами управления курсором и табуляцией. Присвоение фокуса работает следующим образом. Если содержится всего лишь один элемент ***TextInput***, то он получает фокус автоматически. Если же их больше одного, то пользователь может сам изменять фокус нажатиями мыши. Если необходимо установить фокус, то, нужно воспользоваться свойством ***focus***.

```
import QtQuick 2.15
Item {
    width: 200; height: 80
    TextEdit {
        anchors.left: parent.left
        anchors.right: parent.horizontalCenter
        anchors.top: parent.top
        anchors.bottom: parent.bottom
        text: "TextEdit1\nTextEdit1\nTextEdit1"
        font.pixelSize: 20
        color: focus ? "pink" : "black"
        focus: true
    }
    ...
}
```



```
TextEdit {
    anchors.left: parent.horizontalCenter
    anchors.right: parent.right
    anchors.top: parent.top
    anchors.bottom: parent.bottom
    text: "TextEdit2\nTextEdit2\nTextEdit2"
    font.pixelSize: 20
    color: focus ? "red" : "black"
}
```

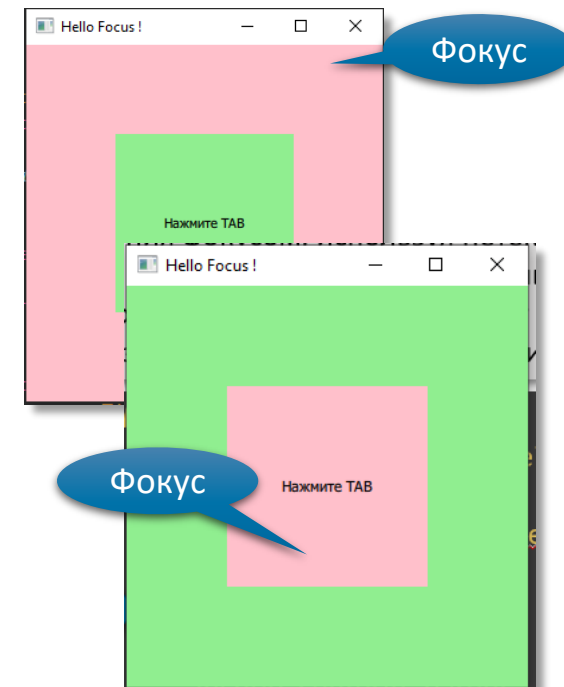

15. Пример работы с фокусом

Продemonстрируем возможность управления фокусом, используя нетекстовые элементы, так как они могут тоже иметь фокус. В следующем примере представлены два прямоугольника: один внешний, а другой внутренний. При получении фокуса прямоугольник изменяет свой цвет со светло-зеленого на красный. Изменение происходит при нажатии пользователем клавиши табуляции.

```
import QtQuick 2.15
```

```
Rectangle {  
    width: 300  
    height: 300  
    color: focus ? "pink" : "lightgreen"  
    KeyNavigation.tab: childrect  
    ...  
}
```

```
Rectangle {  
    id: childrect; width: 150; height: 150  
    anchors.centerIn: parent  
    color: focus ? "pink" : "lightgreen"  
    KeyNavigation.tab: parent  
    focus: true  
    Text {  
        anchors.centerIn: parent  
        text: «Нажмите TAB»  
    }  
}
```

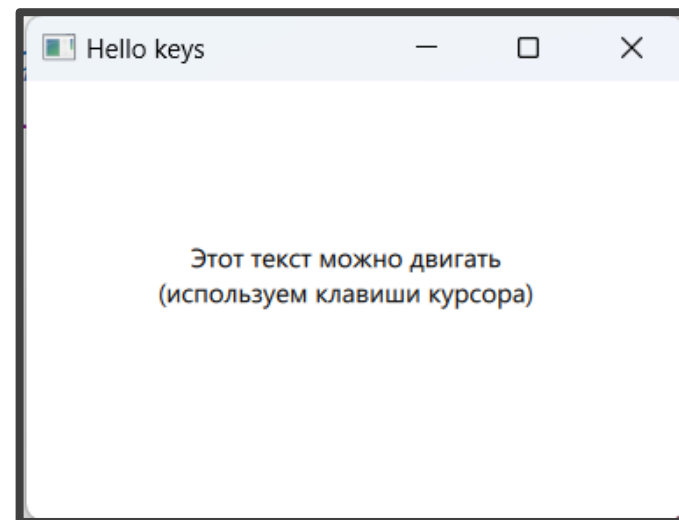


*Для управления фокусом мы могли бы вместо клавиши табуляции использовать, например, клавиши управления курсором, задействовав свойства **KeyNavigation.right**, **KeyNavigation.left**, **KeyNavigation.up**, **KeyNavigation.down**.

16. «Сырой» ввод

Часто нужно обеспечить возможность доступа на уровне событий клавиатуры с полной информацией о событии. Элементы такой возможностью не обладают, поэтому для этого применяется прикрепляемое свойство **Keys**.

```
import QtQuick 2.15
Rectangle {
    width: 300
    height: 200
    Text {
        x: 60 ; y: 60
        text: "Move this text<br>(use the cursor-keys) "
        horizontalAlignment: Text.AlignHCenter
        Keys.onLeftPressed: x -= 3
        Keys.onRightPressed: x += 3
        Keys.onDownPressed: y += 3
        Keys.onUpPressed: y -= 3
        focus: true
    }
}
```



Здесь используем элемент текста **Text**, позицию которого можно изменять при помощи клавиш управления курсором: (->, <-, up, down) .

17. «Сырой» ввод

Можно обойтись и одним обработчиком, а также контролировать все нажатия в одном свойстве **onPressed**, получая дополнительную информацию события (**event.key**) и сравнивая его значение с перечислениями клавиатуры Qt

```
Keys.onPressed: (event)=> {  
    if (event.key === Qt.Key_Left)  
        x -= 3;  
    else if (event.key === Qt.Key_Right)  
        x += 3;  
    else if (event.key === Qt.Key_Down)  
        y += 3;  
    else if (event.key === Qt.Key_Up)  
        y -= 3;  
}
```

```
else if (event.key === Qt.Key_Plus)  
    font.pixelSize++;  
else if (event.key === Qt.Key_Minus)  
    font.pixelSize--;
```

События клавиатуры при помощи **Keys.forwardTo** могут пересылаться и другим элементам для дальнейшей обработки, причем допускаются также и списки объектов.

```
Item {  
    ListView {  
        id: list1  
    }  
    Keys.forwardTo: [list1, list2]  
    focus: true  
}
```

Возможность увеличивать и уменьшать размер шрифта нажатием на клавиши<+> и <->.

18. MultyTouch

В *QML* за область региона, где будет осуществляться мультитач, отвечает элемент ***MultiPointTouchArea***. Этот элемент содержит в себе элементы обработки события касания ***TouchPoint*** - их ровно столько, сколько одновременных касаний мы намереваемся обрабатывать. По своей функциональной сути ***TouchPoint*** можно грубо сравнить с элементом обработки события мыши ***MouseArea***

Код приложения, способное обрабатывать до пяти одновременных прикосновений и отображать их места.

```
import QtQuick 2.15
Rectangle {
    width: 400
    height: 400
    color: "black"
    ...
}
```



```
MultiPointTouchArea {
    anchors.fill: parent
    minimumTouchPoints: 1
    maximumTouchPoints: 5
    touchPoints: [
        TouchPoint {}, TouchPoint {}, TouchPoint {},
        TouchPoint {}, TouchPoint {}
    ]
    Repeater {
        model: parent.touchPoints
        Rectangle {
            color: "white";
            x: modelData.x;
            y: modelData.y;
            width: 30; height: 30
            visible: modelData.pressed
        }
    }
}
```

Термин мультитач (Multi-touch) в переводе с английского языка означает множественное касание.

19. Некоторые свойства MultyTouch

Ниже приведены некоторые свойства элемента *TouchPoint*.

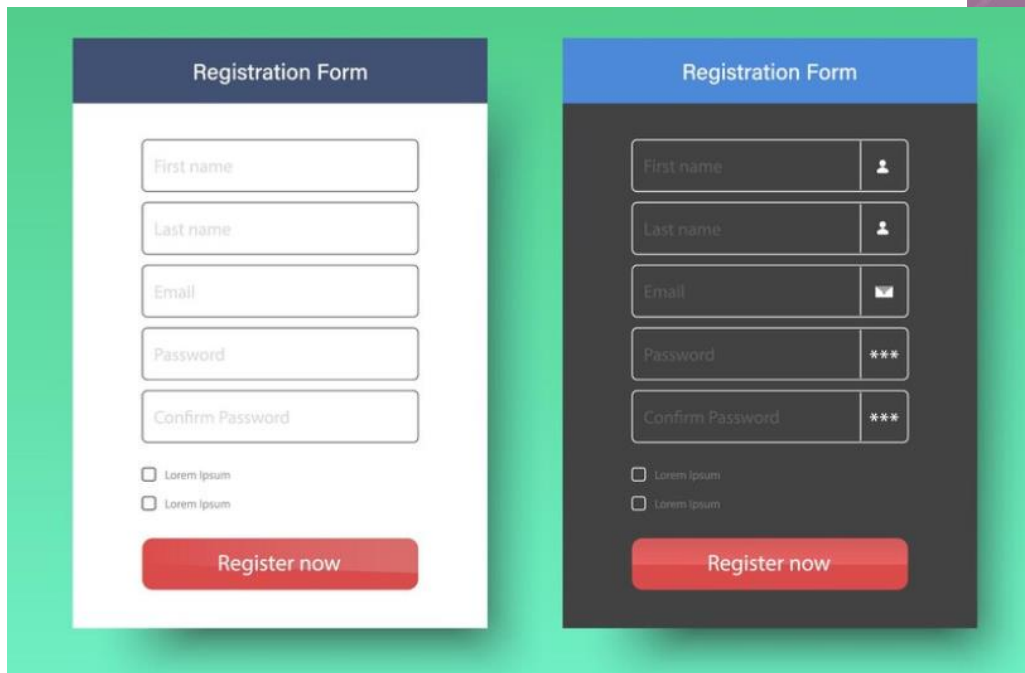
Имя свойства	Описание
pressed	При касании имеет значение true, в противном случае false
pressure	Сила нажатия (не все устройства предоставляют эту информацию)
previousX, previousY	Предыдущие координаты позиций касания
startX, startY	Начальные координаты позиций касания
x, y	Текущие координаты позиций касания

*Помните, что каждое из свойств в *QML* всегда обладает соответствующим методом обработки **on<ИмяСвойства>Changed**.

20. Домашка #10

Создать форму регистрации, используя возможности:

- Позиционирования элементов,
- Кастомизации элементов
- Графические эффекты



The image shows two registration forms side-by-side, each with a green border. The left form has a white background and a dark blue header. The right form has a dark gray background and a blue header. Both forms contain the same fields: First name, Last name, Email, Password, and Confirm Password. The right form also includes icons for each field: a person icon for First name, a person icon for Last name, an envelope icon for Email, and three asterisks for Password and Confirm Password. Both forms have a 'Register now' button at the bottom.

Registration Form

First name

Last name

Email

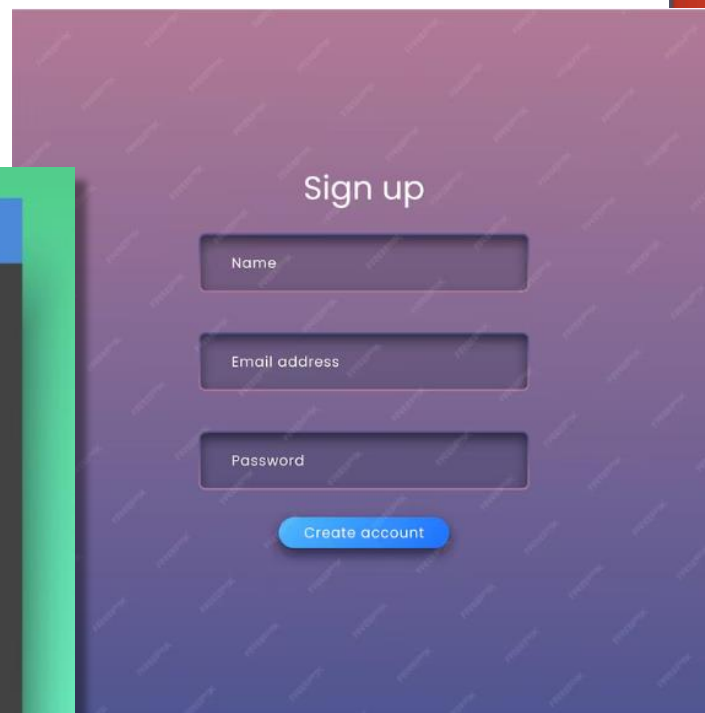
Password

Confirm Password

☐ Lorem Ipsum

☐ Lorem Ipsum

Register now



The image shows a registration form with a purple-to-blue gradient background. The form has a title 'Sign up' and four input fields: Name, Email address, Password, and Confirm Password. The Password and Confirm Password fields have three asterisks. There is a 'Create account' button at the bottom.

Sign up

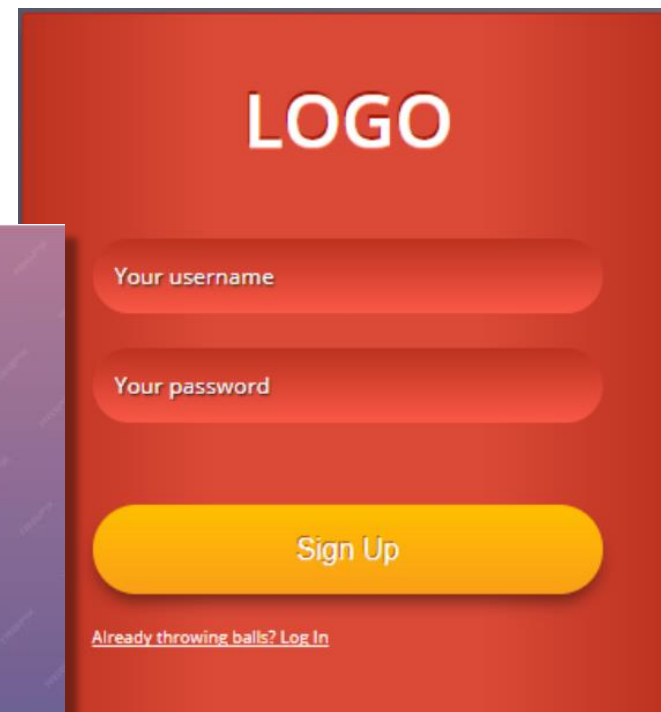
Name

Email address

Password

Confirm Password

Create account



The image shows a registration form with a red background. At the top is a white 'LOGO'. Below it are two input fields: 'Your username' and 'Your password'. There is a yellow 'Sign Up' button. At the bottom, there is a link: 'Already throwing balls? Log In'.

LOGO

Your username

Your password

Sign Up

[Already throwing balls? Log In](#)