

**QML – НОВЫЙ ПОДХОД К
ПОСТРОЕНИЮ GUI**

Введение

Подходы к построению десктопных приложений:

- Императивный
- Декларативный

QML - это декларативный язык, предназначенный для описания пользовательского интерфейса программы.

Сравнение:

- XAML
- CSS



Синтаксис

Дерево объектов со свойствами:

```
Rectangle {  
    id: canvas  
    width: 200  
    height: 200  
    color: "blue"  
    Image {  
        id: logo source: "pics/logo.png"  
        anchors.centerIn: parent  
        x: canvas.height / 5  
    }  
}
```

ОСНОВНЫЕ ТИПЫ ДАННЫХ

- [action](#)
- [bool](#)
- [color](#)
- [date](#)
- [enumeration](#)
- [font](#)
- [int](#)
- [list](#)
- [point](#)
- [real](#)
- [rect](#)
- [size](#)
- [string](#)
- [time](#)
- [url](#)
- [vector3d](#)

```
Item {  
    x: 10.5           // a 'real' property  
    state: "details" // a 'string' property  
    focus: true      // a 'bool' property  
}
```

Идентификаторы объектов

```
Row {  
  Text {  
    id: text1  
    text: "Hello World"  
  }  
  Text { text: text1.text }  
}
```

- Идентификатор элемента виден во всей области компонента, в котором этот элемент находится

Выражения

JavaScript – выражения могут быть использованы для назначения свойств элементов:

Пример 1:

```
Item {  
  width: 100 * 3  
  height: 50 + 22  
}
```

Пример 2:

```
Item {  
  width: 300  
  height: 300  
  Rectangle {  
    width: parent.width - 50  
    height: 100  
    color: "yellow"  
  }  
}
```

Соединения (Connections)

Создает подключение к QML-сигналу:

```
MouseArea {  
  id: area  
}  
...  
Connections {  
  target: area  
  onClicked: foo(...)  
}
```

Преимущества:

- для одного сигнала можно написать несколько соединений
- создание связей за пределами сферы отправителя сигнала
- подключение к источникам не нужно предопределять

Сигналы. Обработка сигналов

Обработка сигнала нажатия кнопки мыши:

```
Item {  
    width: 100;  
    height: 100  
  
    MouseArea {  
        anchors.fill: parent  
        onClicked: {  
            console.log("mouse button clicked")  
        }  
    }  
}
```

- Возможность создания собственных сигналов с последующей их обработкой

Состояния

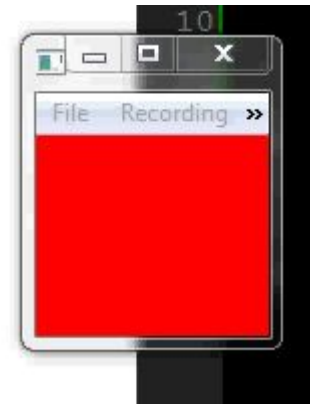
Это множество изменений по отношению к стандартной конфигурации элемента:

```
Rectangle {
  id: myRect
  width: 100; height: 100
  color: "black"
  MouseArea {
    id: mouseArea
    anchors.fill: parent
    onClicked: myRect.state == 'clicked' ?
      myRect.state = "" :
      myRect.state = 'clicked';
  }
  states: [
    State {
      name: "clicked"
      PropertyChanges { target: myRect; color: "red" }
    }
  ]
}
```

Поведение элементов

Поведение определяет анимации, которые должны применяться, когда изменяются определенные значения свойств элемента:

```
Rectangle {  
  id: rect  
  width: 100;  
  height: 100  
  color: "red"  
  
  Behavior on width {  
    NumberAnimation { duration: 1000 }  
  }  
  
  MouseArea {  
    anchors.fill: parent  
    onClicked: rect.width = 50  
  }  
}
```

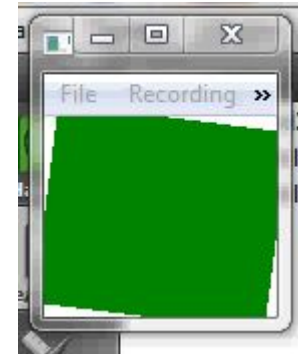


Анимация

Свойства и методы анимации привязаны непосредственно к элементам QML:

```
Rectangle {  
    width: 100  
    height: 100  
    color: "green"
```

```
    RotationAnimation on rotation {  
        loops: Animation.Infinite  
        from: 0  
        to: 360  
    }  
}
```



Свойства, присущие для всех видов анимации:

- alwaysRunToEnd : [bool](#)
- loops : [int](#)
- paused : [bool](#)
- running : [bool](#)

Работа с мышью

Для обработки сигналов, отправленных мышью служит элемент `MouseArea`:

```
Rectangle {  
  width: 100; height: 100  
  color: "green"
```

```
  MouseArea {  
    anchors.fill: parent  
    onClicked: { parent.color = 'red' }  
  }  
}
```



QML в C++ приложениях



Создание и передача свойства «цвет» из C++ приложения в QML:

```
// main.cpp
```

```
#include <QApplication>
#include <QDeclarativeView>
#include <QDeclarativeContext>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QDeclarativeView view;
    QDeclarativeContext *context = view.rootContext();
    context->setContextProperty("backgroundColor",
        QColor(Qt::yellow));
    view.setSource(QUrl::fromLocalFile("main.qml"));
    view.show();
    return app.exec();
}
```

```
// main.qml
```

```
Rectangle {
    width: 300
    height: 300
    color: backgroundColor

    Text {
        anchors.centerIn: parent
        text: "Hello Yellow World!"
    }
}
```