Creating a custom QML element with Qt

This snippet shows how to create a custom QML element.

Overview

Sometimes the elements provided by the Declarative module are not enough and a new element must be implemented.

In the following example we create a `Line` element which simply draws a line. The coordinates of the beginning and end of the line must be given. It is also possible to adjust the color, pen width, and smoothing of the line.

Preconditions

- Qt 4.7 or higher is installed on your platform.

Source

qmlapp.pro

```
QT       += core gui declarative

TARGET = qmlapp
TEMPLATE = app

SOURCES += main.cpp
HEADERS  += line.h
OTHER_FILES += ui.qml
```

The new element is derived from `QDeclarativeItem` which gives us the functionality of a QML `Item`, such as `x, y, width` and `height`. We declare the new properties with the macro `Q_PROPERTY` and state which methods to use when properties are read and updated. The `NOTIFY` feature is used to notify bound properties to update their value when the property here changes. The `paint` method handles the painting of our custom element. Finally, at the end of the header, we declare the class `Line` as a QML type.

line.h

```
#ifndef LINE_H
#define LINE_H

#include <QDeclarativeItem>
#include <QPainter>

class Line : public QDeclarativeItem
{
    Q_OBJECT
    Q_PROPERTY(int x1 READ x1 WRITE setX1 NOTIFY x1Changed);
    Q_PROPERTY(int y1 READ y1 WRITE setY1 NOTIFY y1Changed);
    Q_PROPERTY(int x2 READ x2 WRITE setX2 NOTIFY x2Changed);
    Q_PROPERTY(int y2 READ y2 WRITE setY2 NOTIFY y2Changed);
    Q_PROPERTY(QColor color READ color WRITE setColor NOTIFY colorChanged);
    Q_PROPERTY(int penWidth READ penWidth WRITE setPenWidth NOTIFY penWidthChanged);

public:
    Line(QDeclarativeItem *parent = 0) :
            QDeclarativeItem(parent), m_x1(0), m_y1(0), m_x2(0), m_y2(0),
            m_color(Qt::black), m_penWidth(1)
    {
        // Important, otherwise the paint method is never called
        setFlag(QGraphicsItem::ItemHasNoContents, false);
    }

    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
    {
        QPen pen(m_color, m_penWidth);
        painter->setPen(pen);

        if(smooth() == true) {
            painter->setRenderHint(QPainter::Antialiasing, true);
        }

        int x = qMin(m_x1, m_x2) - m_penWidth/2;
        int y = qMin(m_y1, m_y2) - m_penWidth/2;

        painter->drawLine(m_x1 - x, m_y1 - y, m_x2 - x, m_y2 - y);
    }

    // Get methods
    int x1() const { return m_x1; }
    int y1() const { return m_y1; }
    int x2() const { return m_x2; }
    int y2() const { return m_y2; }
    QColor color() const { return m_color; }
    int penWidth() const { return m_penWidth; }

    // Set methods
    void setX1(int x1) {
        if(m_x1 == x1) return;
        m_x1 = x1;
        updateSize();
        emit x1Changed();
        update();
    }
```


(C) Copyright Nokia 2013. All rights reserved.

```
    void setY1(int y1) {
        if(m_y1 == y1) return;
        m_y1 = y1;
        updateSize();
        emit y1Changed();
        update();
    }

    void setX2(int x2) {
        if(m_x2 == x2) return;
        m_x2 = x2;
        updateSize();
        emit x2Changed();
        update();
    }

    void setY2(int y2) {
        if(m_y2 == y2) return;
        m_y2 = y2;
        updateSize();
        emit x2Changed();
        update();
    }

    void setColor(const QColor &color) {
        if(m_color == color) return;
        m_color = color;
        emit colorChanged();
        update();
    }

    void setPenWidth(int newWidth) {
        if(m_penWidth == newWidth) return;
        m_penWidth = newWidth;
        updateSize();
        emit penWidthChanged();
        update();
    }

signals:
    void x1Changed();
    void y1Changed();
    void x2Changed();
    void y2Changed();
    void colorChanged();
    void penWidthChanged();

protected:
    void updateSize() {
        setX(qMin(m_x1, m_x2) - m_penWidth/2);
        setY(qMin(m_y1, m_y2) - m_penWidth/2);
        setWidth(qAbs(m_x2 - m_x1) + m_penWidth);
        setHeight(qAbs(m_y2 - m_y1) + m_penWidth);
    }

protected:
    int m_x1;
    int m_y1;
    int m_x2;
    int m_y2;
    QColor m_color;
    int m_penWidth;
};

QML_DECLARE_TYPE(Line)

#endif // LINE_H
```

In **main.cpp**, we register the C++ type `Line` in the QML system with the name `Line` to the library `CustomComponents` with version number 1.0.

main.cpp

```
#include "line.h"
#include <QApplication>
#include <QDeclarativeView>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    qmlRegisterType<Line>("CustomComponents", 1, 0, "Line");

    QDeclarativeView view;
    view.setSource(QUrl("./ui.qml"));
    view.setResizeMode(QDeclarativeView::SizeRootObjectToView);

#if defined(Q_WS_S60) || defined(Q_WS_MAEMO)
    view.showMaximized();
#else
    view.setGeometry(100,100, 800, 480);
    view.show();
#endif
    return a.exec();
}
```

In the QML document, we import the `CustomComponents` library which contains the `Line` element.

ui.qml

```
import CustomComponents 1.0
import Qt 4.7

Rectangle {
    property bool evenClick : false

    anchors.fill: parent; color: "lightsteelblue"

    Line {
        id: diagonalLine

        Behavior on x1 { NumberAnimation { duration: 1000 } }
```


(C) Copyright Nokia 2013. All rights reserved.

```
        Behavior on y1 { NumberAnimation { duration: 1000 } }
        Behavior on x2 { NumberAnimation { duration: 1000 } }
        Behavior on y2 { NumberAnimation { duration: 1000 } }

        x1: parent.x + 20; y1: parent.height / 2
        x2: parent.width - 20; y2: parent.height / 2
        color: "tomato"; penWidth: 3; smooth: true
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            if(evenClick) { diagonalLine.x1 = mouseX; diagonalLine.y1 = mouseY }
            else { diagonalLine.x2 = mouseX; diagonalLine.y2 = mouseY }
            evenClick = !evenClick
        }
    }

    Text {
        id: textX1Y1
        anchors.left: parent.left; anchors.top: parent.top
        text: "x1: " + diagonalLine.x1 + " y1: " + diagonalLine.y1
    }

    Text {
        anchors.left: parent.left; anchors.top: textX1Y1.bottom; anchors.topMargin: 10
        text: "x2: " + diagonalLine.x2 + " y2: " + diagonalLine.y2
    }
}
```

Postconditions

The custom QML element `Line` was created with Qt by deriving the class `QDeclarativeItem`, adding a few new properties and overwriting the paint method. The Qt class was registered as QML type and then used in a Qt Quick application.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.