Extending QStackedWidget for sliding page animations in Qt

Overview

This article describes how QStackedWidget can be used/subclassed to manage easily and comfortable multiple widgets, with nice sliding animations when changing from one widget to another. Widgets can be simple and small, or complete pages. This makes it easy, to manage multiple pages on a mobile device (Qt) and slide from one to another.

The new subclass `SlidingStackedWidget` can be used as easy as the original `QStackedWidget`, with very few additional API functions to pre-define the sliding animation behaviour.

This snippet can be self-signed.

Main code snippets are shown here. A complete small sample test project with the re-usable source of the new class `SlidingStackedWidget` is available.

Preconditions

This article was created and verified based on Qt 4.6.2 for Symbian, and tested on 5800 Xpress Music with Firmware Version v31.0.101. It should work on other firmware and phones, and it should basically also work on Maemo; all under Qt.

It was also tested on PC (native, not Symbian Emulator).

Header file

Class Declaration and Constructors

```
class SlidingStackedWidget : public QStackedWidget
{
        Q_OBJECT

public:
        //! This enumeration is used to define the animation direction
        enum t_direction {
                LEFT2RIGHT,
                RIGHT2LEFT,
                TOP2BOTTOM,
                BOTTOM2TOP,
                AUTOMATIC
        };

        //! The Constructor and Destructor
        SlidingStackedWidget(QWidget *parent);
        ~SlidingStackedWidget(void);
```

The main API methods

```
public slots:
        //! Some basic settings API
        void setSpeed(int speed);   //animation duration in milliseconds
        void setAnimation(enum QEasingCurve::Type animationtype); //check out the QEasingCurve documentation for different styles
        void setVerticalMode(bool vertical=true);
        void setWrap(bool wrap);    //wrapping is related to slideInNext/Prev;it defines the behaviour when reaching last/first page

        //! The Animation / Page Change API
        void slideInNext();
        void slideInPrev();
        void slideInIdx(int idx, enum t_direction direction=AUTOMATIC);
```

And finally, internal management variables and methods. Those are protected.

```
signals:
        //! this is used for internal purposes in the class engine
        void animationFinished(void);

protected slots:
        //! this is used for internal purposes in the class engine
        void animationDoneSlot(void);

protected:
        //! this is used for internal purposes in the class engine
        void slideInWgt(QWidget * widget, enum t_direction direction=AUTOMATIC);

        QWidget *m_mainwindow;

        int m_speed;
        enum QEasingCurve::Type m_animationtype;
```


(C) Copyright Nokia 2013. All rights reserved.

```
        bool m_vertical;
        int m_now;
        int m_next;
        bool m_wrap;
        QPoint m_pnow;
        bool m_active;

        QList<QWidget*> blockedPageList;
};
```

Source file

FIRST, the initialization in the constructor or the SlidingStackedWidget class.

```
SlidingStackedWidget::SlidingStackedWidget(QWidget *parent)
    : QStackedWidget(parent)
{
    if (parent!=0) {
            m_mainwindow=parent;
        }
        else {
            m_mainwindow=this;
            qDebug()<<"ATTENTION: untested mainwindow case !";
        }
        //parent should not be 0; at least not tested for any other case yet !!

        //Now, initialize some private variables with default values
        m_vertical=false;
        //setVerticalMode(true);
        m_speed=500;
        m_animationtype = QEasingCurve::OutBack;  //check out the QEasingCurve documentation for different styles
        m_now=0;
        m_next=0;
        m_wrap=false;
        m_pnow=QPoint(0,0);
        m_active=false;
}
```

NOW, the basic settings API methods:

```
void SlidingStackedWidget::setVerticalMode(bool vertical) {
        m_vertical=vertical;
}

void SlidingStackedWidget::setSpeed(int speed) {
        m_speed = speed;
}

void SlidingStackedWidget::setAnimation(enum QEasingCurve::Type animationtype) {
        m_animationtype = animationtype;
}

void SlidingStackedWidget::setWrap(bool wrap) {
        m_wrap=wrap;
}
```

Wrapping means, that if enabled and you are on the highest indexed widget, when calling the "next" widget, it wraps to the first widget. If disabled, there is no action.

Now we add three API methods to handle the selection of the visible sub widget that should slide in. Whenever this is called, the currently shown subwidget is smoothly shifted out.

```
void SlidingStackedWidget::slideInNext() {
        int now=currentIndex();
        if (m_wrap||(now<count()-1))
                // count is inherit from QStackedWidget
                slideInIdx(now+1);
}


void SlidingStackedWidget::slideInPrev() {
        int now=currentIndex();
        if (m_wrap||(now>0))
                slideInIdx(now-1);
}

void SlidingStackedWidget::slideInIdx(int idx, enum t_direction direction) {
        //int idx, t_direction direction=AUTOMATIC
        if (idx>count()-1) {
            direction=m_vertical ? TOP2BOTTOM : RIGHT2LEFT;
            idx=(idx)%count();
        }
        else if (idx<0) {
            direction= m_vertical ? BOTTOM2TOP: LEFT2RIGHT;
            idx=(idx+count())%count();
        }
        slideInWgt(widget ( idx ),direction);
        //widget() is a function inherited from QStackedWidget
}
```

And finally the engine of the SlidingStackedWidget class. It consists of

- slideInWgt method: setting up the newly to be shown sub-widget and starting the animation.
- animationDoneSlot method: called internally, to finalize/postprocess the animation procedure.

For details, please refer to the comments inside the code.

(C) Copyright Nokia 2013. All rights reserved.

```
void SlidingStackedWidget::slideInWgt(QWidget * newwidget, enum t_direction direction) {

        if (m_active) {
                return; // at the moment, do not allow re-entrance before an animation is completed.
                //other possibility may be to finish the previous animation abrupt, or
                //to revert the previous animation with a counter animation, before going ahead
                //or to revert the previous animation abrupt
                //and all those only, if the newwidget is not the same as that of the previous running animation.
        }
        else m_active=true;

        enum t_direction directionhint;
        int now=currentIndex();  //currentIndex() is a function inherited from QStackedWidget
        int next=indexOf(newwidget);
        if (now==next) {
                m_active=false;
                return;
        }
        else if (now<next){
                directionhint=m_vertical ? TOP2BOTTOM : RIGHT2LEFT;
        }
        else {
                directionhint=m_vertical ? BOTTOM2TOP : LEFT2RIGHT;
        }
        if (direction == AUTOMATIC) {
                direction=directionhint;
        }
        //NOW....
        //calculate the shifts

        int offsetx=frameRect().width(); //inherited from mother
        int offsety=frameRect().height();//inherited from mother

        //the following is important, to ensure that the new widget
        //has correct geometry information when sliding in first time
        widget(next)->setGeometry ( 0,  0, offsetx, offsety );

        if (direction==BOTTOM2TOP)  {
                offsetx=0;
                offsety=-offsety;
        }
        else if (direction==TOP2BOTTOM) {
                offsetx=0;
                //offsety=offsety;
        }
        else if (direction==RIGHT2LEFT) {
                offsetx=-offsetx;
                offsety=0;
        }
        else if (direction==LEFT2RIGHT) {
                //offsetx=offsetx;
                offsety=0;
        }
        //re-position the next widget outside/aside of the display area
        QPoint pnext=widget(next)->pos();
        QPoint pnow=widget(now)->pos();
        m_pnow=pnow;

        widget(next)->move(pnext.x()-offsetx,pnext.y()-offsety);
        //make it visible/show
        widget(next)->show();
        widget(next)->raise();

        //animate both, the now and next widget to the side, using animation framework
        QPropertyAnimation *animnow = new QPropertyAnimation(widget(now), "pos");

        animnow->setDuration(m_speed);
        animnow->setEasingCurve(m_animationtype);
        animnow->setStartValue(QPoint(pnow.x(), pnow.y()));
        animnow->setEndValue(QPoint(offsetx+pnow.x(), offsety+pnow.y()));
        QPropertyAnimation *animnext = new QPropertyAnimation(widget(next), "pos");
        animnext->setDuration(m_speed);
        animnext->setEasingCurve(m_animationtype);
        animnext->setStartValue(QPoint(-offsetx+pnext.x(), offsety+pnext.y()));
        animnext->setEndValue(QPoint(pnext.x(), pnext.y()));

        QParallelAnimationGroup *animgroup = new QParallelAnimationGroup;

        animgroup->addAnimation(animnow);
        animgroup->addAnimation(animnext);

        QObject::connect(animgroup, SIGNAL(finished()),this,SLOT(animationDoneSlot()));
        m_next=next;
        m_now=now;
        m_active=true;
        animgroup->start();

        //note; the rest is done via a connect from the animation ready;
        //animation->finished() provides a signal when animation is done;
        //so we connect this to some post processing slot,
        //that we implement here below in animationDoneSlot.
}


void SlidingStackedWidget::animationDoneSlot(void) {
    //when ready, call the QStackedWidget slot setCurrentIndex(int)
    setCurrentIndex(m_next);  //this function is inherit from QStackedWidget
    //then hide the outshifted widget now, and  (may be done already implicitely by QStackedWidget)
    widget(m_now)->hide();
    //then set the position of the outshifted widget now back to its original
    widget(m_now)->move(m_pnow);
    //so that the application could also still call the QStackedWidget original functions/slots for changings
    //widget(m_now)->update();
    //setCurrentIndex(m_next);  //this function is inherit from QStackedWidget
    m_active=false;
    emit animationFinished();
}
```

Note: for just using this SlidingStackedWidget class, there is no need to deal with the details of this engine.

Postconditions

(C) Copyright Nokia 2013. All rights reserved.

SlidingStackedWidget (QStackedWidget with sliding in/out animation) is a QWidget class, that looks very much like the QStackedWidget class, but effectively just manages the animation when changing from one sub widget to another. Usage:

- create any parent widget with any kind of layout.
- create an instance of this SlidingStackedWidget class.
- define few parameters, like animation time and curve/profile.
- define your sub-page widgets and add them to the SlidingStackedWidget in exactly the same way as it would be done for QStackedWidget
- assign the SlidingStackedWidget to the parent layout/widget.
- change from one sub-page widget to another by passing the index of the new - to be shown - page; very easy.

Example for constructor of a MainWindow class:

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    animTime=500;
    createGuiControlComponents();
    createSubSlidingWidgets();
    createSlidingStackedWidget();
    createMainLayout();
    createConnections();
}
```

Example for the creation of the createSlidingStackedWidget() method:

```
void MainWindow::createSlidingStackedWidget() {
    slidingStacked= new SlidingStackedWidget(this);
    slidingStacked->addWidget(slideWidget1);
    slidingStacked->addWidget(slideWidget2);
    slidingStacked->addWidget(slideWidget3);
    slidingStacked->addWidget(slideWidget4);
    slidingStacked->setSpeed(animTime);
}
```

Example for the SIGNAL/SLOT connection to change the page widget either by next/previous, or by index:

```
void MainWindow::createConnections() {
    QObject::connect(buttonNext,SIGNAL(pressed()),slidingStacked,SLOT(slideInNext()));
    QObject::connect(buttonPrev,SIGNAL(pressed()),slidingStacked,SLOT(slideInPrev()));
    QObject::connect(listAll,SIGNAL(currentIndexChanged(int)),slidingStacked,SLOT(slideInIdx(int)));
}
```

Complete Source Code of SlidingStackedWidget class Example and Test application

The complete source code of a class that implements the here described methodology can be found, together with a very simple demo application that makes use of it, on the following page:

Code Example for SlidingStackedWidget class in Qt

Further Work: Users/implementors may want to consider to extend this class with QTabWidget, or with some gesture recognition. But the class is not only useful for complex page type sub-widgets, but also for small widgets like sliding buttons.

References

- Animation Overview - Easing curves
- QPropertyAnimation
- QAnimationGroup
- QStackedWidget

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.