How to create flexible Portrait - Landscape rotation layout in Qt

Overview

This document describes, a comfortable and flexible way, how to manage two different layouts for a specific widget, where one is automatically used when the device is in portrait orientation, and the other when the device is in landscape orientation.

There is no limitation in the kind of QLayout derived layout implementation that can be used/assigned.

There is no need to duplicate subwidgets that shall be visible in our target widget in both views/orientation. They can simply be assigned to both, the portrait and the landscape layout.

The concept of this approach is, to create a subclass from QLayout, as a ProxyLayout class. Two independent layouts are made known to this ProxyLayout class. The ProxyLayout class implements the required re-implementations of a QLayout in the way, that it passes all those calls to the selected (portrait or landscape) layout, which was made known to it.

This snippet can be self-signed.

Main code snippets are shown here. A complete small sample test project with a fully re-usable ProxyLayout (portrait-landscape-layout) class 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 with Symbian rotation capability, and it should basically also work on Maemo; all under Qt.

Note, that this methodology is some kind of hack, and certain limitations may apply on the used sub-layouts. (HBox, VBox and grid layout at least should work.) Other than that, both - the assigned portrait and landscape layouts- must include the SAME client widgets. E.g. if some widget is only part of the portrait sub-layout, but not of the landscape sublayout, then this widget will be placed "uncontrolled" on the screen in landscape mode. This is certainly an area for further improvement work.

Header file

Class Declaration and Constructors

```
class ProxyLayout : public QLayout
{
public:
    ProxyLayout();
```

The main API methods

```
public:
    void addLayout(QLayout *layout);
    void removeLayout(QLayout *layout);
```

Re-implementation of QLayout Methods, as needed

```
public:
    void addItem(QLayoutItem *item);
    QSize sizeHint() const;
    QSize minimumSize() const;
    void setGeometry(const QRect &rect);
    bool hasHeightForWidth() const;
    int heightForWidth( int ) const;
    QLayoutItem * itemAt(int index ) const;
```


(C) Copyright Nokia 2014. All rights reserved.

```
    QLayoutItem * takeAt ( int index );
    int count() const;
```

The next methods are an event filter and helper internal methods. It is used to monitor resize events delivered to a widget the layout is set to.

```
 protected:
    bool eventFilter(QObject *obj, QEvent *event);

private:
    void activateLayout(int idx);
    void setupParentFilter();
```

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

```
 private:
    QList<QLayout*> m_layoutList;
    int m_currentLayoutIdx;
    bool filterInstalled;
};
```

Source file

FIRST, some basic methods that implement the quite universal engine of this portrait landscape manager.

```
void ProxyLayout::addLayout(QLayout *layout)
{
    m_layoutList.append(layout);
    layout->setParent(this);
    activateLayout(m_layoutList.size() - 1);
}

void ProxyLayout::removeLayout(QLayout *layout)
{
    int idx = layouts.indexOf(layout);
    if (idx > -1) {
        layouts.removeAt(idx);
        if (currentLayoutIdx > idx) {
            currentLayoutIdx--;
        } else if (currentLayoutIdx == idx) {
            if (currentLayoutIdx >= layouts.size())
                currentLayoutIdx = layouts.size() - 1;
            activateLayout(currentLayoutIdx);
        }
    }
}

void ProxyLayout::activateLayout(int idx)
{
    if(-1 < idx && idx < m_layoutList.size()) {
        if (idx != m_currentLayoutIdx) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
            m_currentLayoutIdx = idx;
            activate(); //inherited from QLayout
            update();   //inherited from QLayout
            QWidget *w = parentWidget();
            if(w)
                w->updateGeometry();
        }
    } else
        qDebug() << "ERROR: selecting unassigned MultLayoutClient. IDX=" << idx
                 << " numOfClients=" << m_layoutList.size();
}
```

NOW, implement all required methods of a QLayout Subclass, and point them to the actual activated layout

```
void ProxyLayout::addItem(QLayoutItem *item) {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return;
        }
        m_layoutList[getLayoutIdx()]->addItem(item);
}

QSize ProxyLayout::sizeHint() const {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return QSize(0,0);
        }
        return (m_layoutList[getLayoutIdx()]->sizeHint());
}

QSize ProxyLayout::minimumSize() const  {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return QSize(0,0);
        }
        return( m_layoutList[getLayoutIdx()]->minimumSize());
}

void ProxyLayout::setGeometry(const QRect &rect) {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return ;
        }
        m_layoutList[getLayoutIdx()]->setGeometry(rect);
}

bool ProxyLayout::hasHeightForWidth() const {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return false;
        }
        return (m_layoutList[getLayoutIdx()]->hasHeightForWidth());
}

int ProxyLayout::heightForWidth( int n) const {
        if (m_numOfClients<0) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return 0;
        }
        return( m_layoutList[getLayoutIdx()]->heightForWidth( n ));
}


QLayoutItem * ProxyLayout::itemAt(int index ) const  {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return 0;
        }
        return ( m_layoutList[getLayoutIdx()]->itemAt(index ));
}


QLayoutItem * ProxyLayout::takeAt ( int index ) {
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return 0;
        }
        return (m_layoutList[getLayoutIdx()]->takeAt(index ));
}

int ProxyLayout::count () const {
        ((ProxyLayout*)this)->setupParentFilter();
        if (m_numOfClients<0) {
                qDebug() << "ERROR: using MultLayout before assigning client.";
                return 0;
        }
        return( m_layoutList[getLayoutIdx()]->count());
}
```

And finally a event filter handling resize event

```
bool ProxyLayout::eventFilter(QObject *obj, QEvent *event)
{
    if (event->type() == QEvent::Resize) {
        QResizeEvent *resizeEvent = static_cast<QResizeEvent *>(event);
        QSize qs = resizeEvent->size();
        qreal diff, minDiff = 1000.0, ratio = (qreal)qs.width() / qs.height();
        if (ratio > 1.0)
            ratio = -1.0 / ratio;
        int ind = -1;
        QSize sz;
        for (int i= 0; i < m_layoutList.size(); ++i) {
            sz = m_layoutList.at(i)->sizeHint();
            diff = (qreal)sz.width()/sz.height();
            if (diff > 1.0)
                diff = -1.0 / diff;
            diff = qAbs(diff - ratio);
            if (diff < minDiff) {
                minDiff = diff;
                ind = i;
            }
        }
```


(C) Copyright Nokia 2014. All rights reserved.

```
        if (ind > -1)
            activateLayout(ind);
        return false;
    } else {
        // standard event processing
        return QObject::eventFilter(obj, event);
    }
}
```

Postconditions

ProxyLayout is a pseudo layout class, that looks like a layout class, but effectively just manages the switching between two assigned layouts, from which one is intended for portrait usage, and the other for landscape usage. Usage:

- create any layouts for portrait and landscape usage
- create an instance of this ProxyLayout.
- assign the pre-created layouts to the ProxyLayout instance by using the addLayout method.
- assing ProxyLayout to your widget.

Register event handler to the widget you are setting layout:

```
ProxyLayout *layout = new ProxyLayout();
layout->addLayout(...);
layout->addLayout(...);
QWidget *w = new QWidget();
w->setLayout(layout);
```

Example for constructor of a MainWindow class:

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
#ifdef Q_WS_MAEMO_5
    setAttribute(Qt::WA_Maemo5AutoOrientation, true);
#endif
    button1 = new QPushButton("One");
    button2 = new QPushButton("Two");
    button3 = new QPushButton("Three");
    button4 = new QPushButton("Four");

    QMenuBar *bar = win.menuBar();
    bar->addAction("Action 1");
    bar->addAction("Action 2");
    bar->addAction("Action 3");

    layoutPortrait = new QVBoxLayout;
    layoutPortrait->addWidget(button1);
    layoutPortrait->addWidget(button2);
    layoutPortrait->addWidget(button3);
    layoutPortrait->addWidget(button4);

    layoutLandscape = new QHBoxLayout;
    layoutLandscape->addWidget(button1);
    layoutLandscape->addWidget(button2);
    layoutLandscape->addWidget(button3);
    layoutLandscape->addWidget(button4);
```


(C) Copyright Nokia 2014. All rights reserved.

```
    autoLayout = new ProxyLayout();
    autoLayout->addLayout(layoutPortrait);
    autoLayout->addLayout(layoutLandscape);

    centralWidget = new QWidget();
    centralWidget->setLayout(autoLayout);
    setCentralWidget(centralWidget);
}
```

Complete Source Code of ProxyLayout class Example and Test application

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

- Code Example for Portrait / Landscape Autorotate Layout in Qt

References

The idea for this was taken up from some very old Qt forum discussion, where unfortunately no code was published (to my knowledge): http://lists.trolltech.com/qt-interest/2004-01/thread00746-0.html

Other related references:

- Archived:Listening for screen orientation changes in Qt
- Archived:Dynamic Layout handling with QWidget
- How to switch between horizontal and vertical layout in Qt

(C) Copyright Nokia 2014. All rights reserved.