Archived:Customising QTreeView

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Qt Quick should be used for all UI development on mobile devices. The approach described in this article (using C++ for the Qt app UI) is deprecated.

Overview

This article explains how to customise the Qt tree view widget (`QTreeView`).

Description

The `QTreeView` can be customised by overriding the virtual methods like `drawRow()` and `drawBranches()`. This article shows how to customise the tree view to show an image in the right-hand side of every tree view item.

Solution

A custom class `QCustomTreeView` is created by inheriting from `QTreeView`

```
class QCustomTreeView : public QTreeView
{
    Q_OBJECT
public:
    QCustomTreeView( QWidget *parent = 0 );
    ~QCustomTreeView();
protected:
    // QTreeView's virtual member
    void drawRow( QPainter *painter,
                  const QStyleOptionViewItem &options,
                  const QModelIndex &index ) const;
    // We can also override drawBranches() member
    // void drawBranches( QPainter *painter,
                          const QRect &rect,
                          const QModelIndex &index ) const;
};
```

QCustomTreeView class definition

```
QCustomTreeView::QCustomTreeView( QWidget *parent )
: QTreeView( parent )
{
}

// Overriding the QTreeView's drawRow() member
void QCustomTreeView::drawRow( QPainter *painter,
                               const QStyleOptionViewItem &options,
                               const QModelIndex &index ) const
{
  // Calling the base class method for drawing the normal
  // parts of the item of the QTreeView.
  QTreeView::drawRow( painter, options, index );

  // Calculate a rect for the image on the right side of the item
  // We can retrieve the parent and child index of the model index
  // QModelIndex parent(index.parent());
  // QModelIndex child(index.child(0,0));
  // Use your own logic to decide for which items the image on the
  // right side should be drawn
  if(index.row() == 0) {
    QImage image;
    image.load("c:\\data\\red.bmp");
    image.scaledToHeight(options.rect.height());
    image.scaledToWidth(15);

    // Drawing the image on the right side of the item
   painter->drawImage( options.rect.topRight().x() - 50,
                       options.rect.topRight().y(), image );
  }
  if(index.row() == 1) {
    QImage image;
    image.load("c:\\data\\blue.bmp");
    image.scaledToHeight(options.rect.height());
    image.scaledToWidth(15);

    // Drawing the image on the right side of the item
    painter->drawImage( options.rect.topRight().x() - 50,
                        options.rect.topRight().y(), image );
  }
}
```

Using QCustomTreeView

```
  // Create an instance of the custom tree view
  QCustomTreeView* customTree;
  customTree = new QCustomTreeView( this );

  // Populate the model for tree view
```


(C) Copyright Nokia 2013. All rights reserved.

```
    QStandardItemModel* treeModel = new QStandardItemModel( this );
    QStandardItem* item1 = new QStandardItem( QIcon("c:\\data\\610x.bmp"), "Item1" );
    QStandardItem* item2 = new QStandardItem( QIcon("c:\\data\\Nokia-logo.bmp"), "SubItem1" );
    QStandardItem* item3 = new QStandardItem( QIcon("c:\\data\\5.bmp"), "Item2" );
    QStandardItem* item4 = new QStandardItem( "SubItem2" );
    QStandardItem* item5 = new QStandardItem( "SubItem3" );
    item1->appendRow( item2 );
    item1->appendRow( item4 );
    item1->appendRow( item5 );
    treeModel->appendRow( item1 );
    treeModel->appendRow( item3 );

    customTree->setModel( treeModel );

    customTree->setHeaderHidden( true );
    customTree->setAnimated( true );
    customTree->setGeometry( QRect( 0, 0, 400, 500 ) );
```

Sample application

File:CustomTreeView.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.