Simple QML EBook Reader

This article presents a way to split and render a single HTML page (EBook). Instead of using the QtWebkit, the example uses QTextDocument painter to render the webpage.

Introduction

02 Oct 2011

With this article I'm going to show you a way to split and render a single HTML page (EBook) taken from Gutember.org.

The present example won't make use of webkit, but we will see how to use directly QTextDocument painter to render the HTML page. To browse the book pages I used the sliding pages example that can be found here.

The final result is shown in the following video: The media player is loading...

How does it work?

QML Part

The sliding pages are ListView items that show simply a page number and an image with the rendered text. The ListView element has been used because it's convenient. Indeed it keeps in memory only few pages at time, reducing the application memory footprint that for this kind of application can be really big.

```
Rectangle {
  anchors.fill: parent
  anchors.margins: 15

  // rendered text
  Image {
    source: modelData
  }

  // Page number
  Text {
    anchors.bottom: parent.bottom
    anchors.horizontalCenter: parent.horizontalCenter
    color: "grey"
    text: index
  }
}
```

As you can see the image source is set to "modelData". That's a QML keyword that contains the path (a string) to the image to be loaded.

```
ListView {
  id: list
  anchors.fill: parent
  model: dataModel
  delegate: myPageDelegate
  orientation: ListView.Horizontal
  snapMode: ListView.SnapToItem
}
```

So dataModel is basically a string list. It's created in the main.cpp file and it contains string like "image://pages/docPAGE_NUMBER" that are associated to an image; AS told briefly before, those images are loaded in memory only for the shown pages and the pages close to the shown one. So for this example the ListView keeps in memory only 3 pages at the time.

The C++ code

Let's take a look at the C++ implementation. Here is the main.cpp file:

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

    QmlApplicationViewer viewer;

    pageProvider *pages = new pageProvider(QSize(PAGEWIDTH, PAGEHEIGHT));
    viewer.engine()->addImageProvider(QLatin1String("pages"), pages);


    QStringList pageIDs;
    // Add book pages
    for (int i=0; i < pages->count(); i++){
        QString pageID = "image://pages/doc";
        pageID += QString::number(i);
        pageIDs << pageID;
    }

    QDeclarativeContext *ctxt = viewer.rootContext();
    ctxt->setContextProperty("dataModel", QVariant::fromValue(pageIDs));


    //Load QML
    viewer.setOrientation(QmlApplicationViewer::ScreenOrientationLockPortrait);
    viewer.setMainQmlFile(QLatin1String("qml/SlidingPages/main.qml"));
```


(C) Copyright Nokia 2013. All rights reserved.

```
    viewer.showExpanded();

    return app.exec();
}
```

The image provider is here defined and the engine will use it to load image in the "image://pages" path. The core of this application lives here, in the image provider. The code shown below in fact has to split the single HTML page in several pieces (pages).

```
#ifndef PAGEPROVIDER_H
#define PAGEPROVIDER_H

#include <QDeclarativeImageProvider>
#include <QObject>

class QTextDocument;
class pageProvider : public QDeclarativeImageProvider
{

public:
    explicit pageProvider(QSize page);
    virtual ~pageProvider();
    int count() const;
    QPixmap requestPixmap(const QString &id, QSize *size, const QSize &requestedSize);

private:
    QTextDocument *doc;
    QSize pageSize;
};

#endif // PAGEPROVIDER_H
```

pageProvider is a QDeclarativeImageProvider subclass (Please note that it's not a QObject!) that the QML engine uses to get the required images as it is a "kind of" directory. The core of this application is here and it makes use of the QTextDocument ability to render text. In this document I used an HTML code for simplicity, but actually QTextDocument can load other kind of file type by using plugins. If interested in that topic, you maybe want to take a look at the oKular source code available in the KDE SVN repository.

```
#include "pageprovider.h"
#include <QDebug>
#include <QFile>
#include <QTextDocument>
#include <QAbstractTextDocumentLayout>
#include <QPainter>

pageProvider::pageProvider(QSize size) :
    QDeclarativeImageProvider(QDeclarativeImageProvider::Pixmap),
    pageSize(size), doc(new QTextDocument)
{
    // Load the BOOK into the Text Document
    QFile file(":/1268-h.htm");
    if (!file.open(QIODevice::ReadOnly)) {
        qWarning("Unable to open file");
    }
    QByteArray bookData = file.readAll();
    doc->setHtml(bookData);

    // Split the document in several pages.
    doc->setPageSize(size);
}

pageProvider::~pageProvider(){
    delete doc;
}

int pageProvider::count() const{
    doc->pageCount();
}


QPixmap pageProvider::requestPixmap(const QString &id, QSize *size, const QSize &requestedSize){
    Q_UNUSED(size);
    Q_UNUSED(requestedSize);

    qDebug() << "requestPixmap" << id;

    // Does not render pages which don't belong to the document.
    // The Developer could add new pages to the listview without
    // changing the order of the pages
    QString idCopy = id;
    if (!id.startsWith("doc"))
        return QPixmap();
    idCopy.remove("doc");

    if (!size->isEmpty())
        qWarning() << "Size is not used in this code";

    int pageNumber = idCopy.toInt();
    int pageHeight = pageSize.height();
    int pageWidth = pageSize.width();

    QPixmap page(pageSize);

    QPainter painter(&page);
    painter.fillRect(QRectF(0, 0, pageWidth, pageHeight), Qt::white);
    painter.translate(0, -pageHeight * pageNumber);

    QAbstractTextDocumentLayout::PaintContext ctx;
    ctx.palette.setColor(QPalette::Text, Qt::black);
    ctx.clip = QRectF(0, pageHeight * pageNumber, pageWidth, pageHeight);

    QAbstractTextDocumentLayout *docLayout = doc->documentLayout();

    if (!docLayout)
        painter.drawText(ctx.clip, "ERROR: Unable the render the page!");
    else
        docLayout->draw(&painter, ctx);

    return page;
}
```

Above it's shown the C++ implementation of the image provider. Here "requestPixmap" is the most important function. It's called by the engine to render

(C) Copyright Nokia 2013. All rights reserved.

the page when the book page is shown. In this example "size" and "requestSize" arguments of that function, have not used since the page size does not change. Once that image has been rendered, the QML engine takes cache them. So requestPixmap is usually called just one time. That's important because rendering involves a lot of CPU power and can slow down the application.

Conclusion

QTextDocument is a powerful Qt tool that permits developers to handle documents in the right way. It can read several file type thanks to it's modular architecture. The shown example can be improved and used to create an ebook reader that can be sold through OVI Store.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.