Get thumbnail embedded in a JPG image using Qt

This code example shows how to extract an embedded thumbnail data stored within a JPEG file, using Qt C++.

Note: This is an entry in the PureView Imaging Competition 2012Q2

Overview

Pictures taken using mobile devices often embed a thumbnail within the JPEG image meta-information. If a pre-scaled thumbnail is required, extracting this image is *very* much faster than loading the full size image and scaling down for different methods of display.

This article provides a simple class for extracting the thumbnail and returning it as a QImage (or as a an empty image if no thumnail is present). The class which accepts either a QIODevice or a QString containing the file location of an image.

Header file

```
#ifndef __JPEG_THUMBNAILFETCHER__
#define __JPEG_THUMBNAILFETCHER__

#include <QIODevice>
#include <QImage>

class JpegThumbnailFetcher {
public:
    JpegThumbnailFetcher() {}
    ~JpegThumbnailFetcher() {}

    static QImage fetchThumbnail(QIODevice &jpegFile);
    static QImage fetchThumbnail(QString filePath);

protected:
    static bool readWord( QIODevice &sdevice, unsigned short *target, bool invert=true );
    static bool exifScanloop( QIODevice &jpegFile, unsigned int &tnOffset, unsigned int &tnLength );
};

#endif
```

Source file

```
#include <QFile>
#include <QBuffer>
#include <QByteArray>
#include <QImageReader.h>
#include "jpegThumbnailFetcher.h"

    // Exif defines
#define JPEG_SOI 0xffd8
#define JPEG_SOS 0xffda
#define JPEG_EOI 0xffd9
#define JPEG_APP1 0xffe1

/**
 *
 * Read a word from a stream either inverted or not.
 *
 */
bool JpegThumbnailFetcher::readWord( QIODevice &sdevice, unsigned short *target, bool invert ) {
    unsigned short t;
    if (sdevice.read((char*)&t, 2) != 2) return false;
    if (invert)
        *target = ((t&255) << 8) | ((t>>8)&255);
    else
        *target = t;
    return true;
}


/**
 *
 * Scans though exif-chunks, finds the app1-chunk and processes it.
 *
 */
bool JpegThumbnailFetcher::exifScanloop( QIODevice &jpegFile, unsigned int &tnOffset, unsigned int &tnLength ) {
    // LOOP THROUGH TAGS
    while (1) {
        unsigned short tagid, tagLength;
        if (!readWord( jpegFile, &tagid )) return 0;
        if (tagid == JPEG_EOI || tagid == JPEG_SOS) {
                // Data ends
            break;
        }
        if (!readWord( jpegFile, &tagLength )) return 0;

        if (tagid == JPEG_APP1) {
            char str[6];
            jpegFile.read(str,6 );

            // Store the current position for offset calculation
```


(C) Copyright Nokia 2013. All rights reserved.

```
            int basepos = jpegFile.pos();

            // read tiff - header
            unsigned short tifhead[2];
            for (int h=0; h<2; h++) {
                if (!readWord(jpegFile, &tifhead[h])) return false;
            }
            if (tifhead[0] != 0x4949) {
                //qDebug() << "invalid byte order";
                return false;
            }

            while (1) {
                unsigned int offset;
                jpegFile.read( (char*)&offset, 4);
                if (offset==0) break;
                jpegFile.seek( basepos + offset );

                unsigned short fields;
                if (!readWord(jpegFile, &fields, false)) return false;
                while (fields>0) {
                    char ifdentry[12];
                    jpegFile.read( ifdentry, 12 );
                    unsigned short tagnumber = (((unsigned short)ifdentry[0]) | (unsigned short)ifdentry[1]<<8);
                            // Offset of the thumbnaildata
                    if (tagnumber == 0x0201) {
                        memcpy( &tnOffset, ifdentry+8, 4 );
                        tnOffset += basepos;

                    } else  // Length of the thumbnaildata
                        if (tagnumber == 0x0202) {
                            memcpy( &tnLength, ifdentry+8, 4 );
                        };
                    fields--;
                    if (tnOffset != 0 && tnLength!=0) return true;
                }
            }
            return false;
        }
        jpegFile.seek( jpegFile.pos() + tagLength-2 );
    }
    return false;
}


QImage JpegThumbnailFetcher::fetchThumbnail(QIODevice &jpegFile) {
    QImage empty;
    if (!jpegFile.open( QIODevice::ReadOnly )) return empty;
    unsigned short jpegId;
    if (!readWord( jpegFile, &jpegId ))return empty;
    if (jpegId!= JPEG_SOI) return empty;          // JPEG SOI must be here

    unsigned int tnOffset = 0;
    unsigned int tnLength = 0;
    if (exifScanloop( jpegFile, tnOffset, tnLength)) {
            // Goto the thumbnail offset in the file
        jpegFile.seek( tnOffset );
            // Use image reader to decode jpeg-encoded thumbnail
        QByteArray tnArray = jpegFile.read( tnLength );
        QBuffer buf( &tnArray, 0 );
        QImageReader reader(&buf);
        reader.setAutoDetectImageFormat( false );
        reader.setFormat("jpg");
        return reader.read();
    }
    return empty;
}

QImage JpegThumbnailFetcher::fetchThumbnail(QString filePath) {
    QFile jpegFile(filePath);
    return fetchThumbnail( jpegFile );
}
```

Deploying a thumbnail

It's quite straightforward

```
JpegThumbnailFetcher fetcher;
QImage thumbNail = fetcher.fetchThumbnail("some/imagefile.jpg");
```

Summary

Extracting thumbnails from the original JPEG is very fast compared to downscaling the original image. Since the thumbnail is an actual JPEG image inside the large one, the performance difference is the same as it would be when loading this smaller image directly. For example, if you are trying to load an image with resolution 2400x2400 you are decoding 5.76 MPixels. If this JPEG contains a thumbnail with 128x128 and you use that instead, you only have to load 0,016 MPixels which is 360 times smaller than the large one. In this example case, the JPEG decoding process is **36 000%** more effective. Obviously, there are other issues also related of loading the image: File-access, acquiring the thumbnail pointer, etc. Those take some time as well.

So, if you don't need the full resolution images, this method can really give your application a nice performance boost.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.