Get EXIF GPS data to use it in QML

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

This article explains how to get JPG images EXIF GPS coordinates in Symbian to use in QML

Introduction

QML nor Qt does not offer way to get EXIF data of image. In Qt you can use additional libraries in Meego but no need to port those for Symbian, there is already existing library in Symbian. Eventhough this is tested with Symbian Belle, it may and should work with earlier versions too (without QML part).

It is possible to access other Exif data like thumbnail, descriptions and so on. See references for more information.

Full source to example can be found from here: http://projects.developer.nokia.com/EXifExample

Plugin

Symbian offers library to get EXIF data from JPG image but we're implementing QML item with Qt code, extend this chain with Symbian code using PIMPL pattern to access where Qt cannot.

Qt with Symbian

Rather than explaining code mixing, this document focuses to usage of Symbian library ExifLib. This library comes with QtSDK and Symbian SDK's, so there is no need to download any extra packages or libraries.

More info about using Symbian code in Qt and description of PIMPL pattern can be found here:
http://www.developer.nokia.comhttp://www.developer.nokia.com/Community/Wiki/Using_Qt_and_Symbian_C%2B%2B_Together

Implementation

Pro file

Most important in .pro file is to include Symbian ExifLib-library with LIBS option. It doesn't need any extra capabilities.

```
symbian {
    TARGET.UID3 = 0xE40DBC5A
    LIBS+=-lExifLib
    LIBS+=-lefsrv
    TARGET.CAPABILITY = LocalServices \
        NetworkServices \
        ReadUserData \
        UserEnvironment \
        WriteUserData
}
```

Main item

Create QDeclarativeItem derived item, in the example is also created function which takes image URL as argument.

Also properties for latitude and longitude coordinates exists, although when coordinates change this example will launch Maps to navigate coordinate location.

```
class ExifItem : public QDeclarativeItem
{
    Q_OBJECT
    Q_PROPERTY(qreal longitude READ longitude NOTIFY longitudeChanged)
    Q_PROPERTY(qreal latitude READ latitude NOTIFY latitudeChanged)
    Q_PROPERTY(int gpsError READ gpsError NOTIFY gpsErrorChanged)

public:
    explicit ExifItem(QDeclarativeItem *parent = 0);

    Q_INVOKABLE void getGpsPointer(QUrl filename);

private:
    int gpsError(){return m_gpsError;}
    qreal latitude(){return m_latitude;}
    qreal longitude(){return m_longitude;}
    void launchMaps(qreal latitude, qreal longitude);

signals:

    void gpsErrorChanged();
    void longitudeChanged();
    void latitudeChanged();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    void positionChanged();

public slots:
    void gpsPositionReady(int error , qreal longitude, qreal latitude);

private:
    long        m_gpsPointer;
    int         m_gpsError;
    qreal       m_latitude;
    qreal       m_longitude;
    ExifHandler*    m_exifHandler;
};
```

Private class to connect with Symbian class

As known, to use Symbian side with signal/slot functionality, mixing private class is needed to connect these two together. This class is pretty simple, it's main function is to create instance of Symbian class and offer signals to be emitted from Symbian side.

Function used to start image reading:

```
void getImageGps(const QString& filename);
```

This does not need to be slot as implementation in Symbian side is derived from ActiveObject, so it sets AO to active state.

```
class ExifHandler: public QObject
{
    Q_OBJECT
public:
    explicit ExifHandler(QObject* parent = 0);
    ~ExifHandler();

    void getImageGps(const QString& filename);

signals:
    void gpsPosition(int error, qreal longitude, qreal latitude);

private:
    CExifSymbian*    d_ptr;

private:
    friend class CExifSymbian;
};
```

Signal

```
void gpsPosition(int error, qreal longitude, qreal latitude);
```

will be emitted from Symbian class.
Error code is zero in successful image reading, otherwise it is Symbian error code.
http://www.developer.nokia.comhttp://www.developer.nokia.com/Community/Wiki/Symbian_OS_Error_Codes

Symbian side

Symbian class is derived from ActiveObject class CActive, by this way we can use class asynchronously. Otherwise many calls would block UI and that is not what we want.

```
class CExifSymbian: CActive
{
public:
    static CExifSymbian* NewL(ExifHandler& aPublicAPI);

    ~CExifSymbian();

    void LoadGPSPosition(const TDesC& aFilename);

public:
    virtual void DoCancel();
    void RunL();

private:
    CExifSymbian(ExifHandler& aPublicAPI);

    void ConstructL();

private:
    enum{
        ENothing = 0,
        ELoadGPSPosition
    };

    TInt        iState;
    ExifHandler& iPublicAPI;
    TFileName   iFilename;
};
```

After class declaration, have a look to most important function. In here images EXIF is read and parsed. Other EXIF datas could be read here too but now focus to GPS coordinates. After reading, signal will be emitted and it is connected in main item.

Be aware the call:

```
TRAP(error, ptr = read->GetTagIdsL(EIfdGps,tags));
```

will leave if tags are not found from image. Error code is the one which is passed to Qt side and code is Symbian OS error code.

```
void CExifSymbian::RunL()
```


(C) Copyright Nokia 2013. All rights reserved.

```
{
    switch(iState){
    case ELoadGPSPosition:{
        RFile file;
        RFs fs;
        TInt error = 0;
        TReal64 m_longitude = 0;
        TReal64 m_latitude = 0;
        TInt tags;
        TUint16* ptr = NULL;
        TBool longitude = ETrue;
        TBool west=ETrue;
        TBool north = ETrue;

        User::LeaveIfError(fs.Connect());
        User::LeaveIfError( file.Open( fs, iFilename, EFileRead ) );
        CleanupClosePushL( file );
        TInt size = 0;
        file.Size(size);

        // Don't read more than 64k
        if ( size > 65536 )
            size = 65536;
        HBufC8* exif = HBufC8::NewL( size );
        CleanupStack::PushL( exif );
        TPtr8 bufferDes( exif->Des() );
        User::LeaveIfError( file.Read( bufferDes, size ) );
        CleanupStack::Pop( exif );
        CleanupStack::PopAndDestroy();
        CleanupStack::PushL( exif );

        CExifRead* read = CExifRead::NewL( exif->Des(),CExifRead::ENoTagChecking | CExifRead::ENoJpeg );
        CleanupStack::PushL( read );

        if(read->IfdExists(EIfdGps)){
            TRAP(error, ptr = read->GetTagIdsL(EIfdGps,tags));
            if(error==KErrNone && ptr!=NULL){

                for(TInt i=1;i<5;i++){
                    const CExifTag* tag = read->GetTagL(EIfdGps,ptr[i]);

                    if(tag){
                        switch(tag->TagInfo().iDataType)
                        {
                        case CExifTag::ETagAscii:
                        {
                            TBuf8<20> tmp;
                            tmp.CopyLC(tag->Data());
                            if(tmp.Find(_L8("n"))!=KErrNotFound || tmp.Find(_L8("s"))!=KErrNotFound ){
                                longitude = EFalse;
                                if(tmp.Find(_L8("n"))!=KErrNotFound)
                                    north = ETrue;
                                else if(tmp.Find(_L8("s")))
                                    north = EFalse;
                            }else{
                                longitude = ETrue;
                                if(tmp.Find(_L8("w"))!=KErrNotFound)
                                    west = ETrue;
                                else if(tmp.Find(_L8("e")))
                                    west = EFalse;
                            }
                        }break;

                        case CExifTag::ETagRational:
                        {
                            TReal64 num = 0;
                            TInt tmpData;
                            TReal64 numerator = 0;
                            TReal64 denominator = 0;
                            const TUint8* rationalData = tag->Data().Ptr();
                            for(TUint c=0;c<tag->TagInfo().iDataCount;c++){
                                numerator = 0;
                                denominator = 0;
                                Mem::Copy(&tmpData, rationalData + ((c * 2) * sizeof(tmpData)), sizeof(tmpData));
                                numerator = tmpData;
                                tmpData = 0;
                                Mem::Copy(&tmpData, rationalData + (((c * 2) + 1) * sizeof(tmpData)), sizeof(tmpData));
                                denominator = tmpData;

                                switch(c){
                                case 0:{
                                    num = (numerator/denominator);
                                    if(longitude)
                                        m_longitude = m_longitude + num;
                                    else
                                        m_latitude = m_latitude + num;
                                }break;
                                case 1:{
                                    num = (numerator/ denominator)/60;
                                    if(longitude)
                                        m_longitude = m_longitude + num;
                                    else
                                        m_latitude = m_latitude + num;
                                }break;
                                case 2:{
                                    num = (numerator/ denominator)/3600;
                                    if(longitude)
                                        m_longitude = m_longitude + num;
                                    else
                                        m_latitude = m_latitude + num;
                                }break;
                                default:break;
                                }
                            }
                        }break;

                        case CExifTag::ETagByte:
                        case CExifTag::ETagShort:
                        case CExifTag::ETagLong:
                        case CExifTag::ETagSlong:
                        case CExifTag::ETagSrational:
                        case CExifTag::ETagUndefined:
                        default:
                            break;
                        }
                    }
                }
                delete ptr;
            }
        }
```


(C) Copyright Nokia 2013. All rights reserved.

```
        }

        CleanupStack::PopAndDestroy( read );
        CleanupStack::PopAndDestroy( exif );
        iState = ENothing;
        if(west)
            m_longitude = m_longitude*-1;
        if(!north)
            m_latitude = m_latitude*-1;
        QT_TRYCATCH_LEAVING(emit iPublicAPI.gpsPosition(error,m_longitude,m_latitude);)
    }break;
    default:break;
    }
}
```

For successful coordinate reading, values are passed back up to QML item which shows these in properties.

Launching maps

In example, QML does not do anything with coordinates even though those are available in properties. Instead, Qt item will launch Maps with navigating to coordinates.

```
void ExifItem::launchMaps(qreal latitude, qreal longitude)
{
    QDesktopServices::openUrl(QUrl("http://m.ovi.me/?c=" + QString::number(latitude) + "," + QString::number(longitude)));
}
```

References

Symbian Exif API

Exif specification

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.